# Apache Cassandra

## Objectives

- Gain hands-on experience in using Apache Cassandra, an open-source, distributed, NoSQL database management system
- Learn how to design Cassandra data models

## 1. Setting up the environment

Download the latest Apache Cassandra release (apache-cassandra-3.10-bin.tar.gz) from http://cassandra.apache.org/download/ and extract it:

```
$ tar -zxf apache-cassandra-3.10-bin.tar.gz
```

Verify that JAVA_HOME is set appropriately (typically, JAVA_HOME=/usr)

Start the Cassandra server from the installation directory:

```
$ bin/cassandra -f
```

You should now have a Cassandra cluster up and running; the cluster contains a single node.

## 2. Using the CQL shell

Open a new terminal window (leaving Cassandra running), go to the Cassandra installation directory and type:

```
$ bin/cqlsh
```

You can now use the shell to manage Cassandra. Type HELP to see the list of available commands. See [1] for details on the CQL syntax.

Create a new keyspace and connect to it:

```
CREATE KEYSPACE mykeyspace WITH REPLICATION =
{'class':'SimpleStrategy','replication_factor':1};
USE mykeyspace;
```

Create a new table in the keyspace:

```
CREATE TABLE users (user_id int, name text, email text, PRIMARY KEY
(user_id));
```

Get information about the created table and all tables:

```
DESCRIBE TABLE users;
DESCRIBE TABLES;
```

Insert a row into the table:

```
INSERT INTO users (user_id, name, email) VALUES (101, 'john',
'john@example.org');
```

Insert a few rows as above. What happens if two rows have the same *user_id*?

Display all the rows:

```
SELECT * FROM users;
```

Delete columns and rows:

```
DELETE email FROM users WHERE user_id = 101;
DELETE FROM users WHERE user_id = 101;
```

Update data:

```
UPDATE users SET email='john@example.com' WHERE user_id=101;
```

Add a new column to the table with a type *set<text>* and update its value:

```
ALTER TABLE users ADD hobbies set<text>;
UPDATE users SET hobbies = hobbies + {'fishing','sleeping'} WHERE user_id =
101;
UPDATE users SET hobbies = hobbies - {'fishing'} WHERE user_id = 101;
```

Now, try the following query:

```
SELECT email FROM users WHERE name='john';
```

Why doesn't this work? To solve this issue, you can add "ALLOW FILTERING" in the SELECT statement or you can create an index on the *name* column. For example, try:

```
CREATE INDEX name_index ON users(name);
```

The limitation is that filtering or using indices requires querying every node in the cluster. A better option in Cassandra is to create a new table that allows direct lookup (i.e., duplicating or denormalising data). Typically, the new table must be managed manually (e.g., one must manually insert data into the new table). In some cases (see ([1]), we can use *Materialized Views* that automate such management:

```
CREATE MATERIALIZED VIEW users_by_name AS SELECT * FROM users
WHERE name IS NOT NULL AND user_id IS NOT NULL
PRIMARY KEY (name, user_id);
```

You can now perform the previous query on *users_by_name*. Insert some new rows in the table *users* and verify that they are automatically added into *users_by_name*.

A useful *cqlsh* command for importing/exporting selected columns to/from a CSV file is:

```
COPY users (user_id, email, hobbies, name) TO 'temp.csv';
COPY users FROM 'temp.csv';
```

## 3. Searching messages

Create a new table that holds messages published by a specific user (a user can publish multiple messages):

```
CREATE TABLE messages (msg_id int, published_on timestamp, body text,
user_id int,PRIMARY KEY (msg_id) );
```

Insert a few rows into this table. We now want to find all messages published by a given *user-id* ordered by timestamp (latest message first). Propose a solution and test the result.

## 4. Searching movies

Import the movielens dataset in Cassandra [2]. The easiest way to do this is to use the CDM tool [3]. Download CDM, make it executable, and put it in the path. The program assumes that *cqlsh* is also in the path. You can then run:

$ cdm install movielens

This will import the dataset in Cassandra and you can manage it using *cqlsh*.

Add any necessary tables and provide CQL statements that allow us to perform the following queries:

1. Find the movie ID for a given movie title
2. Find the ratings for a given movie ID, ordered from higher to lower ranking
3. Find the highest-rated movie for a particular user
4. Find the minimum/maximum rating for a given movie ID
5. Find the Sci-Fi/Thriller movies with the highest average rating

## 5. Going to the cloud

The goal is to set up a multi-node Cassandra cluster using Amazon EC2 nodes. Sign in to your AWS account and start 3 virtual instances on EC2. Install Java and Cassandra in each instance and edit the configuration file (/conf/cassandra.yaml). You can find detailed tutorials at [4,5].

Verify that your cluster works normally. You can now insert some data and verify its location in the cluster using nodetool (*nodetool endpoints*). Modify the replication factor using the corresponding CREATE KEYSPACE parameter. Modify the consistency level using the cqlsh CONSISTENCY command. Experiment with removing nodes from your cluster using nodetool (*nodetool -h node1_IP decommission)*. You can enable tracing with the cqlsh TRACING command.

*Important*: Don't forget to clean up everything in AWS.

### Some links

[1] https://cassandra.apache.org/doc/latest/cql/index.html

[2] http://files.grouplens.org/datasets/movielens/ml-latest-small.zip

[3] https://github.com/rustyrazorblade/cdm

[4] http://datascale.io/how-to-create-a-cassandra-cluster-in-aws-part-2/

[5] Cassandra 2.x Cluster on AWS EC2,
https://gist.github.com/diegopacheco/08dc5b9f94ef3d788c95c924b6d35f40