Cassandra

Intruduction

Cassandra is an open-source decentralized database with scalability and high availability without compromising performance. Cassandra's data model offers the convenience of column indexes with the performance of log-structured updates, strong support for denormalization and materialized views, and powerful built-in caching. Furthermore, Cassandra supports MapReduce and query language.

Main Features 1

Fault Tolerant

Cassandra supports replication across multiple data center. And based on replication strategy, data is automatically replicated to other distributed nodes. When a node fails, it can be replaced without downtime.

Decentralized

Cassandra has its unique ring structure makes every node in the cluster is identical. And there are no single points of failure or network bottleneck.

Durable

Even an entire datacenter goes down, Cassandra still can keep all the data. This is suitable for applications that cannot afford to lose data.

Query Language

CQL provides an API to Cassandra that is simpler than the Thrift API. The Thrift API and legacy versions of CQL expose the internal storage structure of Cassandra. CQL adds an abstraction layer that hides implementation details of this structure and provides native syntaxes for collections and other common encodings.

```
CREATE KEYSPACE research_demo
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

USE reseach_demo;

CREATE TABLE research_table (id uuid, Last text, First text, PRIMARY KEY(id));

INSERT INTO research_table (id, Last, First)
```

```
VALUES ('36b6c939-2459-4c24-a1ed-84269ebcd7a1', 'Anderson', 'Thomas');

SELECT * FROM research_table;
```

Data Model 3

Cassandra is meant to be distributed over multiple machines. This largest level is known as a cluster, and each machine is a node. Every node contains a replica so that errors don't bring down the system. The cluster is arranged in a ring structure.

Keyspaces: A keyspace is the outermost data container in Cassandra, corresponding to the database in relational systems. A keyspace has three basic attributes: replication factor, replica placement strategy, and column families. The replication factor is just the number of machines in the cluster that will receive copies of the same data. The replica placement strategy just determines how replicas are placed in the ring. There are three options, which are not too important in a basic tutorial, so we will address them as needed.

Column families are the meat of keyspaces, and correspond to tables in relational systems. Keyspaces can have one or more column families, and each column family itself contains a collection of rows. Each row contains ordered columns. Unlike a relational database, the schema is not fixed. Column families are defined, but new columns can be added to the family at any time. In addition, each row isn't forced to have all columns defined in the column family.

Cassandra also has the concept of a super column, which stores a map of subcolumns. In Cassandra, column families are stored on disk in individual files. To help performance, columns that you often query together can be stored in a super column to keep them together.

Installation 4

Tarball installation of Cassandra 3.x on Linux-based platform

Prerequisites * Linux-based platform * Lastest version of Java platform * Python 2.7

Procedure In a terminal window

1. Check the version of Java is installed (latest version of Oracle Java 8 is recommended)

   ```
   $ java -version
   ```

2. Download the Cassandra 3.x form Datastax Distribution:

```
$ curl -L http://downloads.datastax.com/datastax-ddc/datastax-ddc-
version_number-bin.tar.gz | tar xz
```

or from Planet Cassandra http://www.planetcassandra.org/cassandra

Please **replace 3.x to the version number you want to install**

3. Untar the file:

```
$ tar -xvzf datastax-ddc-version_number-bin.tar.gz
```

4. To configure Cassandra, go to the install/conf directory:

```
$ cd datastax-ddc-version_number/conf
```

5. Start Cassandra in a single-node cluster:

```
$ cd install_location
$ bin/cassandra ## use -f to start Cassandra in the foreground
```

6. Verify that Cassandra is running:

```
$ cd install_location
$ bin/nodetool status
```

```
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address             Load       Tokens  Owns
Host ID                               Rack
UN  127.0.0.147.66 KB   47.66 KB   256     100%
aaa1b7c1-6049-4a08-ad3e-3697a0e30e10  rack1
```

For installation on a specific opeartion system, please refer installation manual.

Cassandra Python Package

After you have Cassandra installed and running, you need to install the Cassandra libraries for Python. Since Python 2.7 has been installed on Ubuntu or Debian distribution, you won't bother to install Python. However, you do need to install a client driver. This driver works exclusively with the Cassandra Query Language v3 (CQL3) and Cassandra's native protocol. Cassandra 1.2+ is supported.

Installation through pip

pip is the suggested tool for installing packages. It will handle installing all Python dependencies for the driver at the same time as the driver itself. To install the driver:

```
pip install cassandra-driver
```

Verifying your Installation

To check if the installation was successful, you can run:

```
python -c 'import cassandra; print cassandra.__version__'
```

And it should print out something like '3.2.0'.

