COMP1835

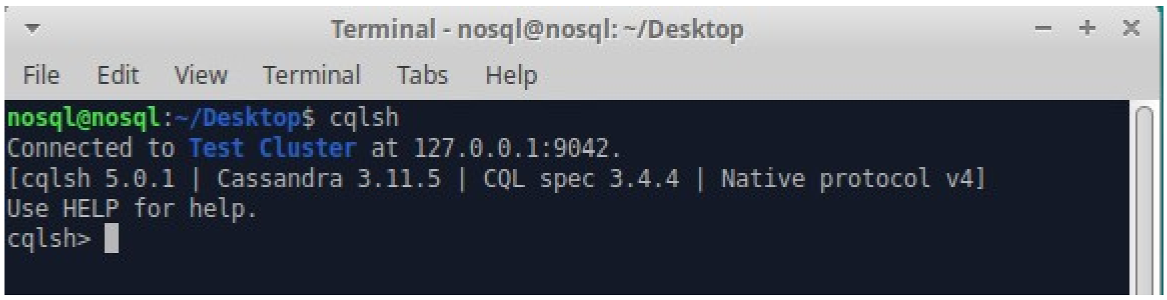
Lab 4 Working with Apache Cassandra database.

Overview

In this lab you will work with Apache Cassandra data store.

Lab 4.1 Ensure your Cassandra is running.

Open Terminal and enter **`cqlsh`** and then press enter. (**`cqlsh>`** means use **Cassandra CQL Shell** – the script to be used with the Cassandra Data store)


Terminal - nosql@nosql: ~/Desktop
File Edit View Terminal Tabs Help

```
nosql@nosql:~/Desktop$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.5 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

You should also see a command prompt appear which says:

```
Connected to: Test Cluster on 127.0.0.1/9042
```

`Test Cluster` is the name of the default database which has been set up for you.

Lab 4.2 Creating a Keyspace and a Table (Column Family)

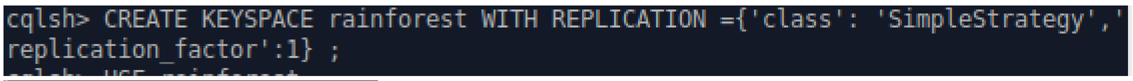
At the **cqlsh**> command prompt, create a keyspace called **`rainforest.`** Now set the replication class to **`SimpleStrategy`**, as we do not have multiple data centres, and the **`replication_factor`** to 1.

The command should look like this:

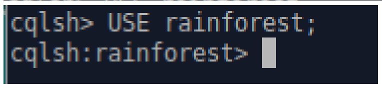
```
CREATE KEYSPACE rainforest

WITH REPLICATION = {'class':
'SimpleStrategy','replication_factor': 1 };
```

Then you should switch to use this new KeySpace that you have just created with the following code;

```
USE rainforest;
```



```
cqlsh> CREATE KEYSPACE rainforest WITH REPLICATION ={'class': 'SimpleStrategy','
replication_factor':1} ;
```



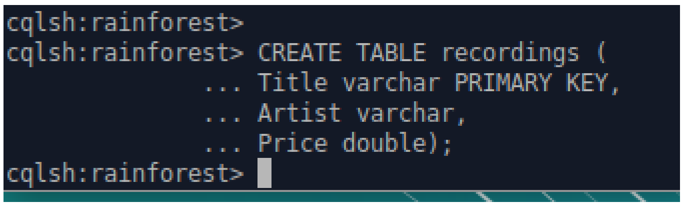
```
cqlsh> USE rainforest;
cqlsh:rainforest>
```

Now create a table (column family) called **`recordings`** with the following columns**:**

1. **`Title`** of type **`String`** and the primary key
2. **`Artist`** of type **`String`**
3. **`Price`** of type **`double`**

The command should look like this:

```
CREATE TABLE recordings  (
Title varchar PRIMARY KEY,
Artist varchar,
Price double);
```



```
cqlsh:rainforest>
cqlsh:rainforest> CREATE TABLE recordings (
              ... Title varchar PRIMARY KEY,
              ... Artist varchar,
              ... Price double);
cqlsh:rainforest>
```

Lab 4.3 **Insert data into the table (column family).**

Insert the following music recordings into the table:

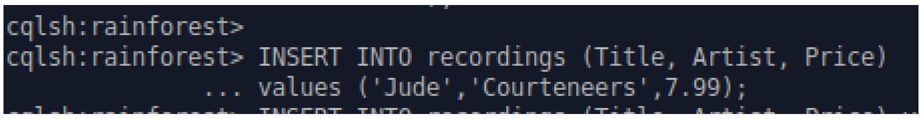
```
Title:  Jude
Artist: Courteneers
Price: 7.99
```

Your insert command should be in the form of:

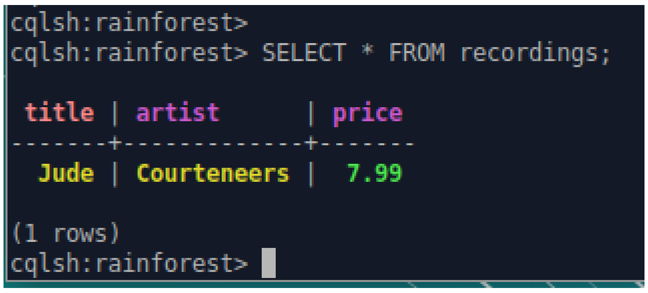
```
INSERT INTO recordings (Title, Artist, Price)
values ('Jude', 'Courteneers', 7.99);
```

If you want to check to see if the music recording has been successfully inserted, then use the following command:

```
SELECT * FROM recordings;
```



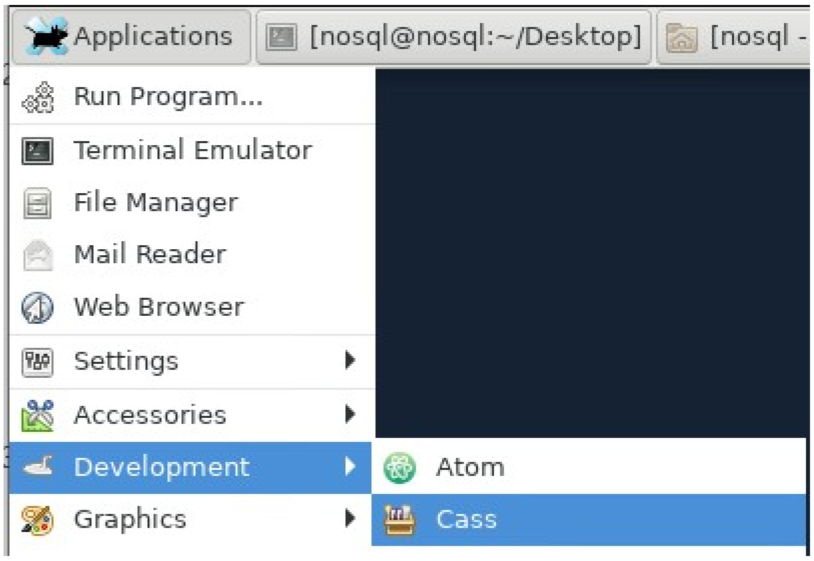
```
cqlsh:rainforest>
cqlsh:rainforest> INSERT INTO recordings (Title, Artist, Price)
              ... values ('Jude','Courteneers',7.99);
```



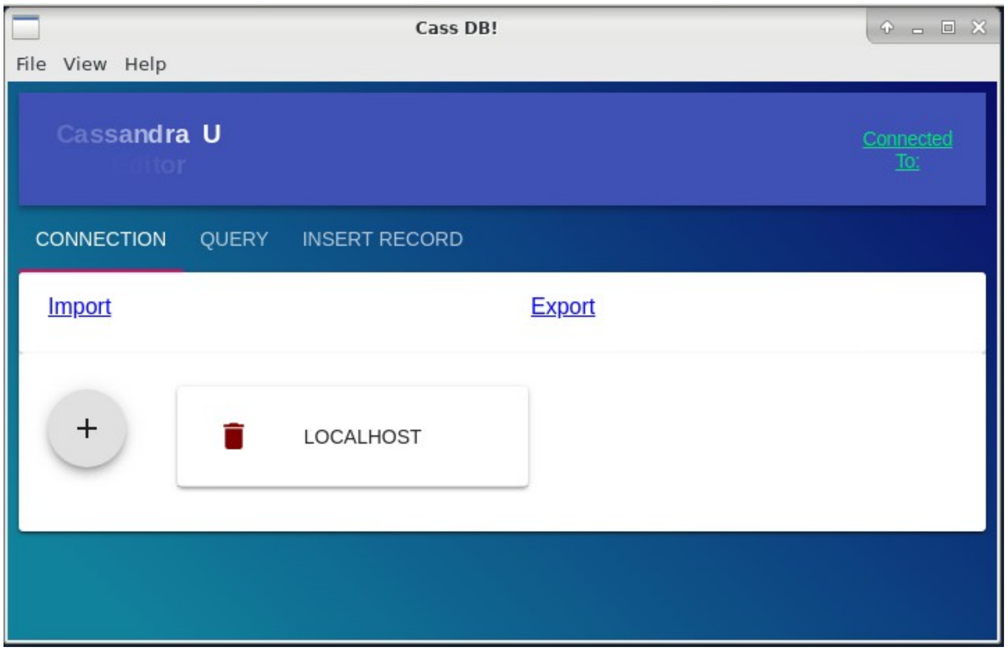
```
cqlsh:rainforest>
cqlsh:rainforest> SELECT * FROM recordings;

