Exercise 02.01: Denormalizing

In this exercise, you will:

- Create tables to support querying for videos by actor or genre

Background

With all of the success you've been having on the video sharing development team, you have been promoted and assigned to work on a high-priority project to incorporate movie content into the KillrVideo application.

Your new team is normalizing their video and actor metadata into separate tables and currently are stuck figuring out how to join tables in DataStax Enterprise. Having been around the block a few times, you know that JOINs are expensive and not supported. It is up to you to show your team the optimal way of performing these queries.

The video metadata is similar to what was in the video sharing domain:

Column Name	Data Type
video_id	timeuuid
added_date	timestamp
description	text
encoding	video_encoding
tags	set<text>
title	text
user_id	uuid

There is also the additional following metadata:

Column Name	Data Type
actor	text
character	text
genre	text

With this metadata, the data model must support the following queries:

- Q1: Retrieve the videos an actor has appeared in (newest first)
- Q2: Retrieve the videos within a particular genre (newest first)

Creating a new *videos_by_actor* Table

1. Navigate to the directory */labwork/denormalization*.

2. In `cqlsh`, you will need to create a new table called 'videos_by_actor' which will support query Q1.

3. Before we do that though, look more closely at the table above. Our 'encoding' column is actually something called a *User Defined Type* (or UDT for short). Fear not! We will be talking about these in the next exercise. For now, copy and paste this code to create the UDT so that our create table works correctly.

4. Creating the 'encoding' UDT.

```
CREATE TYPE IF NOT EXISTS video_encoding (
  encoding TEXT,
  height INT,
  width INT,
  bit_rates SET<TEXT>
);
```

5. We provided most of the CREATE TABLE for you, except for the PRIMARY KEY and the CLUSTERING ORDER BY.

```
CREATE TABLE videos_by_actor (
  actor TEXT,
  added_date TIMESTAMP,
  video_id TIMEUUID,
  character_name TEXT,
  description TEXT,
  encoding FROZEN<VIDEO_ENCODING >,
  tags SET<TEXT>,
  title TEXT,
  user_id UUID,
  PRIMARY KEY ((xxxx), xxxx, xxxx, xxxx)
) WITH CLUSTERING ORDER BY (xxxx);
```

6. Load the file *videos_by_actor.csv* into the 'videos_by_actor' table using the COPY command.

7. Run a query to retrieve the video information for a particular actor/actress (Tom Hanks, Denzel Washington, or one of your favorite actors who might be present).

8. Try using the SELECT command for just the actor and the 'added_date' columns. Notice the order of 'added_date'.

Create a *videos_by_genre* Table

1. In `cqlsh`, create a new table called 'videos_by_genre,' which will support query Q2. We provided most of the CREATE TABLE for you except the PRIMARY KEY and CLUSTERING ORDER BY.

```
CREATE TABLE videos_by_genre (
  genre text,
  added_date timestamp,
  video_id timeuuid,
  description text,
  encoding frozen<video_encoding>,
  tags set<text>,
  title text,
  user_id uuid,
  PRIMARY KEY (xxxx)
) WITH CLUSTERING ORDER BY (xxxx);
```

2. Load the file *videos_by_genre.csv* into the 'videos_by_genre' table using the COPY command.

3. Run a query to retrieve the video information for a particular genre (e.g., Comedy, Fantasy, Western, Parody, Satire, Drama, etc.).

4. Exit `cqlsh`.

END OF EXERCISE