Configuration

The configuration file of Cassandra is cassandra.yaml, and it is located in the following directories:

- Cassandra package installations: /etc/cassandra
- Cassandra tarball installations: install_location/conf
- DataStax Enterprise package installations: /etc/dse/cassandra
- DataStax Enterprise tarball installations:install_location/resources/cassandra/conf

Configuration properties:

cluster_name

(Default: Test Cluster) The name of the cluster.

listen_address

(Default: locahost) The IP address or hostname that Cassandra binds to for connecting to other Cassandra nodes.

- Generally set to empty (never set to 0.0.0.0). Cassandra uses InetAddress.gerLocalHost() to get the local address from the system.
- For a single node cluster, you can use the default setting.
- You must specify the IP address or host name if Cassandra can't find the correct address listen_interface (Default: eth0) The interface that Cassandra binds to for connecting to other Cassandra nodes. commitlog_directory
- The directory where the commit loa is stored.
- For optimal write performance, place the commit log be on a separate disk partition or a separate physical device from the data file directories.

data_file_directories

The directory location where table data (SSTable) is stored.

Saved_caches_directory

The directory location where table key and row caches are stored.

Disk_failure_policy

(Default: stop) Policy for commit disk failures:

- die
  Shut down gossip and Thrift and kill the JVM, so the node can be replaced.
- stop Shut down gossip and Thrift, leaving the node effectively dead, but can be inspected using JMX.
- stop_commit
  Shut down the commit log, letting writes collect but continuing to service reads (as in pre-2.0.5 Cassandra).
- ignore Ignore fatal errors and let the batches fail.

Auto_snapshot

(Default: true) Enable or disable whether a snapshot is taken of the data before keyspace truncation or dropping of tables. Default setting is advised. 5

Basic Commands

Shell Commands

To begin using Cassandra in the shell, type the command "cqlsh" to enter the command line interface. You can type "help" at any point to get a list of shell commands and help topics. To introduce us to the basics, we'll use a few simple commands to create a sample keyspace and tables.

```
CREATE KEYSPACE "Example" WITH replication =
{'class': 'Simple Strategy', 'replication_factor': 3};
```

This creates a keyspace with the name "Example." The WITH clause specifies properties of the keyspace, in this case replication. The three replica placement strategies are specified with the 'class' attribute, with the options being Simple Strategy, Network Topology Strategy, and Old Network Topology Strategy. The replication factor is specified by 'replication_factor'.

```
CREATE TABLE users {
id int PRIMARY KEY,
username text,
password text
};
```

This creates a table (you can also use CREATE COLUMNFAMILY) named "users." Inside we define the columns for the table in the format "name datatype," and specify PRIMARY KEY for the primary key of the table. The syntax is very similar to that of SQL.

```
Select * from users
```

This is the select statement for Cassandra, which is identical to its SQL counterpart. This will display the requested information (in this case, all columns) in the users table, which lets us verify that the table was created.

```
INSERT INTO users (id, username, password) VALUES(1, testex, aaaaa);
```

The insert statement should also look familiar. In addition to the INSERT INTO and VALUES keywords, there is also a USING keyword to specify additional options. We could then run the previous select statement to verify that the insert was successful.

```
UPDATE users
SET password = 9a2kwqtyf
WHERE username = testex.
```

Much like SQL, we specify the table to update, the column to change, and the condition under which to make the change.

Deleting data is a little bit different than SQL.

```
DELETE username FROM users WHERE id = 1
```

Since Cassandra doesn't require columns to have a value, this is perfectly okay as long as we don't delete the primary key. To delete an entire row, we use the command

```
DELETE FROM users WHERE id = 1
```

Notice how we don't specify a column to delete.