 title | artist      | price
-------+-------------+-------
  Jude | Courteneers |  7.99

(1 rows)
cqlsh:rainforest>
```

Lab 4.4 Using Cass DB – Cassandra UI Editor

Open Cass DB application by navigating Applications->Development->Cass



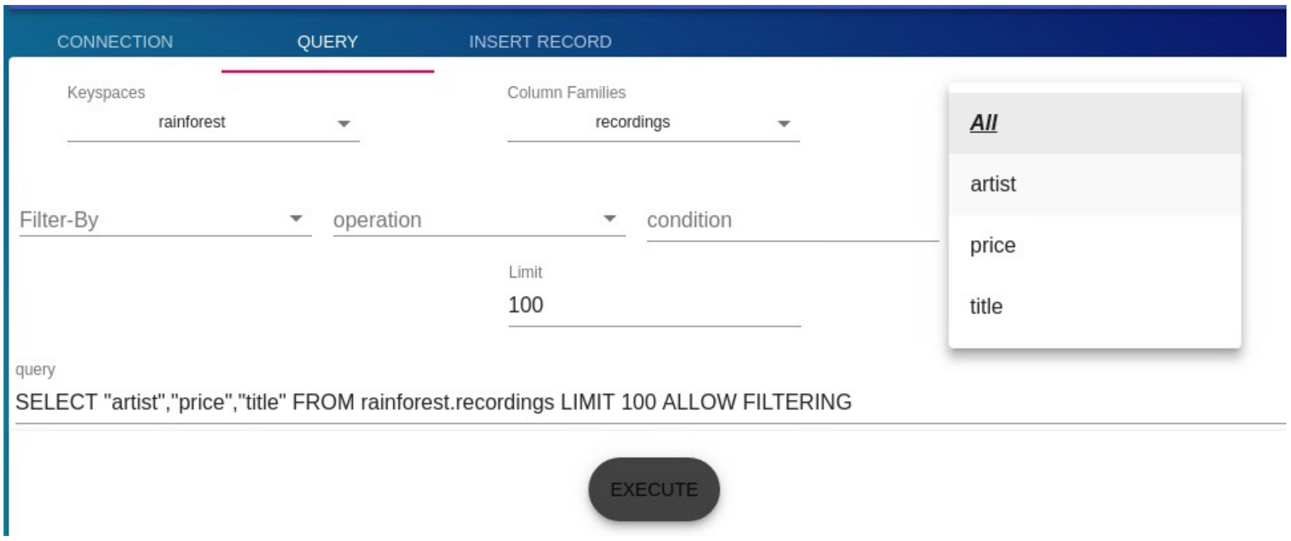
Connect to your Cassandra database by clicking on **LOCALHOST** button **(Be careful not to click on a recycle bin icon!)**



You should get a note in the right top corner

First, we can query the existing table recordings:

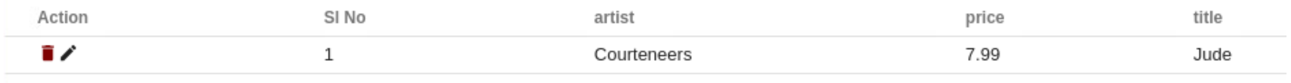
Click on a menu **QUERY**, select Keyspace=rainforest, Column family=recordings and Columns=All



Limit
100

query
SELECT "artist","price","title" FROM rainforest.recordings LIMIT 100 ALLOW FILTERING

Notice the query is written below. Press Execute button to run the query. The results will be displayed below.



Action	Sl No	artist	price	title
🗑 ✎	1	Courteneers	7.99	Jude

NOW enter 3 more records into the table using command line interface and the same type of command show above.

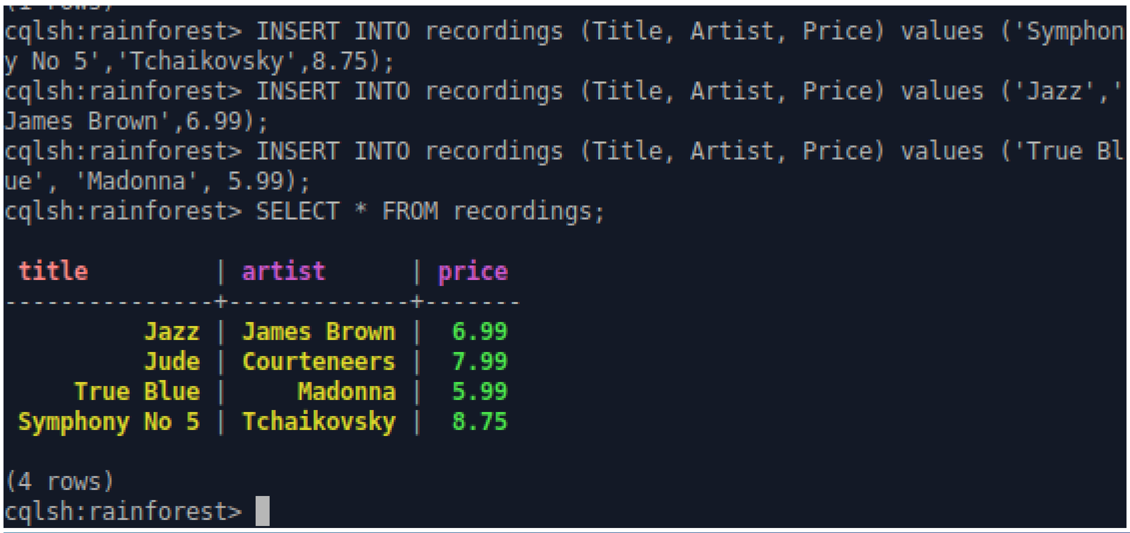
(It seems that INSERT RECOD menu is not working properly)

