HARMANTEPE Melis

TP2 - Cassandra

CREATE THE KEYSPACE

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

CREATION OF THE TABLE

```
CREATE TABLE videos (
id int, name text, runtime int,
year int, PRIMARY KEY (id)
);
```

INSERTION

```
INSERT INTO videos (id,name, runtime, year) VALUES (1, ‘Insurgent',119,
2015) ;
INSERT INTO videos (id,name, runtime, year) VALUES (2,
'Interstellar',98, 2014) ;
INSERT INTO videos (id,name, runtime, year) VALUES (3, 'Mockingjay',122,
2014) ;
```

QUERYING

```
select count(*) from videos;
select * from videos;
select * from videos where name = 'Insurgent' ALLOW FILTERING;

Create index IndexName on KeyspaceName.TableName(ColumnName);
Create index yearIndex on demoVideo.videos(year);
```

Pour filter les données on doit ajouter “ALLOW FILTERING” à la fin de notre requête. On peut créer un index aussi mais ça changera rien à notre requête car cassandra le crée aussi lui même, on donne juste le nom qu’on veut.

```
select * from videos where year > 2014 ALLOW FILTERING;
```

PHYSICAL STORAGE
PARTITIONED STORAGE
IS THIS A SOLUTION? TRY IT

```
CREATE TABLE videos_by_name_year (
             ... name text,
             ... runtime int,
             ... year int,
             ... PRIMARY KEY ((name,year));

Notre primary key devient le couple de (name,year).

INSERT INTO videos_by_name_year (name, runtime, year) VALUES
('Insurgent',119, 2015) ;
INSERT INTO videos_by_name_year (name, runtime, year) VALUES
('Interstellar',98, 2014) ;
INSERT INTO videos_by_name_year (name, runtime, year) VALUES
('Mockingjay',122, 2014) ;
```

```
cqlsh:demovideo> select count(*) from videos;

 count
-------
     3

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:demovideo> select * from videos;

 id | name         | runtime | year
----+--------------+---------+------
  1 |    Insurgent |     119 | 2015
  2 | Interstellar |      98 | 2014
  3 |   Mockingjay |     122 | 2014

(3 rows)
cqlsh:demovideo> select * from videos where name = 'Insurgent' ALLOW FILTERING;

 id | name      | runtime | year
----+-----------+---------+------
  1 | Insurgent |     119 | 2015

(1 rows)
cqlsh:demovideo> select * from videos where year > 2014 ALLOW FILTERING;

 id | name      | runtime | year
----+-----------+---------+------
  1 | Insurgent |     119 | 2015

(1 rows)
cqlsh:demovideo>
```

QUERIES

```
select * from videos_by_name_year where name = 'Insurgent' and year = 2015;
select * from videos_by_name_year where name = 'Interstellar' ALLOW FILTERING;
select * from videos_by_name_year where year = 2014 ALLOW FILTERING;
```

Quand on utilise qu'un seul element de notre couple de Primary Key, il faut activer "allow filtering", pourtant quand on veut filter sur les deux éléments (name et year) pas besoin de faire ça.

```
cqlsh:demovideo> select * from videos_by_name_year where name = 'Insurgent' and year
 = 2015;

 name      | year | runtime
-----------+------+---------
 Insurgent | 2015 |     119

(1 rows)
cqlsh:demovideo> select * from videos_by_name_year where name = 'Interstellar';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"
cqlsh:demovideo> select * from videos_by_name_year where name = 'Interstellar' ALLOW
 FILTERING;

 name         | year | runtime
--------------+------+---------
 Interstellar | 2014 |      98

(1 rows)
cqlsh:demovideo> select * from videos_by_name_year where year = 2014;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"
cqlsh:demovideo> select * from videos_by_name_year where year = 2014 ALLOW FILTERING
;

 name         | year | runtime
--------------+------+---------
 Interstellar | 2014 |      98
   Mockingjay | 2014 |     122

(2 rows)
cqlsh:demovideo>
```

HARMANTEPE Melis

CASSANDRA-UPSERTS

```
INSERT INTO videos_by_name_year (name , year , runtime) VALUES
('Insurgent',2015, 127) ;
```