Python Commands

Create a instance of cluster for each Cassandra cluster

```
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('demo')
```

Connected to the demo keyspace and insert a user

```
session.execute("""
    insert into users (lastname, age, city, email, firstname)
  values ('Jones', 35, 'Austin',    'bob@example.com', 'Bob')
""")
```

Now we can retrieve user information from database

```
result = session.execute("select * from users where lastname='Jones' ")[0]
print result.firstname, result.age
Bob 35
```

We also can update user's information

```
session.execute("update users set age = 36 where lastname = 'Jones'")
result = session.execute("select * from users where lastname='Jones' ")[0]
print result.firstname, result.age

Bob 36
```

Deletion

```
session.execute("delete from users where lastname = 'Jones'")
result = session.execute("select * from users")
for x in result: print x.age
```

Task 1

Your task is to design a Cassandra database to hold the following parts information, along with their manufacturer:

Parts Table

Part #	Description	Compatibility	Price	Manufacturer
FRU324534	128G NVMe PCIe M.2 SSD	W540, W541	$127	Samsung
FRU235123	2133MHz DDR3 16GB Memory	DIMM	$70	Samsung
FRU235123	2133MHz DDR3 16GB Memory	DIMM	$65	Corsair
FRU385271	DVD-ROM	T530, T540, W540	$30	Lenovo

Manufacturer Table

Name	Home Page URL	Stock Code	Customer Service Number
Samsung	http://www.samsung.com/us/	SSNLF	1-800-726-7864
Corsair	http://www.corsair.com/en-us	CRSR	1-888-222-4346
Lenovo	http://www.lenovo.com/us/en/	LNVGY	1-855-253-6686

To Do

1. Submit **a sequence of CQL commands** to store the above table of parts and table of manufacturer. (country code of phone number can be ignored. Be careful when you assign the primary key)
2. Submit **a sequence of CQL commands** to retrieve all parts that produced by a given manufacturer.

3. Submit **a sequence of CQL commands** to retrieve parts information sorted by price in descending order for part FRU235123.

Task 2

Construct a class registration system with a rudimentary interface(Command line interface works fine) using a high-level language(Python) that supports the following features.

To Do

1. Add a course(course number, course name, instructor, cap, days of week, periods and classroom) to upcoming quarter
2. Edit course information
3. Delete a course (This may happen due to low number of enrollment. Students who register this course will be free. They need to re-register by themselves, the system will not help them re-register.)

4. Search by course number, course name, instructor, classroom

5. Add a student (username, name, email, standing(FR, SO, JR, SR)) to system
6. Delete a student
7. Edit a student information
8. Search by username, name, email
9. Allow students to register (can only if a course has seat left) and drop courses
10. Track a course's enrollment
11. Track a student's registration

Possible Solution

Task 1

To Do

- CQL commands:

```
CREATE KEYSPACE IF NOT EXISTS inventory WITH REPLICATION =
{'class' : 'SimpleStrategy', 'replication_factor': 1}
/* Create a KEYSPACE in Cassandra if the keyspace name does not exist.
* To simplify, we will just use simple strategy and use only 1 replication factor. */
use inventory; /* switch to 'inventory' keyspace */
CREATE TABLE IF NOT EXISTS parts (
part_number text, description text, compatibility set<text>, price int,
 manufacturer text, PRIMARY KEY((part_number), price));
/* create a parts table */
INSERT INTO parts (part_number, description, compatibility, price, manufacturer)
V ALUES ('FRU324534', '128G NVMe PCIe M.2 SSD', {'W540', 'W541'}, 127, 'Samsung')
IF NOT EXISTS;
INSERT INTO parts (part_number, description, compatibility, price, manufacturer)
VALUES ('FRU235123', '2133MHz DDR3 16GB Memory', {'DIMM'}, 70, 'Samsung') IF NOT EXISTS;
INSERT INTO parts (part_number, description, compatibility, price, manufacturer)
VALUES ('FRU235123', '2133MHz DDR3 16GB Memory', {'DIMM'}, 65, 'Corsair') IF NOT EXISTS;
INSERT INTO parts (part_number, description, compatibility, price, manufacturer)
VALUES ('FRU385271', 'DVD-ROM', {'T530','T540','W540'}, 30, 'Lenovo') IF NOT EXISTS;
/* insert data into part table */
/* This is how parts table looks like */
 part_number | compatibility            | description              | manufacturer | price
-------------+--------------------------+--------------------------+--------------+-------
   FRU235123 |                {'DIMM'} | 2133MHz DDR3 16GB Memory |      Samsung |    70
   FRU235123 |                {'DIMM'} | 2133MHz DDR3 16GB Memory |      Corsair |    65
   FRU385271 | {'T530', 'T540', 'W540'} |                  DVD-ROM |       Lenovo |    30
   FRU324534 |         {'W540', 'W541'} |   128G NVMe PCIe M.2 SSD |      Samsung |   127
```

```
CREATE TABLE IF NOT EXISTS manufacturers
(name text, home_page_url text, NASDAQ_code text,customer_service_number text,
 PRIMARY KEY(name));
/* create manufacturers table */
INSERT INTO manufacturers (name, home_page_url, NASDAQ_code, customer_service_number)
VALUES ('Samsung', 'http://www.samsung.com/us/', 'SSNLF', '800-726-7864') IF NOT EXISTS;
INSERT INTO manufacturers (name, home_page_url, NASDAQ_code, customer_service_number)
VALUES ('Corsair', 'http://www.corsair.com/en-us', 'CRSR', '888-222-4346') IF NOT EXISTS;
INSERT INTO manufacturers (name, home_page_url, NASDAQ_code, customer_service_number)
VALUES ('Lenovo', 'http://www.lenovo.com/us/en/', 'LNVGY', '855-253-6686') IF NOT EXISTS;
/* insert manufacturers' inforamtion into manufacturers table */

SELECT * FROM manufacturers
/* show all manufacuters' info */
```

```
 name    | customer_service_number | home_page_url                | nasdaq_code
---------+-------------------------+------------------------------+-------------
 Corsair |            888-222-4346 | http://www.corsair.com/en-us |        CRSR
 Samsung |            800-726-7864 |   http://www.samsung.com/us/ |       SSNLF
  Lenovo |            855-253-6686 | http://www.lenovo.com/us/en/ |       LNVGY
```