```
Title:  Symphony No 5
Artist: Tchaikovsky
Price: 8.75

Title:  Jazz
Artist: James Brown
Price: 6.99

Title:  True Blue
Artist: Madonna
Price: 5.99
```

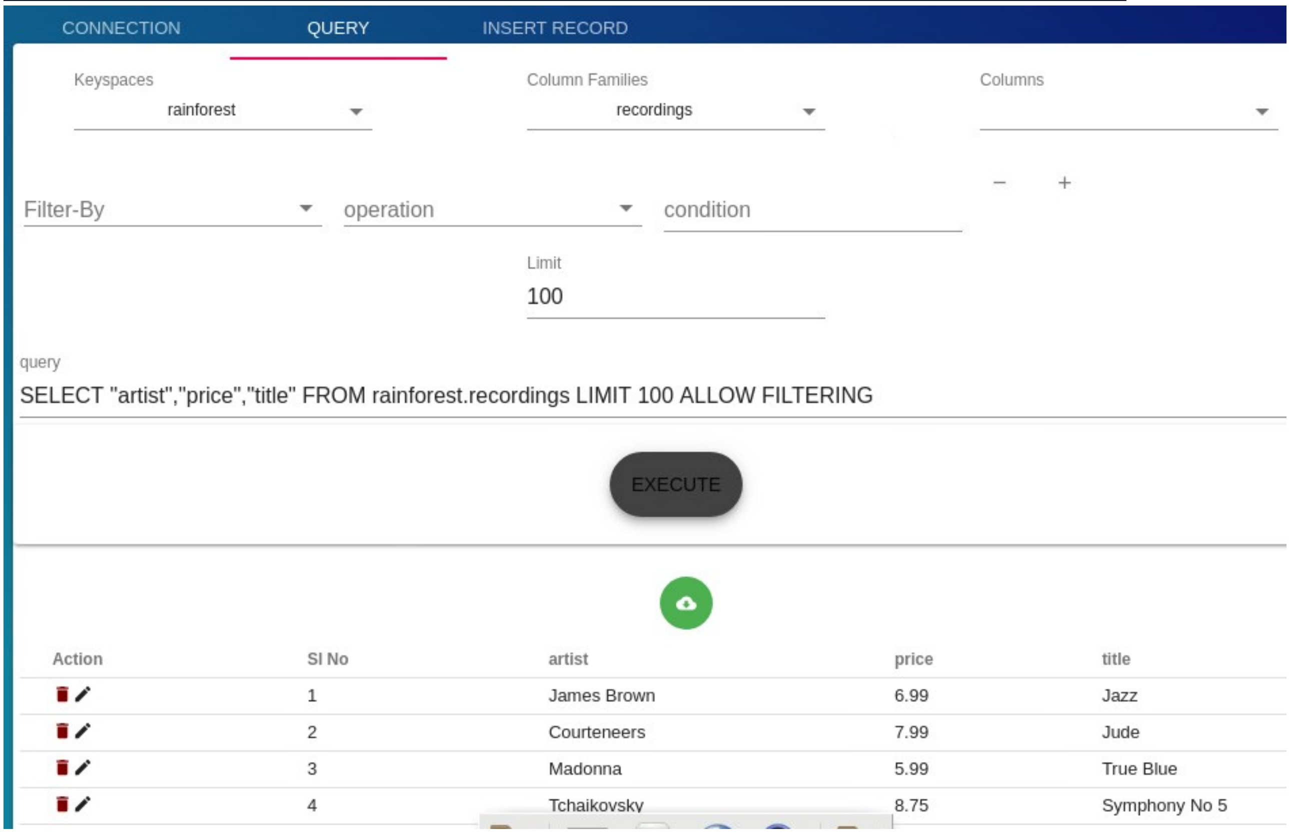
Afterwards you should select all of the data which has been entered into the **`recordings`** column family which is part of your **`rainforest`** keyspace.

You can use either **`cqlsh`** or GUI Cass DB

Your output should look something like this:



```
cqlsh:rainforest> INSERT INTO recordings (Title, Artist, Price) values ('Symphon
y No 5','Tchaikovsky',8.75);
cqlsh:rainforest> INSERT INTO recordings (Title, Artist, Price) values ('Jazz','
James Brown',6.99);
cqlsh:rainforest> INSERT INTO recordings (Title, Artist, Price) values ('True Bl
ue', 'Madonna', 5.99);
cqlsh:rainforest> SELECT * FROM recordings;

 title          | artist      | price
----------------+-------------+-------
           Jazz | James Brown |  6.99
           Jude | Courteneers |  7.99
      True Blue |     Madonna |  5.99
 Symphony No 5 | Tchaikovsky |  8.75

(4 rows)
cqlsh:rainforest> 
```



Keyspaces: rainforest

Column Families: recordings

Columns:

Filter-By | operation | condition

Limit: 100

query: SELECT "artist","price","title" FROM rainforest.recordings LIMIT 100 ALLOW FILTERING

Action	Sl No	artist	price	title
🗑 ✎	1	James Brown	6.99	Jazz
🗑 ✎	2	Courteneers	7.99	Jude
🗑 ✎	3	Madonna	5.99	True Blue
🗑 ✎	4	Tchaikovsky	8.75	Symphony No 5

Lab 4.5 Using Queries with Conditions.

We will now try to query our data in the **recordings** column family with conditions.

Let’s write a query that will select all of the recordings where the price is **5.99**

We will do this by using the where clause in our command in the form of:

```
where column = ‘value’
```

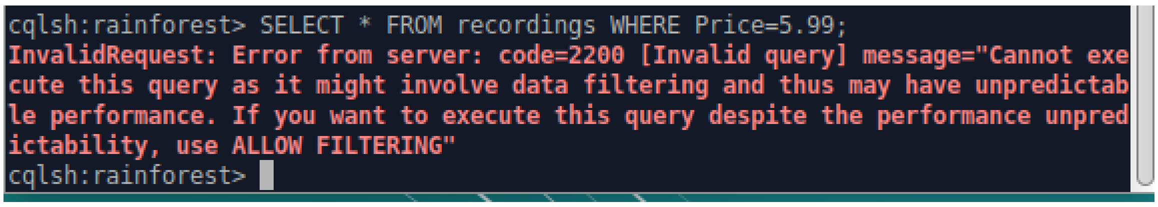
Note: The full syntax of the SELECT command is:

```
SELECT * | select_expression | DISTINCT partition
FROM [keyspace_name.] table_name
[WHERE partition_value
   [AND clustering_filters
   [AND static_filters]]]
[ORDER BY PK_column_name ASC|DESC]
[LIMIT N]
[ALLOW FILTERING]
```

Now try to use the following command and then press enter:

```
SELECT * from recordings WHERE Price = 5.99;
```

What happens? Did you get an error? You should see the following output after running your command.



