Exercise 2.2: Clustering Columns

In this exercise, you will:

- Create a 'videos_by_tag_year' table that allows range scans and ordering by year

Background

There have been some interesting wrinkles in your quest to understand how Cassandra and CQL work. Although you have been able to complete your tasks to the letter, your team cannot query based on tag and year. Fortunately, your new understanding of clustering columns will help to improve your design. You decide to build a table that allows querying by 'tag' and 'added_year'.

The columns are as follows:

Column Name	Data Type
tag	text
added_year	int
video_id	uuid
added_date	timestamp
title	text
user_id	uuid

Steps

1. At the prompt, navigate to '/home/ubuntu/labwork/clustering' directory. Launch 'cqlsh' and switch to the 'killrvideo' keyspace.

In order to demonstrate a little bit more about clustering columns, we are first going to show you upserts.

2. To become an upsert ninja, 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))
);
```

3. As an aside, use DESCRIBE TABLE to view the structure of your 'bad_videos_by_tag_year' table.

NOTE: Notice the column order differs from the CREATE TABLE statement. Cassandra orders columns by partition key, clustering columns (shown later), and then alphabetical order of the remaining columns.

4. Execute the following COPY command to 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
'videos_by_tag_year.csv' WITH HEADER=true;
```

NOTE: We must explicitly list the column names because this table schema no longer matches the CSV structure.

NOTE: Notice the number of imported rows.

5. Now COUNT() the number of rows in the `bad_videos_by_tag_year`.

```
SELECT COUNT(*)
FROM bad_videos_by_tag_year;
```

Notice the number of rows in the 'bad_videos_by_tag_year' does not match the number of rows 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.

6. Drop your nasty table.

```
DROP TABLE bad_videos_by_tag_year;
```

Your mission is to restructure your table and allow users to query on 'tag' and possible 'added_year' ranges while avoiding upserts on import. You must also return your results in descending order of year.

7. 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.

8. We wrote most of the CREATE TABLE for you. Fill in the PRIMARY KEY and CLUSTERING ORDER BY.

```
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 ( )
) WITH CLUSTERING ORDER BY ( );
```

9. Load the data from the 'videos_by_tag_year.csv' file in the provided 'exercise=4' directory using the COPY command.

   ```
   COPY videos_by_tag_year FROM 'videos_by_tag_year.csv' WITH HEADER=true;
   ```

10. Check the number of rows in the 'videos_by_tag_year' table.

NOTE: The number of rows should match the number of rows imported by the COPY command. If not, you had upserts again and will need to adjust your PRIMARY KEY. Ask your instructor for help if necessary.

11. Try running queries on the 'videos_by_tag_year' table to query on a specific tag and added year.

Example queries:

tag	added_year
trailer	2015
cql	2014
spark	2014

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

13. 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?

14. Exit cqlsh.