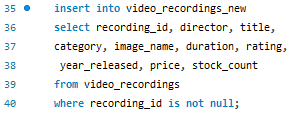
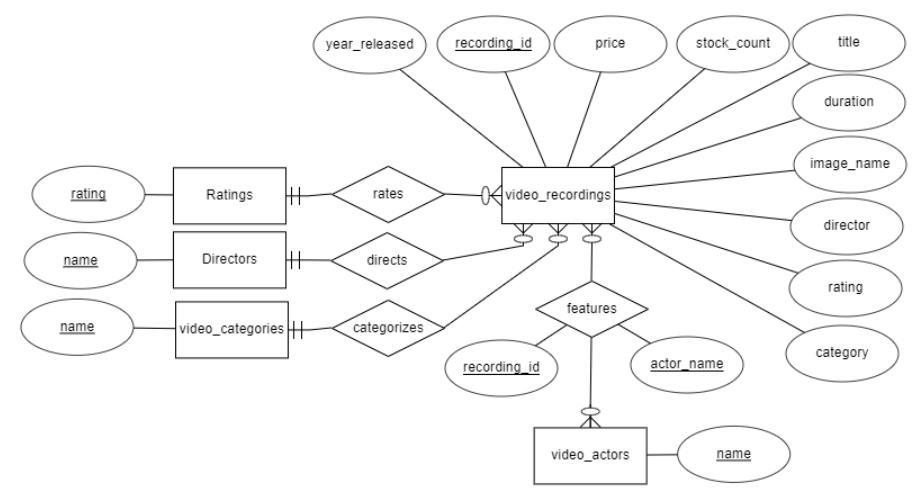
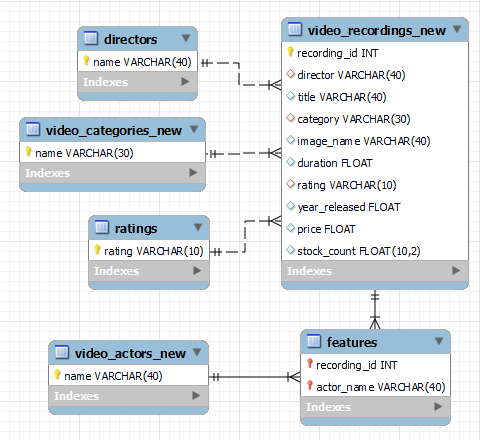
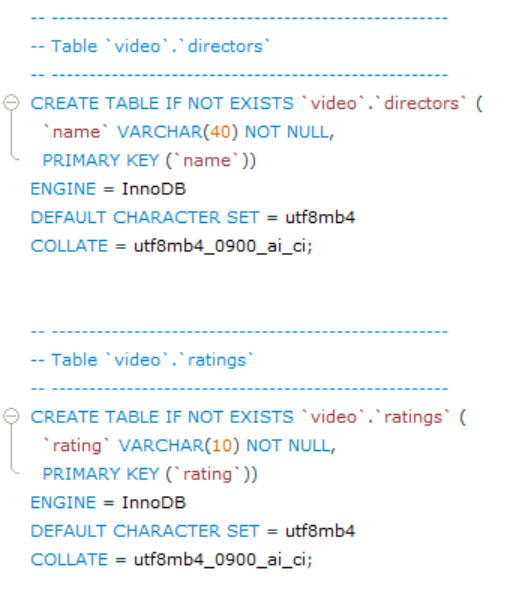
* Part I – Creating *Video* database:
  + Importing the unaltered data into the new data model:
    - Inserting data into video\_recordings\_new table
      * 
    - Inserting data into video\_categories\_new table:
      * 
    - Inserting data into video\_actors\_new table:
      * 
    - Inserting data into ratings table:
      * 
    - Inserting data into directors table:
      * 
    - Inserting data into features table:
      * 
  + Resulting ERD diagram of final model:



* + Resulting Relational Model:

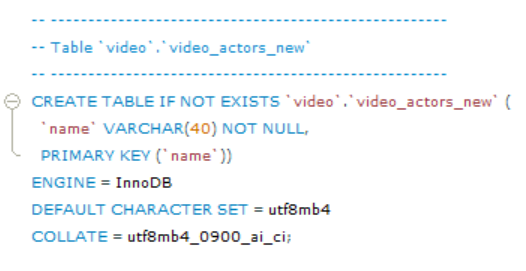


* + Final Model SQL create table scripts:

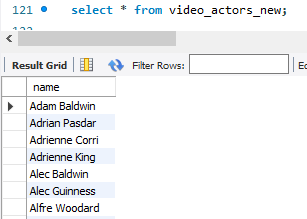
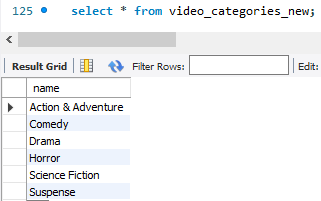
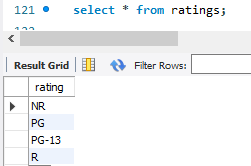
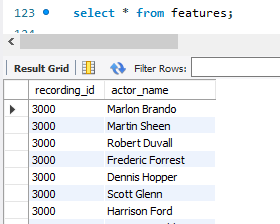
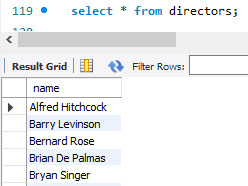