```
cqlsh:rainforest> SELECT * FROM recordings WHERE Price=5.99;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot exe
cute this query as it might involve data filtering and thus may have unpredictab
le performance. If you want to execute this query despite the performance unpred
ictability, use ALLOW FILTERING"
cqlsh:rainforest>
```

To solve the problem, you need to use clause ALLOW FILTERING.

Note on ALLOW FILTERING:

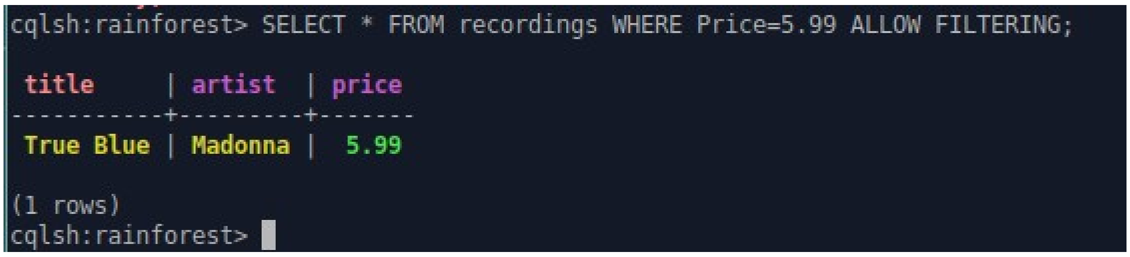
Cassandra knows that it might not be able to execute the query in an efficient way. It is therefore warning you: “Be careful. Executing this query as such might not be a good idea as it can use a lot of your computing resources”.

The only way Cassandra can execute this query is by retrieving all the rows from the table blogs and then by filtering out the ones which do not have the requested value for the time1 column.

If your table contains for example a 1 million rows and 95% of them have the requested value for the price column, the query will still be relatively efficient, and you should use ALLOW FILTERING.

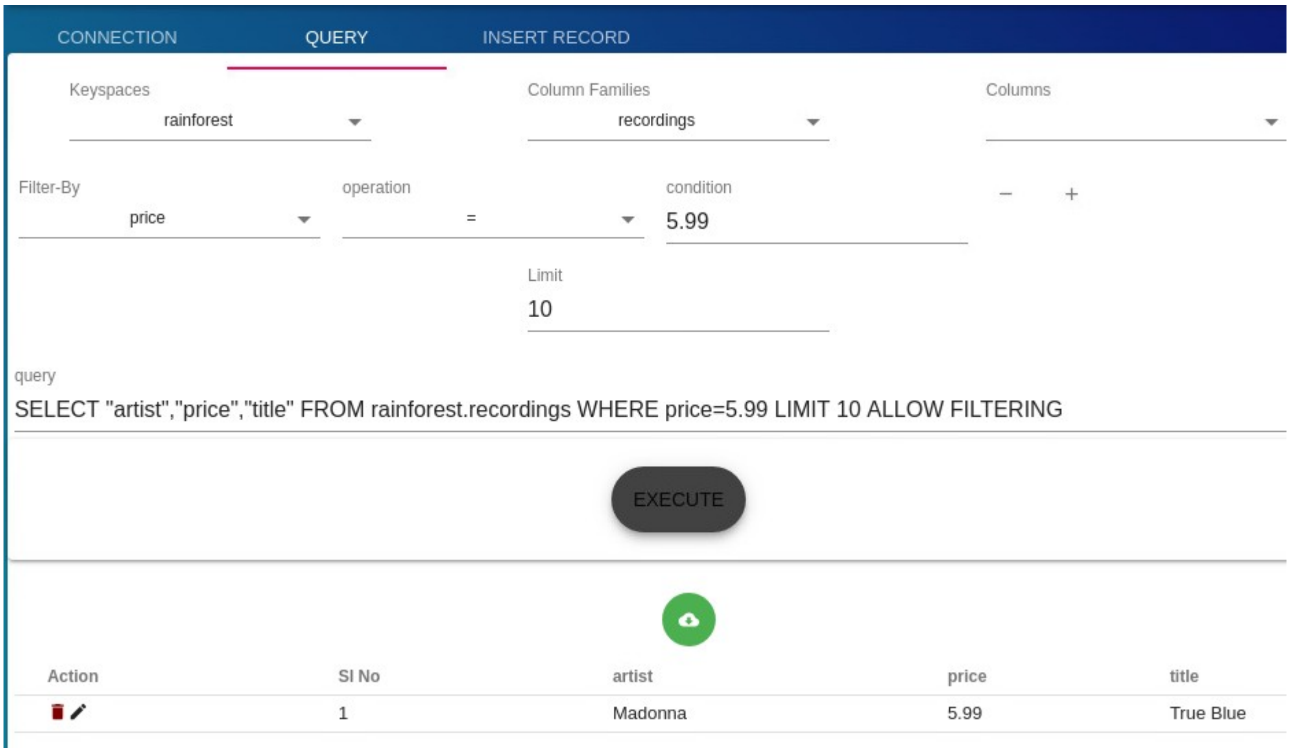
On the other hand, if your table contains 1 million rows and only 2 rows contain the requested value for the price column, your query is extremely inefficient. Cassandra will load 999, 998 rows for nothing. If the query is often used, it is probably better to add an index on the price column.

Unfortunately, Cassandra has no way to differentiate between the 2 cases above as they are depending on the data distribution of the table. Cassandra is therefore warning you and relying on you to make the good choice.



```
cqlsh:rainforest> SELECT * FROM recordings WHERE Price=5.99 ALLOW FILTERING;

 title      | artist  | price
------------+---------+-------
 True Blue  | Madonna |  5.99

(1 rows)
cqlsh:rainforest>
```

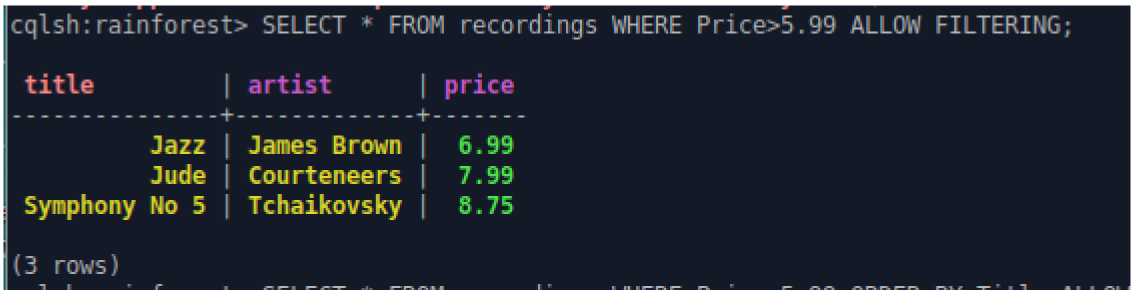
You can try to get the same in Cass DB. If selecting Filter-By does not work, you always can update the query written on the screen manually and then press EXECUTE.



Keyspaces: rainforest
Column Families: recordings
Columns:
Filter-By: price
operation: =
condition: 5.99
Limit: 10
query: SELECT "artist","price","title" FROM rainforest.recordings WHERE price=5.99 LIMIT 10 ALLOW FILTERING

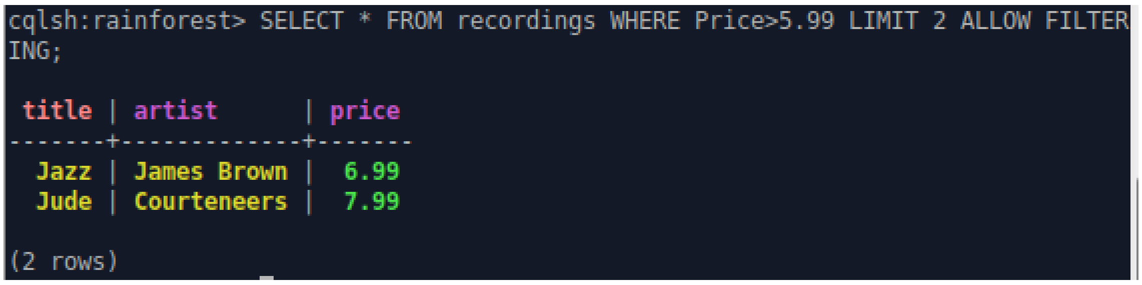
Action	Sl No	artist	price	title
	1	Madonna	5.99	True Blue

Now get all recordings where price is >5.99



