Exercise 1:

After installation of Cassandra by running

*helm install bitnami/cassandra --set dbUser.password=cassandra --generate-name*

A black screen with green text

Description automatically generated

We need to get password by running:

*$base64Password = kubectl get secret --namespace "default" cassandra-1697799782 -o jsonpath="{.data.cassandra-password}"*

*$CASSANDRA\_PASSWORD = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($base64Password))*

*Write-Host $CASSANDRA\_PASSWORD*

Connect to Cassandra:

*cqlsh -u cassandra -p cassandra cassandra-1697799782*

A screenshot of a computer

Description automatically generated

Create keyspace

*CREATE KEYSPACE killrvideo WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': 1};*

Use keyspace

*USE killrvideo;*

Create a single table:

*CREATE TABLE videos (*

*video\_id timeuuid PRIMARY KEY,*

*added\_date timestamp,*

*description text,*

*title text,*

*user\_id uuid*

*);*

Load the newly created table with the videos.csv file

*COPY videos FROM '/tmp/videos.csv' WITH HEADER = true;*

Use SELECT to verify the data loaded correctly. Include LIMIT to retrieve only the first 10 rows.

*SELECT \* FROM videos LIMIT 10;*

A close-up of a text

Description automatically generated

Use SELECT to COUNT(\*) the number of imported rows. It should match the number of rows COPY reported as imported.

*SELECT COUNT(\*) FROM videos;*

A screenshot of a computer

Description automatically generated

Use select to find a row where the video\_id=6c4cffb9-0dc4-1d59-af24-c960b5fc3652

*SELECT \* FROM videos WHERE video\_id = 6c4cffb9-0dc4-1d59-af24-c960b5fc3652;*

A close-up of a text

Description automatically generated

Exercise 2:

Create table

*CREATE TABLE videos\_by\_title\_year (*

*title text,*

*added\_year int,*

*added\_date timestamp,*

*description text,*

*user\_id uuid,*

*video\_id uuid,*

*PRIMARY KEY ((title, added\_year))*

*);*

Load the data from the videos\_by\_title\_year.csv

*COPY videos\_by\_title\_year FROM '/tmp/ videos\_by\_title\_year.csv  ' WITH HEADER = true;*

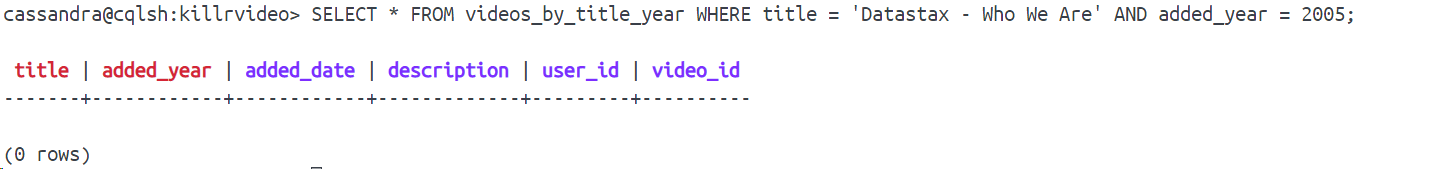
Try running queries on the videos\_by\_title\_year table to query on a specific title and added\_year.

*SELECT \* FROM videos\_by\_title\_year WHERE title = 'Datastax - Who We Are' AND added\_year = 2014;*

A screenshot of a computer

Description automatically generated

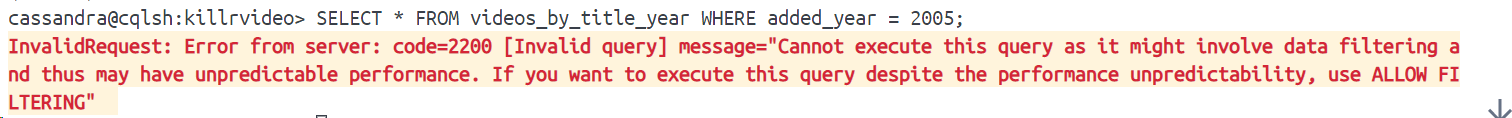
*SELECT \* FROM videos\_by\_title\_year WHERE title = 'Datastax - Who We Are' AND added\_year = 2005;*



What error does Cassandra return when you try to query on just title or just year? Why?

*SELECT \* FROM videos\_by\_title\_year WHERE added\_year = 2005;*

*InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"*



When we try to query on just "title" or just "year" without specifying both, Cassandra returns an error “InvalidRequest”. This is because the table's primary key is defined as a composite key ("title" and "added\_year"). To query the data, we must provide values for both parts of the composite key. Cassandra returns an error when we trying to query on just "title" or just "year" because the primary key expects both of them to be specified for proper data retrieval.