Cette requête mis à jour la ligne avec la vidéo Insurgent en changeant son “runtime” par 127. Avant son runtime était 119. Comme c’est une mise à jour, le nombre de lignes de notre table ne change pas.

```
cqlsh:demovideo> INSERT INTO videos_by_name_year (name , year , runtime) VALUES ('In
surgent',2015, 127) ;
cqlsh:demovideo> SELECT count(*) from videos_by_name_year
             ... ;

 count
-------
     3

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:demovideo> select * from videos_by_name_year;

 name         | year | runtime
--------------+------+---------
 Interstellar | 2014 |      98
   Mockingjay | 2014 |     122
    Insurgent | 2015 |     127

(3 rows)
```

HARMANTEPE Melis

CLUSTERING COLUMNS

```
CREATE TABLE videos_by_year ( id int,
             ... name text,
             ... runtime int,
             ... year int,
             ... PRIMARY KEY ((year), name ) );
```

Notre primary key est seulement **(year)** mais on associe id et runtime à name. Le changement des clés primaires résulte en le changement de la structure des données. L'avantage de ça c'est qu'on a moins de lignes qui représente les mêmes données. Quand on fait la requête on aura le même résultat.

```
cqlsh:demovideo> select * from videos_by_name_year;

 name         | year | runtime
--------------+------+---------
 Interstellar | 2014 |      98
   Mockingjay | 2014 |     122
    Insurgent | 2015 |     127

(3 rows)
cqlsh:demovideo> CREATE TABLE videos_by_year ( id int,
             ... name text,
             ... runtime int,
             ... year int,
             ... PRIMARY KEY ((year), name ) );
cqlsh:demovideo> select * from videos_by_year;

 year | name | id | runtime
------+------+----+---------

(0 rows)
cqlsh:demovideo> INSERT INTO videos_by_name_year (name, runtime, year) VALUES ('Insu
rgent',144, 2015) ;
cqlsh:demovideo> INSERT INTO videos_by_year (name, runtime, year) VALUES ('Insurgent
',144, 2015) ;
cqlsh:demovideo> INSERT INTO videos_by_year (name, runtime, year) VALUES ('Interstel
lar',98, 2014) ;
cqlsh:demovideo> INSERT INTO videos_by_year (name, runtime, year) VALUES ('Mockingja
y',122, 2014) ;
cqlsh:demovideo> select * from videos_by_year;

 year | name         | id   | runtime
------+--------------+------+---------
 2014 | Interstellar | null |      98
 2014 |   Mockingjay | null |     122
 2015 |    Insurgent | null |     144

(3 rows)
```

CLUSTERING COLUMN WITH ORDER

```
CREATE TABLE videos_by_year3 ( id int, name text, runtime int, year int,
PRIMARY KEY (year, name)) WITH CLUSTERING ORDER BY (name DESC);

INSERT INTO videos_by_year3 (id,name, runtime, year) VALUES (1,
'Interstellar',98, 2014) ;
INSERT INTO videos_by_year3 (id,name, runtime, year) VALUES (2,
'Mockingjay',113, 2014) ;
INSERT INTO videos_by_year3 (id,name, runtime, year) VALUES (3,
'Insurgent',119, 2015) ;
```

On voit que les données sont classée dans l'ordre alphabetic car on a activé l'option : WITH CLUSTERING ORDER BY (name DESC);

```
[cqlsh:demovideo> SELECT * FROM videos_by_year3;

 year | name         | id | runtime
------+--------------+----+---------
 2014 |   Mockingjay |  2 |     113
 2014 | Interstellar |  1 |      98
 2015 |    Insurgent |  3 |     119

(3 rows)
```

QUERYING CLUSTERING COLUMNS

`SELECT * FROM videos_by_year3 WHERE year = 2014 AND name = 'Mockingjay';`
=> Returns the row(s) that has year 2014 and the name Mockingjay.