- Search by manufacturer name

```
/* In order to search a non-primary key, you need to create an index on the column */
CREATE INDEX IF NOT EXISTS parts_manu ON parts (manufacturer);
CREATE INDEX IF NOT EXISTS ON parts (price);

/* retrieve all parts info produced by Samsung */
SELECT * FROM parts where manufacturer = 'Samsung' ;
```

```
 part_number | compatibility    | description              | manufacturer | price
-------------+------------------+--------------------------+--------------+-------
   FRU235123 |         {'DIMM'} | 2133MHz DDR3 16GB Memory |      Samsung |    70
   FRU324534 | {'W540', 'W541'} |   128G NVMe PCIe M.2 SSD |      Samsung |   127
```

- Sort by price in descending order for a particular part

```
SELECT * FROM parts WHERE part_number = 'FRU235123' ORDER BY price DESC;
```

```
 part_number | price | compatibility | description              | manufacturer
-------------+-------+---------------+--------------------------+-------------
   FRU235123 |    70 |      {'DIMM'} | 2133MHz DDR3 16GB Memory |      Samsung
   FRU235123 |    65 |      {'DIMM'} | 2133MHz DDR3 16GB Memory |      Corsair
```

Task 2

To install cassandra python driver cassandra-driver, please follow the instruction in previous pages.

Once you have the driver installed, you are ready to begin. The rest of commands, I will use interactive shell. And '>>>' indicate a python command, rests are comments.

```
>>> from from cassandra.cluster import Cluster
>>> cluster = Cluster()
```

Right now, the assumption is that the connection is on local machine. You also can connect to multiple nodes:

```
>>> cluster = Cluster(['137.112.40.139', '137.112.104.138'])
# This connect 2 IPs (our node 1 and node 2)
```

If you already have a KEYSPACE in Cassandra, you can directly connect to the KEYSPACE

```
>>> session = cluster.connect('task2')
```

Or, you can just connect to Cassandra and create your KEYSPACE

```
>>> session = cluster.connect()
>>> session.execute("""CREATE KEYSPACE IF NOT EXISTS task2
          WITH replication = {'class':'SimpleStrategy', 'replication_factor':3};""")
>>> session.set_keyspace('task2')
>>> session.execute('USE task2')
```

You can also use 'sek_keyspace()' or executing "USE " to switch a keyspace

In Cassandra python driver, most executing ccommands are just string parsing. You execute the exactly same CQL query in python as in Cassandra shell command.

```
>>> session.execute("""CREATE TABLE courses
(c_number text PRIMARY KEY, c_name text, instructor text, cap int,
days_of_week set<text>, period set<int>, room text);""")
```

Create TABLE courses and set PRIMARY KEY and data tpyes for each columns.

```
>>> session.execute("""INSERT INTO courses(c_number, c_name,
instructor, cap, days_of_week, period, room) VALUES('CSSE433-01',
'Advanced Database Systems','mohan', 24, {'M','T','R','F'},
{4}, 'O157')""")
>>> session.execute("""INSERT INTO courses(c_number, c_name,
instructor, cap, days_of_week, period, room)
VALUES( %s, %s, %s, %s, %s, %s, %s)""",('CSSE333-01',
'Intro to Database Systems','mohan', 24, {'M','W','F'}, {1,2}, 'O269'))
```