```
cqlsh:rainforest> SELECT * FROM recordings WHERE Price>5.99 ALLOW FILTERING;

 title          | artist      | price
----------------+-------------+-------
           Jazz | James Brown |   6.99
           Jude | Courteneers |   7.99
 Symphony No 5  | Tchaikovsky |   8.75

(3 rows)
```

In order to show first few records - restrict the output to **`n`** rows use **`LIMIT n`**. For example:



```
cqlsh:rainforest> SELECT * FROM recordings WHERE Price>5.99 LIMIT 2 ALLOW FILTER
ING;

 title | artist      | price
-------+-------------+-------
  Jazz | James Brown |   6.99
  Jude | Courteneers |   7.99

(2 rows)
```

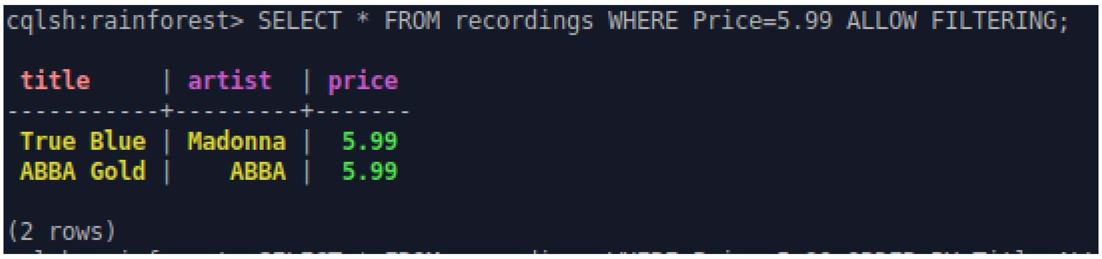
Lab 4.6 Insert one more record with

```
Title:  ABBA Gold
Artist: ABBA
Price: 5.99
```

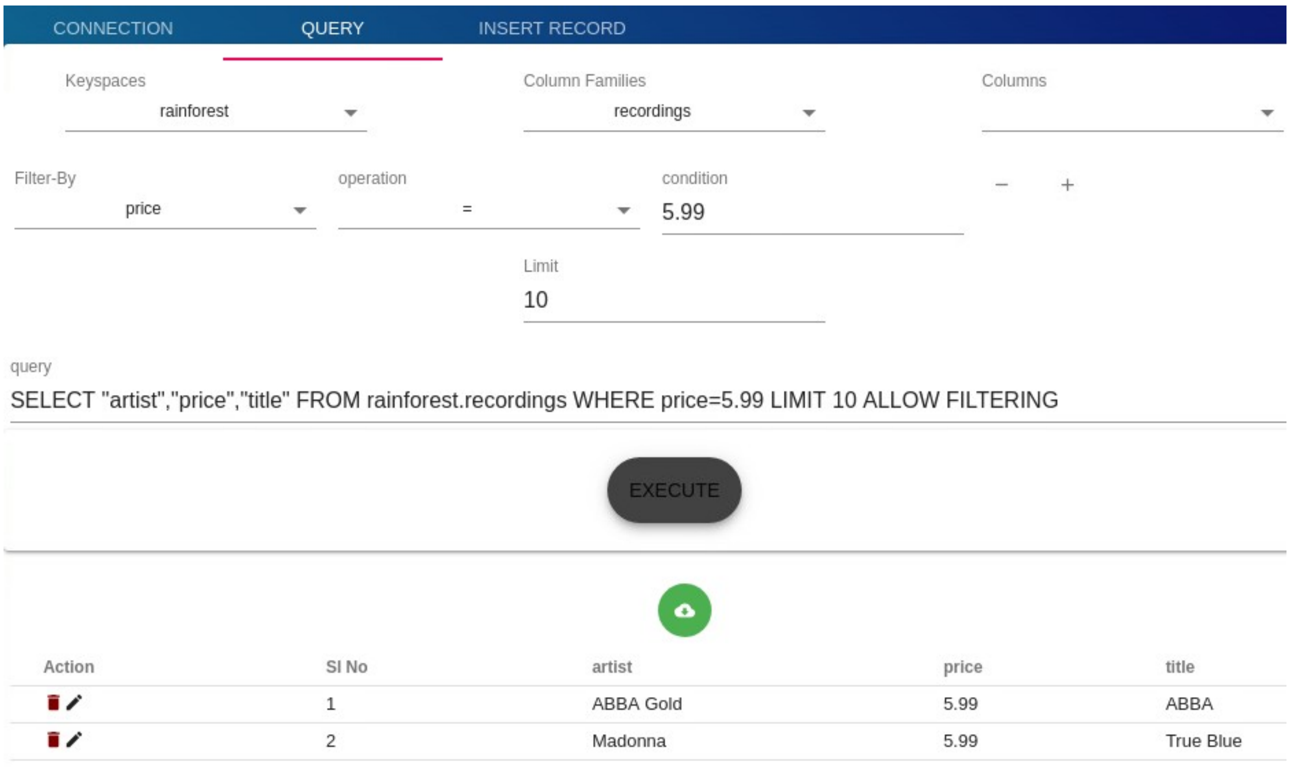
And run the same query again:

`SELECT * from recordings WHERE Price = 5.99 ALLOW FILTERING;`

Your output should look like this:



```
cqlsh:rainforest> SELECT * FROM recordings WHERE Price=5.99 ALLOW FILTERING;

 title     | artist  | price
-----------+---------+-------
 True Blue | Madonna |   5.99
 ABBA Gold |    ABBA |   5.99

(2 rows)
```



Keyspaces: rainforest
Column Families: recordings
Columns:
Filter-By: price
operation: =
condition: 5.99
Limit: 10
query: SELECT "artist","price","title" FROM rainforest.recordings WHERE price=5.99 LIMIT 10 ALLOW FILTERING

Action	Sl No	artist	price	title
	1	ABBA Gold	5.99	ABBA
	2	Madonna	5.99	True Blue

Lab 4.7 Using aggregate functions in Cassandra.

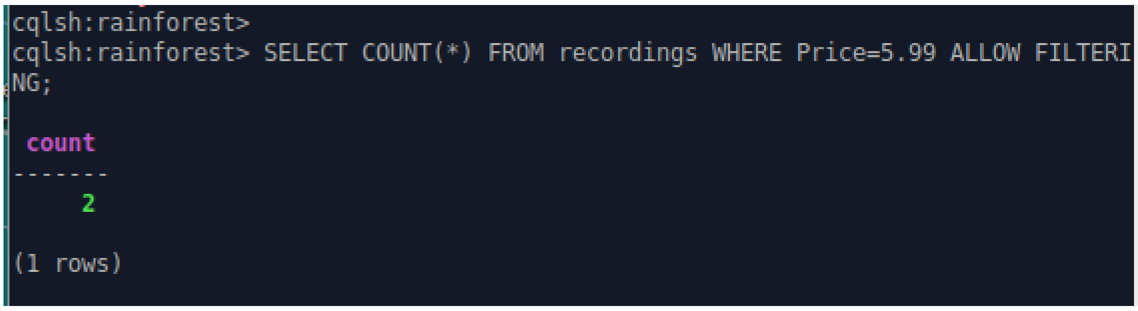
Try the following conditional queries with aggregate functions:

```
SELECT count (*) FROM recordings ALLOW FILTERING;
```

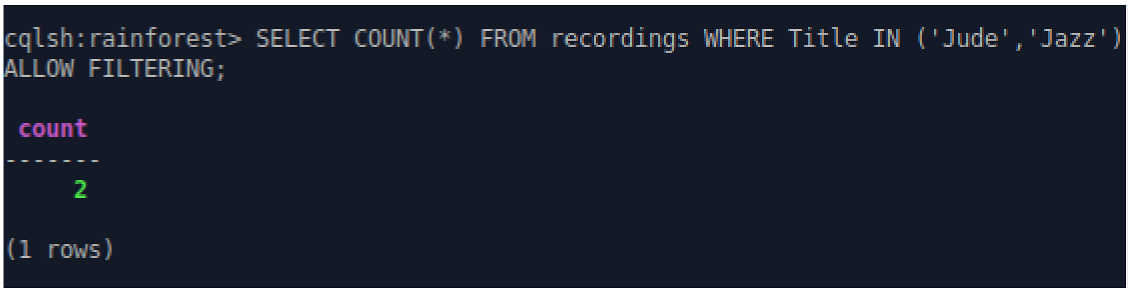
How many recordings were there?

