Change configuration

Update cassandra.yamml

Change cqlsh configuration

/data/conf/cqlshrc

### Administration

The main start script is bin/cassandra

As of Cassandra 3, you must use Java 8

Also, you must start as root user, so use sudo bin/cassandra

Start cassandra as a daemon

$CASSANDRA\_HOME/bin/cassandra

Start cassandra in foreground

$CASSANDRA\_HOME/bin/cassandra -f

Error: Could not find or load main class org.apache.cassandra.service.CassandraDaemon

Kill Cassandra daemon

pkill -f CassandraDaemon

# General

Comments?

--

//

/\* foo \*/

# Structures

“if not exists” clause

Only create if doesn’t exist already

### keyspaces

Create keyspace

create keyspace pluralsight with replication = {'class':'SimpleStrategy', 'replication\_factor':3};

create keyspace if not exists pluralsight

Alter a keyspace

Alter keyspace

Turn off durable writes

Alter keyspace …. AND durable\_writes = false

Change replication strategy of a keyspace

Alter keyspace Foo with replication = {‘class’:’SimpleStrategy’,’replication\_factor’:2}

This will require you to run nodetool repair

This will reorganize the data to match the new strategy

remove a keyspace

drop keyspace

### tables

table properties are specified with the “WITH” clause

use pluralsight

create table courses (id varchar primary key);

id will also be the partition key as well

or create table pluralsight.courses….

Primary key can also be specified as a separate line…

create table courses (id varchar), primary key(id)

create table with table property

create table courses (id varchar primary key) with comment = ‘A table of courses’;

Options are specified by including a WITH clause…

Add column

alter table pluralsight.courses add name varchar

Drop column

alter table pluralsight.courses drop title

Rename col or change datatype

TODO

Remove all data from a table

truncate pluralsight.courses

Remove table

drop table pluralsight.courses

Alter table property

alter table courses with comment = ‘Hello there’

Create table with default TTL for all rows

create table reset\_tokens (id varchar PRIMARY KEY, token varchar) WITH default\_time\_to\_live = 10800;

Cassandra doesn’t support dropping a collection column and then readding it.

You need to recreate the entire database.

### Tombstones

A flag that is associated with columns / rows signifying that the data is deleted

Tombstones are partitioned just like any other data written to a cluster

Hinted handoffs, read-repairs, etc are handled for tombstones just like anything else

Periodically purged from the database

It’s important not to purge tombstones too frequently, otherwise data may get resurrected following an attempt to delete partially fails to be written to all replication nodes.

gc\_grace\_seconds specifies the frequency with which tombstones are deleted.

default is 10 days

You do want to remove tombstones periodically, since they affect performance, needing to be skipped over when querying for data.

## Counters

Must live in tables where the only column types other than the PK column is counter.

Creating counters table

create table ratings (

course\_id varchar primary key,

ratings\_count counter,

ratings\_total counter

);

Incrementing a counter

update ratings set ratings\_count = ratings\_count + 1, ratings\_total = ratings\_total + 4 where course\_id = 'node-intro’

This is an upsert, since a column is inserted in the syntax of an update

## Inserting / update data

INSERTs and UPDATEs are the same thing

DELETEs are also basically the same thing

Following are equivilent

insert into users(id, fname, lname) values (‘john-doe’, ‘joe’, ‘doe’);

update users set fname=’mary’, lname=’doe’ where id=’mary-doe’

consistency all

insert into courses(id) values (‘course-intro’)

-- update one partition

update courses set author=’Mike’ where id = ‘courses-cassandra’;

WHERE is required

-- update multiple partitions

update courses set author=’Mike’ where id in (‘courses-cassandra’, ‘foo’)

-- delete row

delete from courses where id = ‘foo’

-- delete column

delete author from courses where id = ‘foo’

-- update value to null

update courses set author = null where id = ‘foo’

-- insert null value

insert into courses (id, author) values (‘node-intro’, null);

-- Automatic deletes columns

update courses using TTL 32400 set reset\_token = ‘13432’ where id = ‘foo’

reset\_token is a handle you can use to subsequently query the TTL info about data.

TTL is specified in seconds

doesn’t delete the entire row.

A row is still left.

-- Automatically delete rows

Must use an insert statement.

insert into reset\_tokens (id, token) VALUES (‘joe’, ‘24432’) using TTL 10800;

row will be removed after TTL

Where a clustering key is present in a table, you must include that key in your insert.

## Selecting data

Very similar to rdbms

Considerations

How many nodes are required to return the data requested

Without a where clause, all nodes will be queried

Aliases are supported

select title as foo from courses where id=’cassandra-developers’

Insert data into a table

insert into courses (id) values (‘cassandra-developers’);

specify a consistency prior to performing an action…

consistency quorum

insert into courses(id) values (‘building-async-restful-services-jersey’);

Where clauses

IN is supported

select courses as foo from courses where id in (‘cassandra-devs’,’node-intro’)

Use with care, as it may require coordinator to visit several nodes to obtain all the data

Limit results with LIMIT clause

select name from courses LIMIT 100;

Select time data was entered

select id, WRITETIME(author) FROM courses

unix timestamp

Stored on a per-column basis

Select token for a given partition-key

select id, TOKEN(id) from courses where id = ‘angular’

output can be mapped back to info output from nodetool ring and describe ring commands

Lets you determine which partition a given row is stored

-- Retrieve TTL for a column value

select TTL(reset\_token) from courses where id = ‘foo’

Data is indexed automatically by partition key.

Data can also be accessed without specifying the partition key in the where-clause, but that requires use of other mechanisms

secondary indexes

Filtering / Sorting

Somewhat constrained in cassandra

You can only sort where partition key is constrained using partition key column

Can’t sort by

Results can only be sorted where the partition key is restricted using an = or IN operator

There is no OR in CQL

You’ll need to perform two separate queries and combine the results outside of Cassandra.

distinct

Can only be applied when all partition key columns are included in the select

## Operations

Math isn’t supported natively in Cassandra

Clients must perform their own math using values returned from cassandra