Two different ways to insert data into table in KEYSPACE. One is to give the values directly, and the other one is parse the value via string format. One thing to pay attention is in the second way, even if your desire data type is integer, you still need a string format '%s' indicator.

c_number	c_name	cap	days_of_week	instructor	period	room
CSSE333-01	Intro to Database Systems	24	{'F', 'M', 'W'}	mohan	{1, 2}	O269
CSSE433-01	Advanced Database Systems	24	{'F', 'M', 'R', 'T'}	mohan	{4}	O157

Now assume we have another CSSE333, and we have wrong instructor

```
>>> session.execute("""INSERT INTO courses(c_number, c_name, instructor,
cap, days_of_week, period, room)
VALUES( %s, %s, %s, %s, %s, %s, %s)""",('CSSE333-02',
'Intro to Database Systems','mohan', 24, {'M','W','F'}, {5,6}, 'O269'))
```

c_number	c_name	cap	days_of_week	instructor	period	room
CSSE333-01	Intro to Database Systems	24	{'F', 'M', 'W'}	mohan	{1, 2}	O269
CSSE333-02	Intro to Database Systems	24	{'F', 'M', 'W'}	mohan	{5, 6}	O269
CSSE433-01	Advanced Database Systems	24	{'F', 'M', 'R', 'T'}	mohan	{4}	O157

In python, we can edit instructors information by INSERT or UPDATE

```
>>> session.execute("""INSERT INTO courses(c_number, c_name, instructor,
cap, days_of_week, period, room)
VALUES( %s, %s, %s, %s, %s, %s, %s)""",('CSSE333-02', 'Intro to Database Systems',
'wilkin', 24, {'M','W','F'}, {5,6}, 'O269'))
>>> session.execute("UPDATE courses SET instructor = 'wilkin'
where c_number = 'CSSE333-02' ")
```

Both queries give us the following result:

c_number	c_name	cap	days_of_week	instructor	period	room
CSSE333-01	Intro to Database Systems	24	{'F', 'M', 'W'}	mohan	{1, 2}	O269
CSSE333-02	Intro to Database Systems	24	{'F', 'M', 'W'}	wilkin	{5, 6}	O269
CSSE433-01	Advanced Database Systems	24	{'F', 'M', 'R', 'T'}	mohan	{4}	O157

Using the 'INSERT' way, as long as your primay key is consistent, you can modify other information. Using the 'UPDATE' way, is like what we did in relational database. We specify a table we want to update, set the value and give the condition.

Now, assume we will not offer CSSE333-02, we can delete it.

```
>>> session.execute("DELETE FROM courses where c_number = 'CSSE333-02'")
```

c_number	c_name	cap	days_of_week	instructor	period	room
CSSE333-01	Intro to Database Systems	24	{'F', 'M', 'W'}	mohan	{1, 2}	O269
CSSE433-01	Advanced Database Systems	24	{'F', 'M', 'R', 'T'}	mohan	{4}	O157

And you can also delete a column.

```
>>> session.execute("DELETE room FROM courses WHERE c_number = 'CSSE333-01'")
```

c_number	c_name	cap	days_of_week	instructor	period	room
CSSE333-01	Intor to Database Systems	24	{'F', 'M', 'W'}	mohan	{1, 2}	null
CSSE433-01	Advanced Database Systems	24	{'F', 'M', 'R', 'T'}	mohan	{4}	O157

Cassandra does take 'null' string as a string. In our trail, if we delete room info for course CSSE333-01, it will return a null value. While we update room info as a 'null' string, the data type of room info of CSSE333-01 is going to be a string.