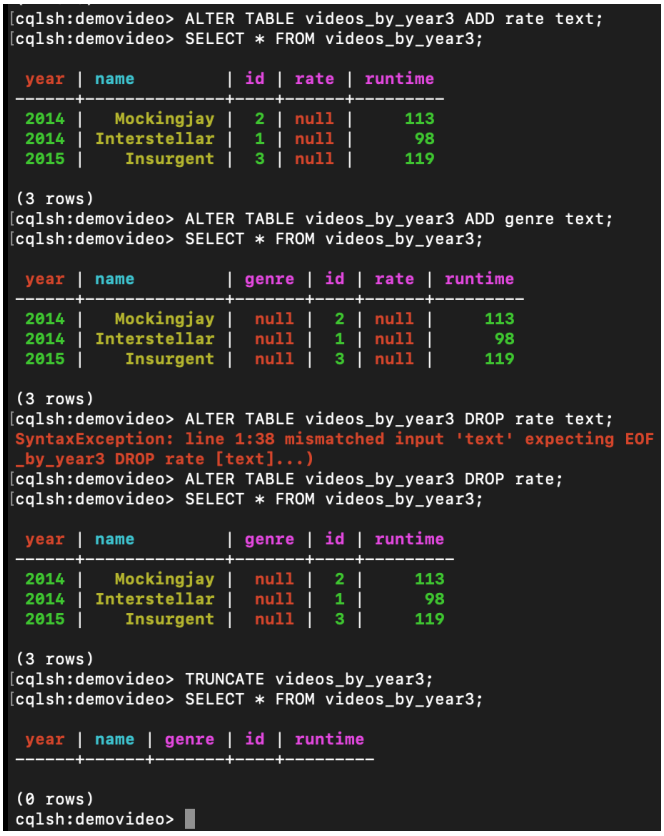
`SELECT * FROM videos_by_year3 WHERE year = 2014 AND name >= 'Interstellar';`
=> Returns the row(s) that has year 2014 and the name Mockingjay or any other name that starts with a letter greater that I.

```
[cqlsh:demovideo> SELECT * FROM videos_by_year3;                              ]

 year | name         | id | runtime
------+--------------+----+---------
 2014 |   Mockingjay |  2 |     113
 2014 | Interstellar |  1 |      98
 2015 |    Insurgent |  3 |     119

(3 rows)
[cqlsh:demovideo> SELECT * FROM videos_by_year3 where year = 2014;           ]

 year | name         | id | runtime
------+--------------+----+---------
 2014 |   Mockingjay |  2 |     113
 2014 | Interstellar |  1 |      98

(2 rows)
[cqlsh:demovideo> SELECT * FROM videos_by_year3 where year = 2014 and name= 'Mockingj]
ay';

 year | name       | id | runtime
------+------------+----+---------
 2014 | Mockingjay |  2 |     113

(1 rows)
[cqlsh:demovideo> SELECT * FROM videos_by_year3 WHERE year = 2014 AND name >= 'Inters]
tellar';

 year | name         | id | runtime
------+--------------+----+---------
 2014 |   Mockingjay |  2 |     113
 2014 | Interstellar |  1 |      98

(2 rows)
cqlsh:demovideo>
```

HARMANTEPE Melis

ALTER TABLE

`ALTER TABLE videos_by_year3 ADD genre text;` => adds a new column named “genre” of type text
`ALTER TABLE videos_by_year3 DROP rate;` => deletes the column named “rate”
`TRUNCATE videos_by_year3;` => deletes all of the data contained in the table



MULTI-VALUED COLUMN

SET <TEXT> collection of typed and ordered values (depending on value)
LIST <TEXT> ordered by position
MAP <TEXT, INT> key-value collection ordered by key

UDT (USER DEFINED TYPE)

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

CREATE TYPE full_name (first_name text, last_name text);