Meanwhile we can use:

*SELECT \* FROM videos\_by\_title\_year WHERE added\_year = 2013 ALLOW FILTERING;*

In this query, ALLOW FILTERING is added at the end to indicate that we are aware of the potential performance issues associated with filtering and we still want to execute the query. However, we should remember that using ALLOW FILTERING on large amount of data can have an impact on cluster's performance, and it's generally recommended to avoid “allow filtering” whenever possible.

A screenshot of a computer

Description automatically generated

Exercise 3:

Create the following (bad) table with the (crummy) primary key:

*CREATE TABLE bad\_videos\_by\_tag\_year (*

*tag text,*

*added\_year int,*

*added\_date timestamp,*

*title text,*

*description text,*

*user\_id uuid,*

*video\_id timeuuid,*

*PRIMARY KEY ((video\_id))*

*);*

As an aside, use DESCRIBE TABLE to view the structure of your bad\_videos\_by\_tag\_year table.

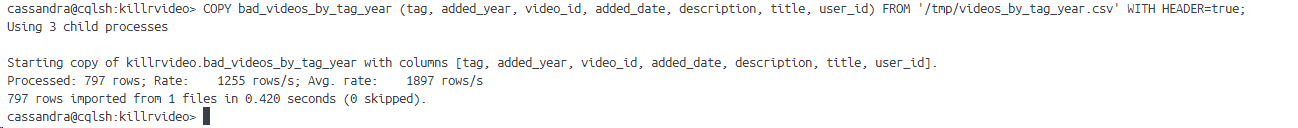
*DESCRIBE TABLE bad\_videos\_by\_tag\_year;*

A computer screen shot of a computer code

Description automatically generated

Import videos\_by\_tag\_year.csv file:

*COPY bad\_videos\_by\_tag\_year (tag, added\_year, video\_id, added\_date, description, title, user\_id) FROM '/tmp/videos\_by\_tag\_year.csv' WITH HEADER=true;*



*SELECT COUNT(\*) FROM bad\_videos\_by\_tag\_year;*

A screen shot of a computer

Description automatically generated

Notice the number of rows in the bad\_videos\_by\_tag\_year does not match the number of rowb s imported from videos\_by\_tag\_year.csv. Since videos\_by\_tag\_year.csv duplicates video\_id for each unique tag and year per video, Cassandra upserted several records during the COPY. video\_id is not a proper partition key for this scenario.

Drop your nasty table.

*DROP TABLE bad\_videos\_by\_tag\_year;*

Create a table with the columns above to facilitate querying for videos by tag within a given year range returning the results in descending order by year.

*CREATE TABLE videos\_by\_tag\_year (*

*tag text,*

*added\_year int,*

*video\_id timeuuid,*

*added\_date timestamp,*

*description text,*

*title text,*

*user\_id uuid,*

*PRIMARY KEY (tag, added\_year, video\_id)*

*) WITH CLUSTERING ORDER BY (added\_year DESC);*

Load the data from the videos\_by\_tag\_year.csv

*COPY videos\_by\_tag\_year FROM '/tmp/videos\_by\_tag\_year.csv' WITH HEADER=true;*

A white background with black text

Description automatically generated

Check the number of rows in the videos\_by\_tag\_year table.

*SELECT COUNT(\*) FROM videos\_by\_tag\_year;*

A screenshot of a computer

Description automatically generated

Try running queries on the videos\_by\_tag\_year table to query on a specific tag and added year.

*SELECT \* FROM videos\_by\_tag\_year WHERE tag = 'trailer' AND added\_year = 2015;*

A screenshot of a computer

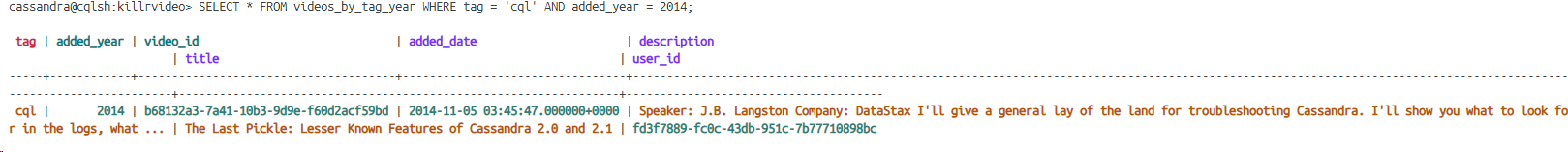
Description automatically generated

*SELECT count(\*) FROM videos\_by\_tag\_year WHERE tag = 'trailer' AND added\_year = 2015;*