```
SELECT count (*) FROM recordings WHERE Price = 5.99 ALLOW
FILTERING;
```

How many recordings were there priced at 5.99?

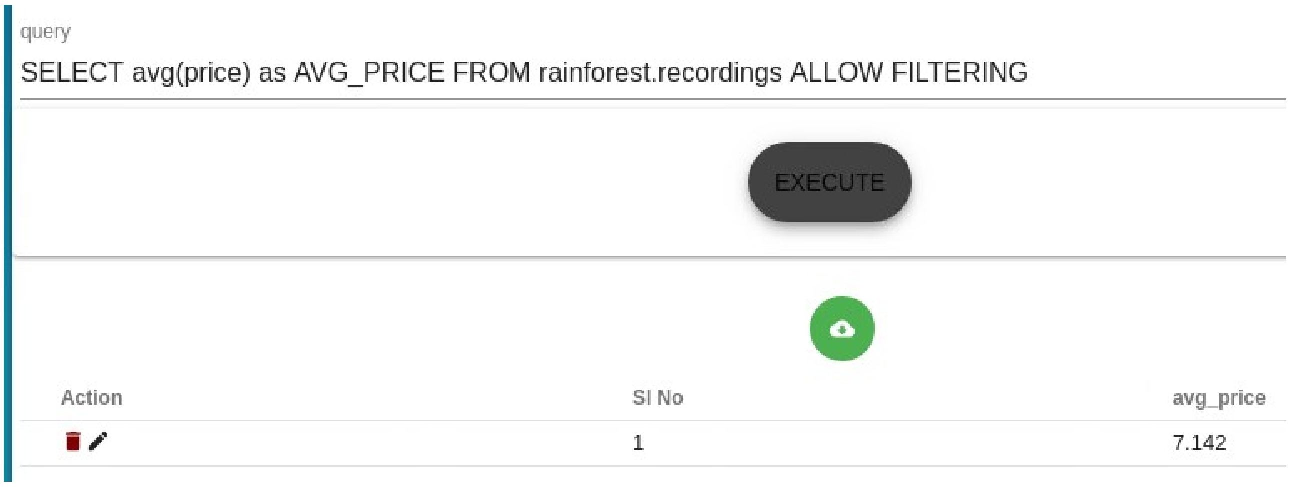

```
cqlsh:rainforest>
cqlsh:rainforest> SELECT COUNT(*) FROM recordings WHERE Price=5.99 ALLOW FILTERI
NG;

 count
-------
     2

(1 rows)
```

How many recordings had the title of either Jude or Jazz?



```
cqlsh:rainforest> SELECT COUNT(*) FROM recordings WHERE Title IN ('Jude','Jazz')
ALLOW FILTERING;

 count
-------
     2

(1 rows)
```

What is the average price of the recordings?

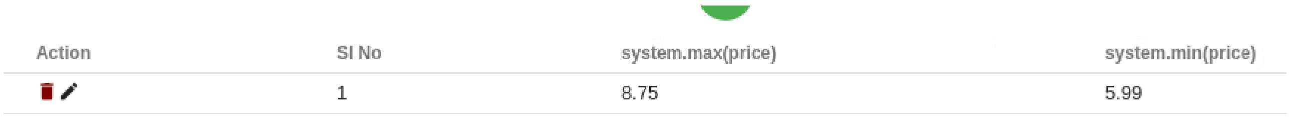


query

```
SELECT avg(price) as AVG_PRICE FROM rainforest.recordings ALLOW FILTERING
```

Action	Sl No	avg_price
🗑 ✎	1	7.142

Identify maximum and minimum price, using **`max`** and **`min`** functions.

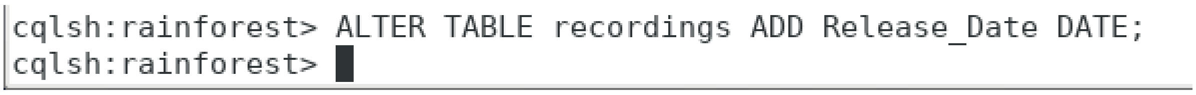


Action	Sl No	system.max(price)	system.min(price)
🗑 ✎	1	8.75	5.99

Lab 4.8 Altering the table

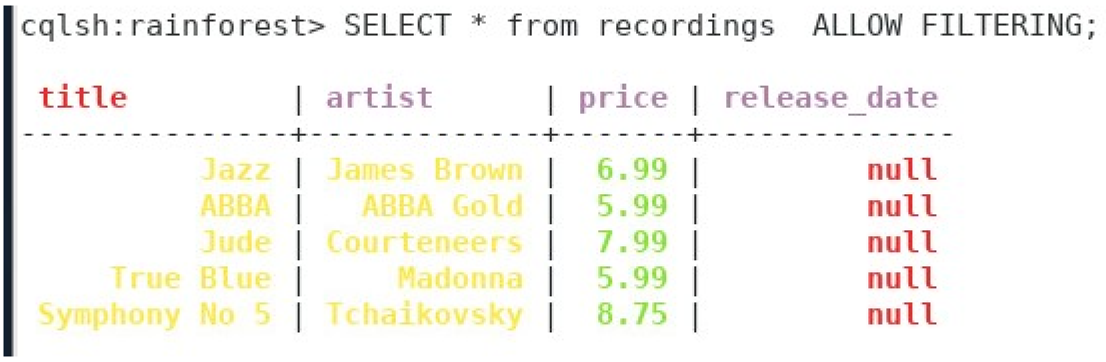
Now you would need to add additional column to the table recordings to store the information on when the album was released.

Use ALTER TABLE command to add a new column Release_Date



```
cqlsh:rainforest> ALTER TABLE recordings ADD Release_Date DATE;
cqlsh:rainforest> █
```

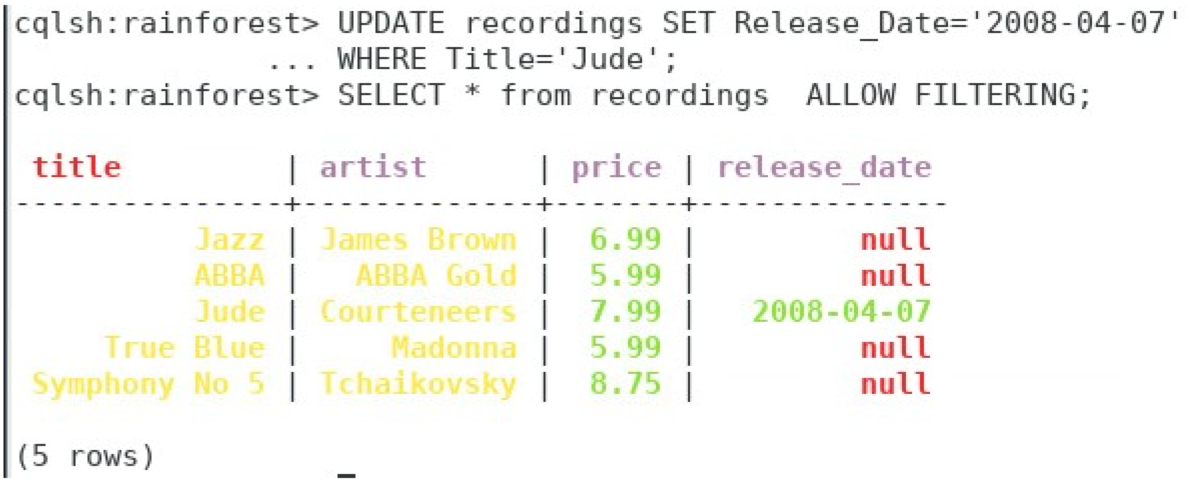
Select all records from all columns:



```
cqlsh:rainforest> SELECT * from recordings  ALLOW FILTERING;

 title           | artist       | price | release_date
-----------------+--------------+-------+--------------
            Jazz | James Brown  |  6.99 |          null
            ABBA |    ABBA Gold |  5.99 |          null
            Jude |  Courteneers |  7.99 |          null
       True Blue |      Madonna |  5.99 |          null
  Symphony No 5  |  Tchaikovsky |  8.75 |          null
```

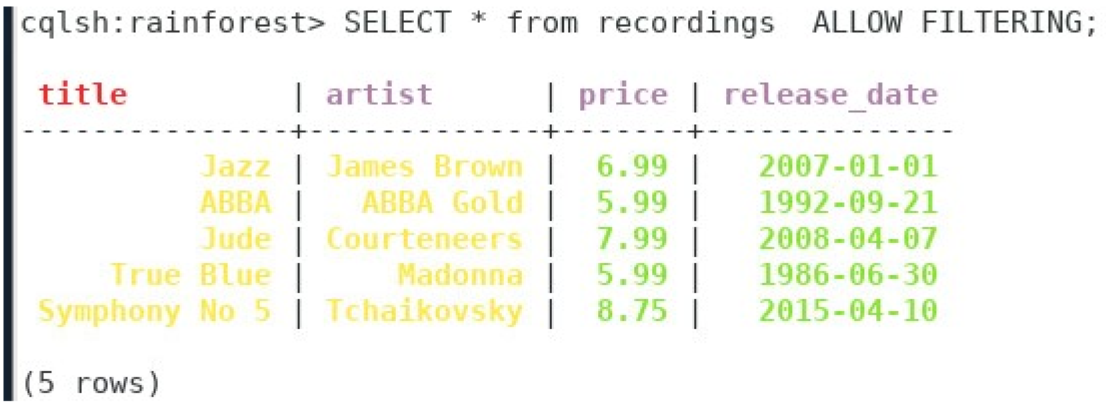
Now you need to update data in the column release_date using UPDATE command.

Remember, Cassandra accepts DATE type values in the following format: yyyy-mm-dd (so '2021-02-09' for instance).



```
cqlsh:rainforest> UPDATE recordings SET Release_Date='2008-04-07'
              ... WHERE Title='Jude';
cqlsh:rainforest> SELECT * from recordings  ALLOW FILTERING;

 title           | artist       | price | release_date
-----------------+--------------+-------+--------------
            Jazz | James Brown  |  6.99 |          null
            ABBA |    ABBA Gold |  5.99 |          null
            Jude |  Courteneers |  7.99 |    2008-04-07
       True Blue |      Madonna |  5.99 |          null
  Symphony No 5  |  Tchaikovsky |  8.75 |          null

(5 rows)
```

Update the rest of the records with the recording's release date

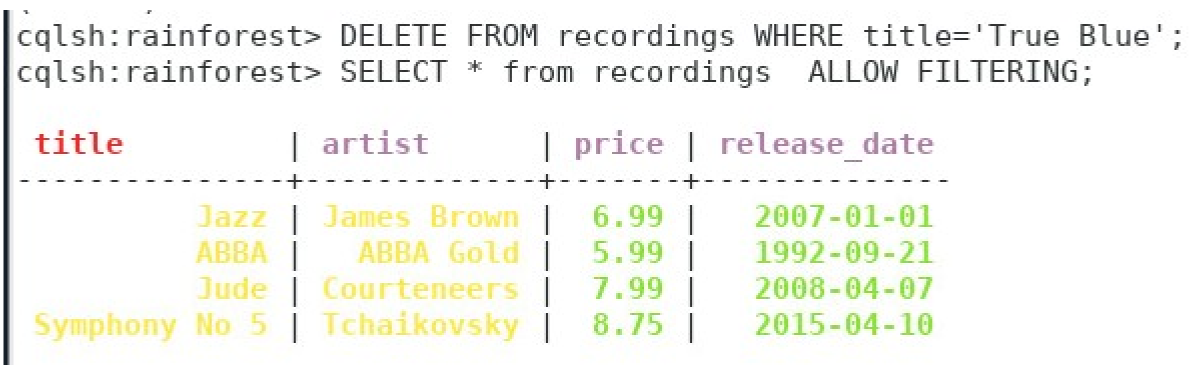


```
cqlsh:rainforest> SELECT * from recordings  ALLOW FILTERING;

 title           | artist       | price | release_date
-----------------+--------------+-------+--------------
            Jazz | James Brown  |  6.99 |    2007-01-01
            ABBA |    ABBA Gold |  5.99 |    1992-09-21
            Jude |  Courteneers |  7.99 |    2008-04-07
       True Blue |      Madonna |  5.99 |    1986-06-30
  Symphony No 5  |  Tchaikovsky |  8.75 |    2015-04-10

(5 rows)
```

Lab 4.9 Delete data.

Remove record with Title 'True Blue'.

How many records are now left in recordings?

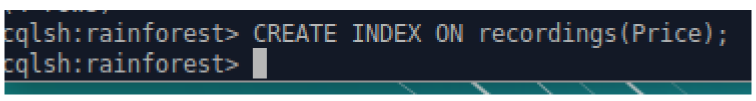


```
cqlsh:rainforest> DELETE FROM recordings WHERE title='True Blue';
cqlsh:rainforest> SELECT * from recordings  ALLOW FILTERING;

 title          | artist      | price | release_date
----------------+-------------+-------+--------------
           Jazz | James Brown |  6.99 |   2007-01-01
           ABBA |   ABBA Gold |  5.99 |   1992-09-21
           Jude | Courteneers |  7.99 |   2008-04-07
 Symphony No 5  | Tchaikovsky |  8.75 |   2015-04-10
```

Lab 4.10 Creating an index.

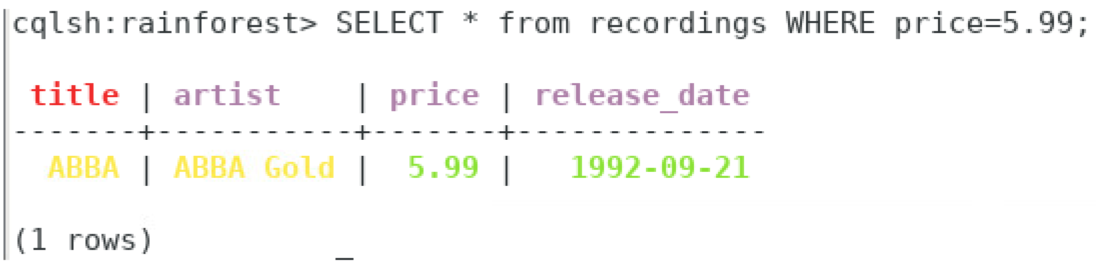
4.10.1 To avoid ALLOW FILTERING we need to create an index on a column we use frequently in the where condition.

`CREATE INDEX ON recordings (Price);`