In cassandra, we cannot search on a non primay key directly. In order to search, we need to create index on these non primay key columns

```
>>> result = session.execute("select * from courses where instructor = 'mohan'")
InvalidRequest: code=2200 [Invalid query]
message="No supported secondary index found for the non primary
key columns restrictions"
```

An error will occur if we want to search on a non primary key without index.

```
>>> session.execute("CREATE INDEX ON courses (c_name)");
>>> session.execute("CREATE INDEX ON courses (instructor)");
>>> session.execute("CREATE INDEX ON courses (room)");
```

After creating index, we can search on thess columns. However, since our primary key is course number, the result is ordered by the primay key.

```
>>> result = session.execute("select * from courses where instructor = 'mohan'")
>>> for row in result:
...     print row

Row(c_number=u'CSSE333-01', c_name=u'Intro to Database Systems',
cap=24, days_of_week=SortedSet([u'F', u'M', u'W']), instructor=u'mohan',
period=SortedSet([1, 2]), room=u'O269')
Row(c_number=u'CSSE433-01', c_name=u'Advanced Database Systems',
cap=24, days_of_week=SortedSet([u'F', u'M', u'R', u'T']), instructor=u'mohan',
period=SortedSet([4]), room=u'O157')
```

Cassandra python drive also support **Object Mapper**. This is migrated from legacy cqlengine, and now integrated into this driver. So you don't need to install additional package to use this feature.

In cqlengine, a CQL table is represented by a model. This model is a python class and defines basic properties and columns for a table. Each column in the model maps to the column in the CQL table. As CQL table, you need at least one primary key column and one non-primary key column. The requirement is the same in a model. The order how you define your columns in model is the same order they are defined in the corresponding table.

Now we are going to use this model to create table in CQL.

```
>>> from cassandra.cqlengine import columns
>>> from cassandra.cqlengine.models import Model
>>> class students(Model):
      username = columns.Text(primary_key=True)
      first_name = columns.Text()
      last_name = columns.Text()
      email = columns.Text()
      standing = columns.Text()
```

Now we define a model. Next step is to setup connection to the database and sync the model/table to the database.

```
>>> from cassandra.cqlengine import connection
>>> from cassandra.cqlengine.management import sync_table
>>> connection.setup(['137.112.40.139'],'task2')
>>> sync_table(students)
```

Now a table has been created in our KEYSPACE, and we can verify in cqlsh

```
> describe tables
students  courses
```

And we can insert some rows

```
>>> student1 = students.create(username = 'xuez', first_name = 'Zhihao',
last_name = 'Xue', email = 'xuez@rose-hulman.edu', standing = 'GR')
>>> student2 = students.create(username = 'wuj', first_name = 'Jiaren',
last_name = 'Wu', email = 'wuj@rose-hulman.edu', standing = 'SR')
>>> student3 = students.create(username = 'raspst', first_name = 'Steven',
last_name = 'Rasp', email = 'raspst@rose-hulman.edu', standing = 'JR')
```

Now the table looks like:

username	email	first_name	last_name	standing
xuez	xuez@rose-hulman.edu	Zhihao	Xue	GRAD
raspst	raspst@rose-hulman.edu	Steven	Rasp	JR
wuj	wuj@rose-hulman.edu	Jiaren	Wu	SR

We can count how many rows are in this table:

```
>>> students.objects.count()
3
```

Like we can use query to retrieve data in CQL shell, we also can retrieve objects in Python. This is accomplished with QuerySet objects.

Retrieving all objects from students table using .all() method.

```
>>> all_students = students.objects.all()
>>> for student in all_students:
    print student

students <username=xuez>
students <username=raspst>
students <username=wuj>
```

If your database is large and you want to limit number of records retrieved, you can use .limit() method to limit the number of results. When use .limit(v) method Cassandra will return the first v number in the table.

```
>>> student4 = students.create(username = 'gaysojj', first_name = 'Joshua',
last_name = 'Gayso', email = 'gaysojj@rose-hulman.edu', standing = 'JR')
>>> for student in students.objects().limit(2):
    print student.username
gaysojj
xuez
```

Sometimes, we only need some fields of an object, using .filter() method can help us achieve this. As mentioned before, we need to create an index on non-primary key column, otherwise, the database will complain about this action.

To retrieve student whose username is 'xuez' and count how many records we have

```
>>> filtered_students = students.objects.filter(username ='xuez')
>>> filtered_students.count()
1
>>> for student in filtered_sutdents:
    print student
students <username=xuez>
```

IMPORTANT!!! So far, we did not find how to create an index using cqlengine. So if you want to create an index on a non-primary key column, you must specify when you create the model! Even you create an index using cqlsh, cqlengine seems not be happy with that.

So, we have to drop the students table and recreate.