HARMANTEPE Melis

ALTER TABLE VIDEOS (SET)

```
cqlsh:demovideo> ALTER TABLE videos ADD tags SET<TEXT>;
cqlsh:demovideo> select * from videos;

 id | name         | runtime | tags | year
----+--------------+---------+------+------
  1 |    Insurgent |     119 | null | 2015
  2 | Interstellar |      98 | null | 2014
  3 |   Mockingjay |     122 | null | 2014

(3 rows)
cqlsh:demovideo> INSERT INTO videos (id,name, runtime, year,tags) VALUES (1, 'Insurg
ent',119, 2015, {'tag1', 'tag2'}) ;
cqlsh:demovideo> select * from videos;

 id | name         | runtime | tags             | year
----+--------------+---------+------------------+------
  1 |    Insurgent |     119 | {'tag1', 'tag2'} | 2015
  2 | Interstellar |      98 |             null | 2014
  3 |   Mockingjay |     122 |             null | 2014

(3 rows)
cqlsh:demovideo> UPDATE videos SET tags = tags + {'tag3'} WHERE id = 1 ;
cqlsh:demovideo> select * from videos;

 id | name         | runtime | tags                       | year
----+--------------+---------+----------------------------+------
  1 |    Insurgent |     119 | {'tag1', 'tag2', 'tag3'} | 2015
  2 | Interstellar |      98 |                       null | 2014
  3 |   Mockingjay |     122 |                       null | 2014

(3 rows)
cqlsh:demovideo> UPDATE videos SET tags = tags - {'tag1'} WHERE id = 1 ;
cqlsh:demovideo> select * from videos;

 id | name         | runtime | tags             | year
----+--------------+---------+------------------+------
  1 |    Insurgent |     119 | {'tag2', 'tag3'} | 2015
  2 | Interstellar |      98 |             null | 2014
  3 |   Mockingjay |     122 |             null | 2014

(3 rows)
cqlsh:demovideo> DELETE tags FROM videos WHERE id= 1 ;
cqlsh:demovideo> select * from videos;

 id | name         | runtime | tags | year
----+--------------+---------+------+------
  1 |    Insurgent |     119 | null | 2015
  2 | Interstellar |      98 | null | 2014
  3 |   Mockingjay |     122 | null | 2014

(3 rows)
```

ALTER TABLE videos ADD tags SET<TEXT>; => adds a new column of multiple types named “tags”

INSERT INTO videos (..., tags) VALUES (..., {’tag1’, ‘tag2’}); => modifies the column that has the id = 1 by adding {’tag1’, ‘tag2’} as the new values for it’s tags column

UPDATE videos SET tags = tags + {‘tag3’} WHERE id = 1 ; adds a new value ‘tag3’ to the tags column of the line that has the id = 1

UPDATE videos SET tags = tags - {‘tag1’} WHERE id = 1 ; => removes the value ‘tag1’ from the list of values of the column tags in the line that has the id =1

DELETE tags FROM videos WHERE id= 1 ; => deletes all of the values contained in the tags column of the line that has id=1

ALTER TABLE VIDEOS (LIST)

ALTER TABLE videos ADD artists LIST<TEXT>; => adds a new column called “artists” that can contain a list of values

INSERT INTO videos (..., artists) VALUES (..., [’A1’, ‘A2’]); => inserts the list of values ’A1’, ‘A2’ into the specified line

HARMANTEPE Melis

```
[cqlsh:demovideo> ALTER TABLE videos ADD artists LIST<TEXT>;                    ]
[cqlsh:demovideo> INSERT INTO videos (id,name, runtime, year, artists) VALUES (1, 'In]
 surgent',119, 2015, ['A1', 'A2']) ;
[cqlsh:demovideo> select * from videos;                                         ]

 id | artists      | name         | runtime | tags | year
----+--------------+--------------+---------+------+------
  1 | ['A1', 'A2'] |    Insurgent |     119 | null | 2015
  2 |         null | Interstellar |      98 | null | 2014
  3 |         null |   Mockingjay |     122 | null | 2014

(3 rows)
```

UPDATE videos SET artists = [‘A3’] WHERE id = 1 ; => keeps only the value ‘A3’ in the artists colum of the line that has id=1

DELETE artists[0] FROM videos WHERE id= 1 ; => deletes the first element of the array of artists from the line with id = 1

```
[cqlsh:demovideo> UPDATE videos SET artists = ['A3'] WHERE id = 1 ;
[cqlsh:demovideo> select * from videos;

 id | artists | name         | runtime | tags | year
----+---------+--------------+---------+------+------
  1 |  ['A3'] |    Insurgent |     119 | null | 2015
  2 |    null | Interstellar |      98 | null | 2014
  3 |    null |   Mockingjay |     122 | null | 2014

(3 rows)
[cqlsh:demovideo> DELETE artists[0] FROM videos WHERE id= 1 ;
[cqlsh:demovideo> select * from videos;

 id | artists | name         | runtime | tags | year
----+---------+--------------+---------+------+------
  1 |    null |    Insurgent |     119 | null | 2015
  2 |    null | Interstellar |      98 | null | 2014
  3 |    null |   Mockingjay |     122 | null | 2014

(3 rows)
```

