|  |  |
| --- | --- |
| 1. | Start Cassandra |
| 1.1 | make sure vagrant is installed: http://www.vagrantup.com/downloads |
| 1.2 | install virtualbox: https://www.virtualbox.org/wiki/Downloads |
| 1.3 | install git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git |
| 1.4 | download the project   * go to a folder of your choice via command line window * execute the command: *git clone https://github.com/tomvdbulck/training-no-sql.git* |
| 1.5 | start up the vagrant box   * navigate to the redis/vagrant folder: *cd cassandra/vagrant* * enter the command: *vagrant up* |
| This will now download and install the vagrant box, and download ,install and start the cassandra server. This might take a while | |
| 1.6 | The cassandra server will start automatically, we will connect to it with the CQL-shell   * enter the command: vagrant ssh * you will now go into the vagrant box * navigate to : apache-cassandra-2.1.9/bin/ * start the CQL-shell by typing : ./cqlsh   You should see something like this :  cqlsh start |
| 2 | CRUD (Create Read Update Delete) |
| 2.1 | Create a keyspace |
| CREATE KEYSPACE demo WITH replication = {'class':'SimpleStrategy', 'replication\_factor':3}; | |
|  | Verify the Keyspace was created |
| DESCRIBE KEYSPACES; | |
|  | you should see something like this |
| system\_traces system demo | |
|  | switch to the “demo” keyspace |
| USE demo; | |
|  | see which tables are in the keyspace |
| DESCRIBE TABLES; | |
|  | you should see something like this |
| <empty> | |
| 2.2 | Create a table |
|  | CREATE TABLE users (user\_id int PRIMARY KEY,fname text,lname text); |
|  | Verify the table was created |
|  | DESCRIBE TABLE users; |
|  |  |
| 2.3 | Inserting and reading data |
|  | INSERT INTO users (user\_id, lname) VALUES (3333, 'De Bruyne'); |
|  | Let’s see what’s in the table :   * select \* from users ; |
|  | you should see something like this :   * select 1 user |
|  | Let’s insert a second user :   * INSERT INTO users (user\_id, fname, lname) VALUES (4444, 'Tom', 'Van Den Bulck');   And check what’s in the table   * select \* from users ; |
|  | you should see something like this :  select 2 users.png |
|  | Lets do an upsert :   * INSERT INTO users (user\_id, fname, lname) VALUES (3333, 'Chris', 'De Bruyne');   and check the outcome :   * select \* from users;   you should see something like this :  upsert.png  An insert automatically becomes an update if the record already exists. |
|  | Let’s insert a second user :   * INSERT INTO users (user\_id, fname, lname) VALUES (4445, 'Tom', 'Time To Live') USING TTL 5;     After 5 seconds the row with time to live will be gone. |
| 2.5 | updating data |
|  | UPDATE users USING TTL 5 SET fname = 'null' WHERE user\_id = 3333;    After 5 seconds the value revert back to null;  Please note: after TTL the column will be emptied => not reset to the previous value |
| 2.5 | deleting data |
|  | Let’s delete a user :   * DELETE FROM users WHERE user\_id = 4444;   and verify :   * select \* from users;   you should see something like this :  delete 1 user.png |
|  | You can also delete all data from a table   * TRUNCATE users ; |
|  | Please note that a delete is not immediately a delete.  Cassandra places a tombstone (marker) to indicate a column has been deleted.  When retrieving the values - all columns / rows indicated by a tombstone is ignored and the values with the most recent timestamps are shown.  Tombstones are kept by default for 10 days.  They allow for propagation of updates to the other nodes. |
|  |  |
| 3.0 | Compound primary keys |
|  | CREATE KEYSPACE stockwatcher WITH replication = {'class':'SimpleStrategy', 'replication\_factor':3}; |
| 3.1 | Create a table with a compound primary key   * CREATE TABLE stockwatcher.WatchListItem (  watchlist\_id TIMEUUID,  stock\_symbol VARCHAR, start\_price DECIMAL, created TIMESTAMP, PRIMARY KEY (watchlist\_id, stock\_symbol) );   and verify the creation :   * SELECT \* FROM stockwatcher.watchlistitem ; * you should see something like this   select empty watchlistitem.png  Take note of the different colors for the primary key (partition key), clustering column and simple columns. |
| 3.2 | Ordering in tables and queries |
|  | First load some data.  Perform the following command :   * SOURCE '/vagrant/load-table-watchlistitem.cql';   And verify the contents of the table   * SELECT \* FROM stockwatcher.watchlistitem ;   select loaded watchlistitem.png |
|  | Query the table with stock\_symbol descending   * SELECT \* FROM stockwatcher.watchlistitem WHERE watchlist\_id = ca58ee3c-d8a7-11e2-a440-85054b6a0b99 ORDER BY stock\_symbol DESC ;   You should see something like this:  watchlistitem-order-by-stock_symbol.png  A WHERE clause can only be executed on the PK => and it must be an equals statement.  Ordering can only be done on columns in the primary key and after a WHERE or IN statement has been executed on the partition key. |
|  |  |
| 3.4 | Limitations of the WHERE clause => it must be exact values  You can use the TOKEN function together with a condition operator.  Which only works on the partition key.   * SELECT \* from users where TOKEN(user\_id) < 4000   Will result in: |