Create new model with index

```
>>> class students(Model):
      username = columns.Text(primary_key=True)
      first_name = columns.Text(index=True)
      last_name = columns.Text(index=True)
      email = columns.Text(index=True)
      standing = columns.Text(index=True)
```

After recreate table and insert test data, now we can filter on non-primary key column.

```
>>> filtered_students = students.objects.filter(standing='JR')
>>> filtered_students.count()
2
```

This retrieve all students whose standing is 'JR' To get student's username from retrieved data

```
>>> for student in filtered_students:
  print student.username

gaysojj
raspst
```

Filtering Operators
Like MongoDB, Cassandra also supports operators to filter data by appending a ___< op > to the field name on filtering call.

in a range (___in)
> (___gt)
>= (___gte)
< (___lt)
<= (___lte)
CONTAINS (___contains)

Delete
To delete a record, we can simply call .delete() method.

```
>>> students(username = 'gaysojj').delete()
```

delete student whose username is 'gaysojj'

Update
We can update a column value by specifying a constraint

```
>>> student4 = students.create(username = 'gaysojj', first_name = 'Joshua',
last_name = 'Gayso', email = 'gaysojj@rose-hulman.edu', standing = 'SR')
```

username	email	first_name	last_name	standing
gaysojj	gaysojj@rose-hulman.edu	Joshua	Gayso	SR

```
>>> students(username='gaysojj').update(standing = 'JR')
students(username='gaysojj', first_name=None, last_name=None,
email=None, standing='JR')
```

Connect to a table in KEYSPACE but not created by model
We can use named tables to querying a table without creating a model. This is helpful especially when you are not familiar with a database.

```
>>> from cqlengine.connection import setup
>>> from clqengine.named import NamedTable
>>> setup(['137.112.40.139'], 'task2')
```

The host ip address must be a list. Even thought the documentation gives a string, Cassandra will send you an error if you do so.

```
>>> courses.objects()[0] # retrieve the first course in the courses table
{u'days_of_week': SortedSet([u'F', u'M', u'W']), u'instructor': u'mohan',
u'c_name': u'Intro to Database Systems', u'room': u'0269', u'cap': 24,
u'period': SortedSet([1, 2]), u'c_number': u'CSSE333-01'}
```

If you call a filter operation on both Named Tables and Model class, you will get two different types of objects returned.

```
>>> courses.objects.filter(room = '0269') # Named table filter operation
<cqlengine.query.SimpleQuerySet object at 0x7fcfa0cf1f90>
# this is a SimpleQuereySet
>>> students.objects.filter(standing = 'SR') # Model class
<cassandra.cqlengine.query.ModelQuerySet object at 0x7fcfa0160050>
# this is a ModelQuerySet
```

Batch Queries

Batch queries combines multiple data modification operation such as insert, update and delete into a single logical operation. Batching multiple statements will short network exchanges between client/server and server coordinator/replicas. However, batches is not a good idea to optimize performance. It is useful to synchronize data to tables is a legitimate operation. When execute a batch query, if a row is out of the database and you are trying to read it, this batch query will fail.

create a new table to record students' registration

```
>>> class registrations(Model):
    c_number = columns.Text(primary_key=True)
      username = columns.Text(primary_key=True)

>>> sync_table(registrations)
```

And we can use batch query in two ways. The first way is to use session and execute the cql query. And the other way is to use cqlengine BatchQuery Class.

```
>>> from cassandra.cluster import Cluster
>>> cluster = Cluster()
>>> session = cluster.connect('task2')
>>> batch.add("INSERT INTO registrations (c_number, username)
VALUES (%s, %s)"),('CSSE433-01', 'xuez')
>>> batch.add("INSERT INTO registrations (c_number, username)
```

```
VALUES (%s, %s)"),('CSSE433-01', 'wuj')
>>> batch.add("INSERT INTO registrations (c_number, username)
VALUES (%s, %s)"),('CSSE433-01', 'raspst')
>>> session.execute(batch)
```

```
>>> from cqlengine import BatchQuery
>>> # using a context manager
>>> with BatchQuery as b:
        reg1 = students.create(c_number = 'CSSE433-01', username = 'raspst')
        reg2 = students.create(c_number = 'CSSE433-01', username = 'wuj')
        reg3 = students.create(c_number = 'CSSE433-01', username = 'xuez')

>>> # manually
b = BatchQuery()
reg1 = students.create(c_number = 'CSSE433-01', username = 'raspst')
reg2 = students.create(c_number = 'CSSE433-01', username = 'wuj')
reg3 = students.create(c_number = 'CSSE433-01', username = 'xuez')
b.execute()
```

c_number	username
CSSE433-01	raspst
CSSE433-01	wuj
CSSE433-01	xuez

Scalability In Cassandra

The Apache Cassandra database is an open source, distributed, column-based, NoSQL database management system. It claims to provide linear scalability, low latency, and fault-tolerance both locally and on the cloud. It not only provides replication on clusters, it also supports replications across multiple data centers.