ALTER TABLE VIDEOS (MAP)

```
[cqlsh:demovideo> ALTER TABLE videos ADD realisateurs MAP<TEXT, TEXT>;          ]
[cqlsh:demovideo> select * from videos;                                         ]

 id | artists | name         | realisateurs | runtime | tags | year
----+---------+--------------+--------------+---------+------+------
  1 |    null |    Insurgent |         null |     119 | null | 2015
  2 |    null | Interstellar |         null |      98 | null | 2014
  3 |    null |   Mockingjay |         null |     122 | null | 2014

(3 rows)
[cqlsh:demovideo> UPDATE videos SET realisateurs = {'nom':'Dupont'} WHERE id = 1 ;  ]
[cqlsh:demovideo> select * from videos;                                         ]

 id | artists | name         | realisateurs      | runtime | tags | year
----+---------+--------------+-------------------+---------+------+------
  1 |    null |    Insurgent | {'nom': 'Dupont'} |     119 | null | 2015
  2 |    null | Interstellar |              null |      98 | null | 2014
  3 |    null |   Mockingjay |              null |     122 | null | 2014

(3 rows)
[cqlsh:demovideo> UPDATE videos SET realisateurs = realisateurs+ {'prenom':'Jean'} WH]
 ERE id = 1 ;
[cqlsh:demovideo> select * from videos;                                         ]

 id | artists | name         | realisateurs                          | runtime | tags
 | year
----+---------+--------------+---------------------------------------+---------+------
+------
  1 |    null |    Insurgent | {'nom': 'Dupont', 'prenom': 'Jean'} |     119 | null
 | 2015
  2 |    null | Interstellar |                                  null |      98 | null
 | 2014
  3 |    null |   Mockingjay |                                  null |     122 | null
 | 2014

(3 rows)
[cqlsh:demovideo> UPDATE videos SET realisateurs['nom'] = 'machin' WHERE id = 1 ;  ]
[cqlsh:demovideo> select * from videos;                                         ]

 id | artists | name         | realisateurs                          | runtime | tags
 | year
----+---------+--------------+---------------------------------------+---------+------
+------
  1 |    null |    Insurgent | {'nom': 'machin', 'prenom': 'Jean'} |     119 | null
 | 2015
  2 |    null | Interstellar |                                  null |      98 | null
 | 2014
  3 |    null |   Mockingjay |                                  null |     122 | null
 | 2014

(3 rows)
```

HARMANTEPE Melis

ALTER TABLE videos ADD realisateurs MAP<TEXT, TEXT>; => adds a new column named realisateurs of type key-value pairs

UPDATE videos SET realisateurs = {‘nom’:’Dupont’} WHERE id = 1 ; => adds a key-value pair of nom and Dupont to the line that has id=1

UPDATE videos SET realisateurs = realisateurs+ {‘prenom’:’Jean’} WHERE id = 1 ; => adds a new key-value pair prenom - Jean to the line that has id=1

UPDATE videos SET realisateurs[‘nom’] = ‘machin’ WHERE id = 1 ; => updates and changes the key “nom” ’s value with “machin”

UDT

```
CREATE TYPE video_encoding (
    encoding text,
    height int,
    width int,
    bit_rates set<text>
);
```

ALTER TABLE AND ADD INFO

ALTER TABLE videos ADD encodingg video_encoding; => adds the new column named encodingg of type “video_encoding”

```
[cqlsh:demovideo> ALTER TABLE videos ADD encodingg video_encoding;                  ]
[cqlsh:demovideo> select * from videos;                                             ]

 id | encodingg | name         | realisateurs                          | runtime | yea
r
----+-----------+--------------+---------------------------------------+---------+----
--
  1 |      null |    Insurgent | {'nom': 'machin', 'prenom': 'Jean'} |     119 | 201
5
  2 |      null | Interstellar |                                  null |      98 | 201
4
  3 |      null |   Mockingjay |                                  null |     122 | 201
4

(3 rows)
```