```
cqlsh:rainforest> CREATE INDEX ON recordings(Price);
cqlsh:rainforest>
```

Now you can use SELECT with WHERE condition and omit ALLOW FILTERING



```
cqlsh:rainforest> SELECT * from recordings WHERE price=5.99;

 title | artist    | price | release_date
-------+-----------+-------+--------------
  ABBA | ABBA Gold |  5.99 |   1992-09-21

(1 rows)
```

4.10.2 Secondary indices and ALLOW FILTERING
If we add an index on the price column and execute the following query:

```
SELECT * FROM recordings WHERE Price = 5.99;
```

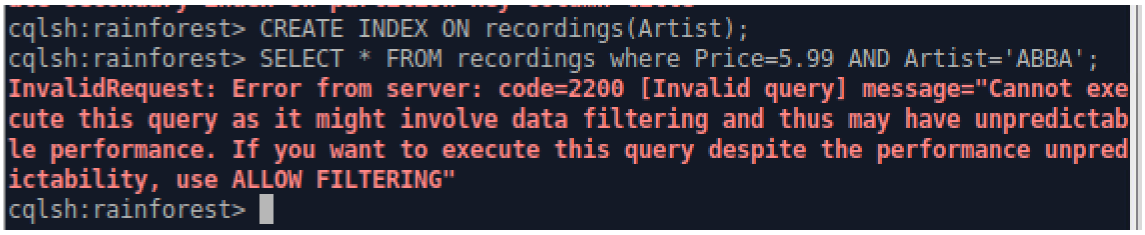
Cassandra will return all the recordings with that price and will not request ALLOW FILTERING. This is because Cassandra can use the secondary index on the price column to find the matching rows and does not need to perform any filtering.

But if we execute the following one:

```
SELECT * FROM recordings WHERE Price=5.99 and Title = ‘ABBA Gold’;
```

Cassandra will request ALLOW FILTERING as it will have to first find and load the rows containing Jonathan as author, and then to filter out the ones which do not have a time2 column equal to the specified value.

Adding an index on time2 might improve the query performance. Cassandra will then use the index with the highest selectivity to find the rows that need to be loaded. It will however not change anything regarding the need for ALLOW FILTERING, as it will still have to filter the loaded rows using the remaining predicate.



```
cqlsh:rainforest> CREATE INDEX ON recordings(Artist);
cqlsh:rainforest> SELECT * FROM recordings where Price=5.99 AND Artist='ABBA';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot exe
cute this query as it might involve data filtering and thus may have unpredictab
le performance. If you want to execute this query despite the performance unpred
ictability, use ALLOW FILTERING"
cqlsh:rainforest>
```

Making the right choice
When your query is rejected by Cassandra because it needs filtering, you should resist the urge to just add ALLOW FILTERING to it. You should think about your data, your model and what you are trying to do. You always have multiple options.

You can change your data model, add an index, use another table or use ALLOW FILTERING.

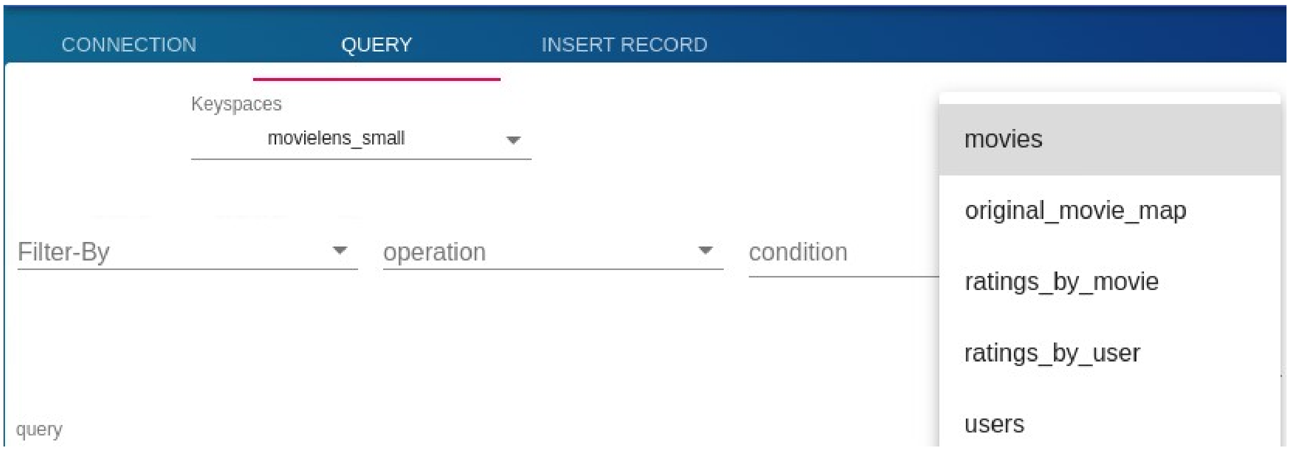
You have to make the right choice for your specific use case.

Lab 4.11 Working with large data set.

Your VM has pre-installed dataset in Cassandra, called **`movielens`** (https://github.com/srviest/movie-recommender/tree/master/dataset/movielens/small)

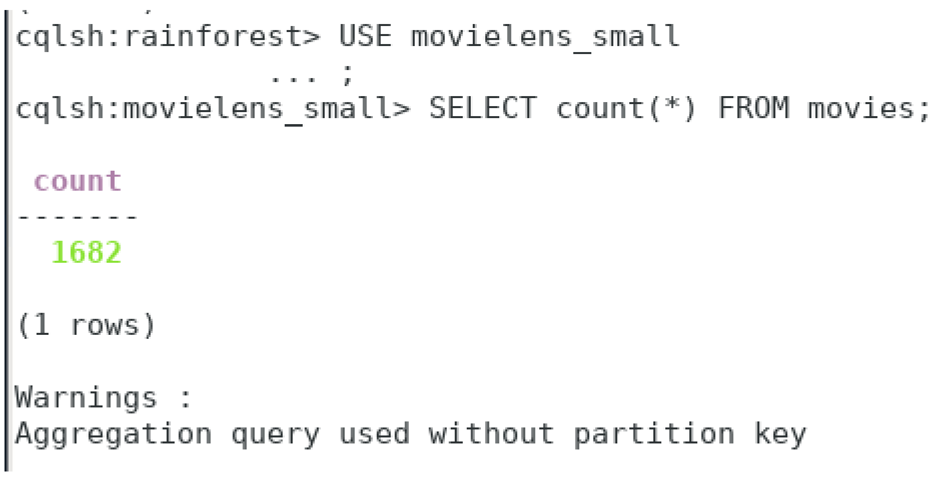
To perform queries on that set you need to navigate to Keyspace= movielens_small.

You will see 5 tables there: `movies, original_movie_map, ratings_by_movie, ratings_by_user` and `users:`



To use **`cqlsh`** command line you need first to switch to movielens_small.

Then you can query how many records you have in each table, for example:



```
cqlsh:rainforest> USE movielens_small
              ... ;
cqlsh:movielens_small> SELECT count(*) FROM movies;

 count
-------
  1682

(1 rows)

Warnings :
Aggregation query used without partition key
```

Do not worry about warning, this is Cassandra's way of informing you that the query was not very efficient, and it only works if your table is small and there are no partitions, which is our case. (If the table is partitioned, then we would need to add WHERE condition based on the partition key.)

Investigate this data set, performing queries of your choice.