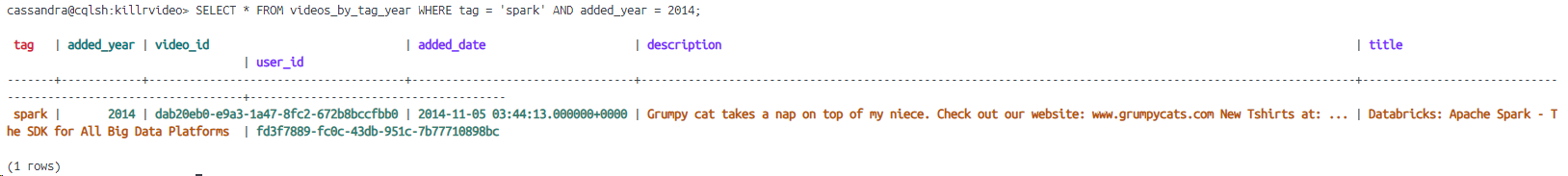
A white background with black text

Description automatically generated

*SELECT \* FROM videos\_by\_tag\_year WHERE tag = 'cql' AND added\_year = 2014;*

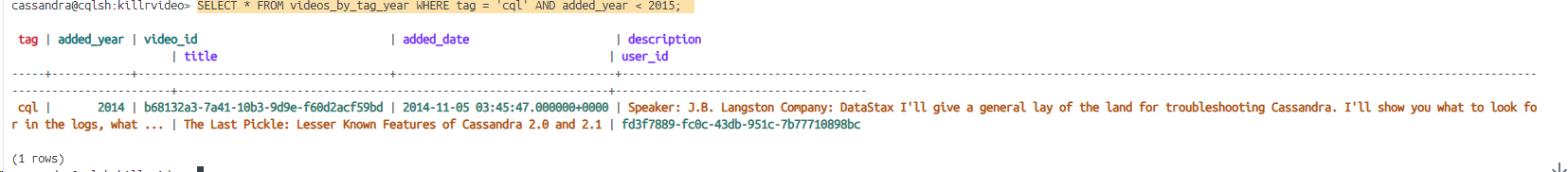


*SELECT \* FROM videos\_by\_tag\_year WHERE tag = 'spark' AND added\_year = 2014;*



Try querying for all videos with tag "cql" added before the year 2015. Notice you can do range queries on clustering columns.

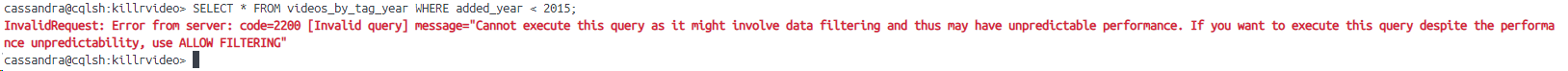
*SELECT \* FROM videos\_by\_tag\_year WHERE tag = 'cql' AND added\_year < 2015;*



Try querying for all videos added before 2015. The query will fail. What error message does cqlsh report? Why did the query fail whereas the previous query worked?

*SELECT \* FROM videos\_by\_tag\_year WHERE added\_year < 2015;*

*InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"*



The reason for the failure is that Cassandra requires us to specify an equality condition for all preceding clustering columns before we can perform a range query on a clustering column. In the case of the "videos\_by\_tag\_year" table, "tag" is the preceding clustering column, and "added\_year" is the clustering column we want to use in a range query. Since we didn't provide an equality condition for "tag," Cassandra cannot perform the query.

In contrast, the previous query that worked included both "tag" and "added\_year" in the WHERE clause, so it is a valid query.

### **Exercise 4:**

Run the TRUNCATE command to erase the data from the videos table.

*TRUNCATE videos;*

Alter the videos table to add a tags column.

*ALTER TABLE videos ADD tags set<text>;*

Load the data from the videos.csv

*COPY videos FROM '/tmp/videos.csv' WITH HEADER=true;*

Create a user defined type called video\_encoding with the structure given above.

*CREATE TYPE video\_encoding (*

*bit\_rates set<text>,*

*encoding text,*

*height int,*

*width int*

*);*

Alter your table to add an encoding column of the video\_encoding type.

*ALTER TABLE videos ADD encoding video\_encoding;*

Load the data from the videos\_encoding.csv file.

*COPY videos (video\_id, encoding) FROM '/tmp/videos\_encoding.csv' WITH HEADER=true;*A close-up of a computer code

Description automatically generated

Run a query to retrieve the first 10 rows of the videos table

*SELECT \* FROM videos LIMIT 10;*

A screenshot of a computer

Description automatically generated