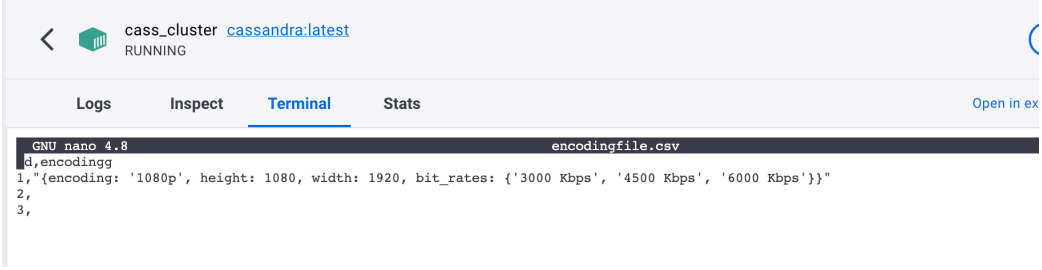
INSERT INTO videos (id, encodingg,name, realisateurs, runtime,year) VALUES (1,{encoding: '1080p', height: 1080, width: 1920, bit_rates: {'3000 Kbps', '4500 Kbps', '6000 Kbps'}}, 'Insurgent', {'nom': 'machin', 'prenom': 'Jean'},119, 2015); => add the new data type into the table

```
[cqlsh:demovideo> INSERT INTO videos (id, encodingg,name, realisateurs, runtime,year)]
  VALUES (1,{encoding: '1080p', height: 1080, width: 1920, bit_rates: {'3000 Kbps', '
4500 Kbps', '6000 Kbps'}}, 'Insurgent', {'nom': 'machin', 'prenom': 'Jean'},119, 201
5);
[cqlsh:demovideo> select * from videos;                                             ]

 id | encodingg
                    | name         | realisateurs                          | runtime
| year
----+-----------------------------------------------------------------------------
--------------------+--------------+---------------------------------------+---------
+------
  1 | {encoding: '1080p', height: 1080, width: 1920, bit_rates: {'3000 Kbps', '4500
Kbps', '6000 Kbps'}} |    Insurgent | {'nom': 'machin', 'prenom': 'Jean'} |     119
| 2015
  2 |
                 null | Interstellar |                                  null |      98
| 2014
  3 |
                 null |   Mockingjay |                                  null |     122
| 2014

(3 rows)
[cqlsh:demovideo> copy demVideo.videos(id, encodingg) to '/encodingfile.csv' with hea]
der = TRUE;
Keyspace 'demvideo' not found.
[cqlsh:demovideo> copy demovideo.videos(id, encodingg) to '/encodingfile.csv' with he]
ader = TRUE;
Using 1 child processes

Starting copy of demovideo.videos with columns [id, encodingg].
Processed: 3 rows; Rate:      15 rows/s; Avg. rate:      15 rows/s
3 rows exported to 1 files in 0.208 seconds.
cqlsh:demovideo>
```

HARMANTEPE Melis

COPY demovideo.videos(id, encodingg) TO '/encodingfile.csv' with HEADER = TRUE; => copy the videos table into the csv file

In cassandra's terminal type, first do ls to see if the file is created and then write nano encodingfile.csv and visualize the file created and the data writtent into it.



COUNTER

```
cqlsh:demovideo> CREATE TABLE videos_count_by_tag ( tag TEXT,
             ... added_year INT,
             ... video_count counter,
             ... PRIMARY KEY (tag, added_year) );
cqlsh:demovideo> select * from videos_count_by_tag;

 tag | added_year | video_count
-----+------------+-------------

(0 rows)
```

```
cqlsh:demovideo> UPDATE videos_count_by_tag SET video_count = video_count + 1 WHERE
 tag='MyTag' AND added_year=2015;
cqlsh:demovideo> select * from videos_count_by_tag;

 tag   | added_year | video_count
-------+------------+-------------
 MyTag |       2015 |           1

(1 rows)
cqlsh:demovideo> UPDATE videos_count_by_tag SET video_count = video_count + 1 WHERE
 tag='newtag' AND added_year=1999;
cqlsh:demovideo> select * from videos_count_by_tag;

 tag    | added_year | video_count
--------+------------+-------------
 newtag |       1999 |           1
 MyTag  |       2015 |           1

(2 rows)
cqlsh:demovideo> UPDATE videos_count_by_tag SET video_count = video_count + 1 WHERE
 tag='newtag' AND added_year=1999;
cqlsh:demovideo> select * from videos_count_by_tag;

 tag    | added_year | video_count
--------+------------+-------------
 newtag |       1999 |           2
 MyTag  |       2015 |           1

(2 rows)
```