Cassandra clusters work in a peer-to-peer (or "master-less") ring structure, which means that there is no master or slave nodes across the ring structure. Each node in the ring has the same role and responsibility. And all nodes communicate with each other via a gossip protocol. Furthermore, each ring structure can be a node on a higher level ring.

Cassandra is linear scalable which mean that the capacity can be increased by adding new nodes or data centers. It scales up and down automatically without any changes in your application's source because there are no "special" nodes. . The overall architecture will depend on whether the simple strategy or network topology strategy is used. Under the simple strategy, sharding will automatically be handled by Cassandra's partitioner. The partitioner by default will randomly hash data into each node, but can be configured to follow an order. So each

node would end up with the same amount of data. We can also set a replication factor, which would produce copies of each row. So if we have three nodes and a replication factor of two, each node ends up with 33% of the data based on the sharding, but then ends up getting copies from the previous node in the ring, so that each node should end up holding 66% of the data.

The network topology strategy can lead to more complex options. Each cluster of nodes would be a data center, and the data centers would form a larger ring. We would then have options like sharding between data centers, so that one data center ends up with all the data for, say, a certain geographic area, and each node in that data center would hold a replica of the data. Since we don't have a large area to cover, we currently plan to stick to the simple strategy, but if we were to scale up we would want to switch to the network topology strategy as we got bigger.

We also have to consider the impact of the read and write level. These values determine how many nodes we have to be able to access to successfully perform an operation. The options Cassandra offers for read and write level are one, quorum, and all. For level one, the operation only has to be able to successfully perform the operation on one node for the operation to succeed. This means that more nodes can fail before the application experiences problems, but also makes the data only eventually consistent, since the data will get sent to any nodes that need it, but not necessarily at the time of write since a node could be down. Quorum means that a majority of nodes need to respond to the operation for it to be considered successful. All means that every node involved needs to respond. The last two lead to consistent reads for our three node cluster, since under a majority, two of the three would need to write data, and so the data is guaranteed to be there, while for a consistency level of all, all three nodes would get the write from the very beginning

This brief tutorial will teach you set up a 3 node cluster with Cassandra.

1.You need to know all the IP address of the node. If you are using Ubuntu, following command will show the network interfaces of your machine.

```
/sbin/ifconfig
```

The address of eth0 is the IP address we are going to use.

For example, there are 3 node with following IP address:

Node 1: 192.0.0.1 Node 2: 192.0.0.2 Node 3: 192.0.0.3

2.Before we begin to configure each node, you should close Cassandra in all the nodes.

```
$ sudo ps auwx | grep cassandra
```

replace PID with the PID of cassandra

```
$ sudo kill -9 PID
```

3 Find the location of cassandra.yaml file.

```
$ sudo vim /etc/cassandra/cassandra.yaml
```

The information that you need to change is same for all nodes. Here is the example configuration for node 1.

```
seed_provider:
 - seeds: "192.0.0.1, 192.0.0.2, 192.0.0.3"

# Setting listen_address to 0.0.0.0 is always wrong.
listen_address: 192.0.0.1

# Address to broadcast to other Cassandra nodes
broadcast_address: 192.0.0.1

rpc_address: 0.0.0.0

broadcast_rpc_address: 192.0.0.1
```

After you configured all the node, start Cassandra and you are good to go.

```
$ sudo service Cassandra start
```

You can verify your setup by running the command below on node 1

```
$ sudo nodetool status
```

Issues we ran into while using Cassandra

1. Search by a non primary key column.
   When we tried to search by a non primary key column without an index, Cassandra cannot execute this query. In order to search by a non primary key column, we need to create an index on this column.

2. Create index in cqlengine.
   If you want to use an index, you may need to create the index when you create the model. Otherwise, we didn't find a way to create index after you initiate the model. This requires us design the model carefully.

3. Batch Query in cqlengine.
   Thought the official document said you need a .batch(b) method when you want to use a batch query, it is unnecessary to do so. Actually, if you do as the document says, Cassandra will give you an error instead.

Reference

1. Cassandra Home Page
2. Cassandra Python Driver
3. How to Configure Multiple Nodex