Attention !! : INSERT statements are not allowed on counter tables, use UPDATE instead

At first, we didn't have any data in our table when we created it. When we want to use a counter, we can not use the insert statement to initialize the data, we must directly update the table and add a data while updating its counter.

```
UPDATE videos_count_by_tag SET video_count = video_count + 1 WHERE
tag='newtag' AND added_year=1999;
```

The program directly creates and updates its counter when we we try a counter update with a tag and a year that does not exist in the table. Afterwards, when we apply the same method on the same data, the counter of the tag is incremented.

HARMANTEPE Melis

TEMPORAL DATA

```
[cqlsh:demovideo> CREATE TABLE user (id int primary key, name text);                ]
[cqlsh:demovideo> INSERT INTO user (id, name) values (1, 'user 1');                 ]
[cqlsh:demovideo> INSERT INTO user (id, name) values (2, 'user 2') using TIMESTAMP 10]
 ;
[cqlsh:demovideo> INSERT INTO user (id, name) values (3, 'user 3');                 ]
[cqlsh:demovideo> select * from user;                                               ]

 id | name
----+--------
  1 | user 1
  2 | user 2
  3 | user 3

(3 rows)
[cqlsh:demovideo> select id, name, writetime(name) from user;                       ]

 id | name   | writetime(name)
----+--------+------------------
  1 | user 1 | 1674463712066495
  2 | user 2 |               10
  3 | user 3 | 1674463727847352

(3 rows)
[cqlsh:demovideo> UPDATE user USING TIMESTAMP 11 set name = 'user 4' where id = 2;  ]
[cqlsh:demovideo> select id, name, writetime(name) from user;                       ]

 id | name   | writetime(name)
----+--------+------------------
  1 | user 1 | 1674463712066495
  2 | user 4 |               11
  3 | user 3 | 1674463727847352

(3 rows)
[cqlsh:demovideo> UPDATE user USING TIMESTAMP 12 set name = 'user 5' where id = 2;  ]
[cqlsh:demovideo> select id, name, writetime(name) from user;                       ]

 id | name   | writetime(name)
----+--------+------------------
  1 | user 1 | 1674463712066495
  2 | user 5 |               12
  3 | user 3 | 1674463727847352

(3 rows)
```

INSERT INTO user (id, name) values (2, 'user 2') **using TIMESTAMP** 10; => assigns a user defined timestamp at the creation of the second user

select id, name, writetime(name) from user; => displays the timestamps of each user

UPDATE user **USING TIMESTAMP** 11 set name = 'user 4' where id = 2; => updates the timestamp and the name of the user with the id = 2 by assigning 11 as timestamp and user4 as name

```
[cqlsh:demovideo> DELETE name FROM user USING TIMESTAMP 13 WHERE id=2;
[cqlsh:demovideo> select id, name, writetime(name) from user;

 id | name   | writetime(name)
----+--------+------------------
  1 | user 1 | 1674463712066495
  2 |   null |             null
  3 | user 3 | 1674463727847352

(3 rows)
[cqlsh:demovideo> UPDATE user USING TTL 60 SET name = 'user 10' where id = 2;
[cqlsh:demovideo> select id, name, ttl(name) from user;

 id | name    | ttl(name)
----+---------+-----------
  1 |  user 1 |      null
  2 | user 10 |        47
  3 |  user 3 |      null

(3 rows)
```

HARMANTEPE Melis

DELETE name FROM user USING TIMESTAMP 13 WHERE id=2; => deletes the user with the id =2, at the timestamp 13

UPDATE user **USING TTL** 60 SET name = 'user 10' where id = 2; => used to specify a time-to-live (TTL) value for the updated data, meaning that the data will automatically be deleted after 60 seconds

TTL (time-to-live) is a feature that allows you to specify a period of time after which a particular piece of data will be automatically deleted.
So in the last screenshot, we see that ttl for user 2 is 47 seconds. So it means that it took 47 seconds to delete this data. After 47 seconds we don’t see this data, it is null.