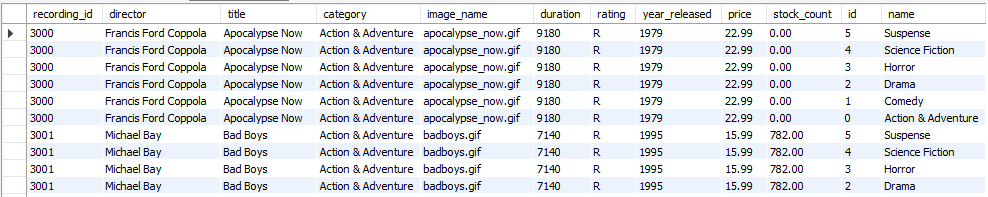


* + Running select statements to verify that data has been successfully imported:
    - *Video\_recordings\_new*, Rows returned: 55
      * 
    - *Video\_actors\_new*, Rows returned: 335
      * 
    - *Video\_categories\_new*, Rows returned: 6
      * 
    - *ratings*, Rows returned: 4
      * 
    - *features*, Rows returned: 372
      * 
    - *directors*, rows returned: 51
      * 
* Part I – Reflection Questions:
  + Why would you select csv- or tab-delimited files over the other?
    - The primary reason to use tab-delimited files over csv files is that tab-delimited files make rows and columns much more distinguishable. This is directly applicable to this class as the majority of the databases used in this class are relational databases. This is where columns hold attributes and each row acts as an entry. Therefore, by using tab-delimited files over csv files it can be much more intuitive to begin to build relational databases.
  + Why would I create the primary key index after the table has been created and the data imported versus defining the primary key in the table definition?
    - This is because if primary keys are defined prior to importing the data, constraint checks can fail once that data is attempted to be imported. For example, if the features table uses a combination of the foreign keys recording\_id, and actor\_name, for the primary key, then if only an actor\_name is attempted to be added, with the intention of next adding the recording\_id, the query would fail. This is because the query would fail the constraint that each entry must have a unique combination of recording\_id and actor\_name Another example pertains to foreign keys. If a full recording\_id entry is attempted to be imported, but the data contains directors that are not yet contained in the director table, then the import query will fail. This is due to the fact that the director attribute in recording\_id has a foreign key constraint that references the primary key attribute, name, in the directors table.
* Part II – Executing SQL Queries

1. Execute select \* from Video\_Recordings, Video\_Categories. Note the cross-product effect of joining two tables. Record the number of rows generated. Do all permutations of Video\_Recordings X Video\_Categories make sense? Explain.
   1. Query:



* 1. Results:



* 1. Analysis
     1. This query returned 330 rows for this given request. However, all of the returned permutations do not necessarily make much sense. The reason for this is that the query returned the cartesian product of each video category and each video recording, resulting in video recordings being concatenated with mismatched video categories.

1. Execute select \* from Video\_Recordings vr, Video\_Categories vc where vr.category=vc.name. Note the cross-product effect of joining two tables when restricted on the appropriate keys. Record the number of rows generated. Explain the purpose of the join.
   1. Query:

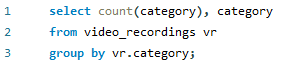


* 1. Results:

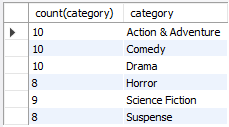


* 1. Analysis
     1. This query returned 55 rows for this given request. The purpose of this “join” is to ensure that the only combinations of video recordings and video categories appear where the video recording’s category matches the video category’s name. This corrects the randomness of the permutations combined by the last query, and ensures only matching permutations are returned.

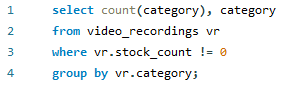
1. List the number of videos for each video category.
   1. Query:



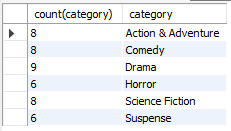
* 1. Results:



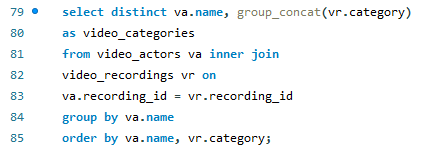
1. List the number of videos for each video category where the inventory is non-zero.
   1. Query:



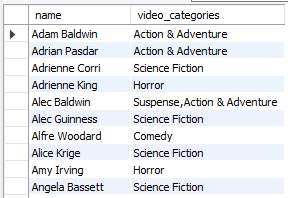
* 1. Results:



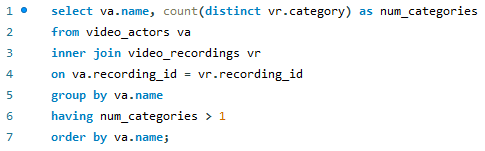
1. For each actor, list the video categories.
   1. Query:



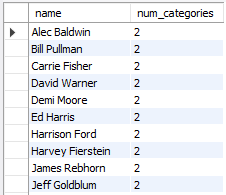
* 1. Results: **335 rows**



1. Which actors have appeared in movies in different video categories?
   1. Query:



* 1. Results: **25 rows**



1. Which actors have not appeared in a comedy?
   1. Query:



* 1. Results: **265 rows**



1. Which actors have appeared in comedy and action adventure movies?
   1. Query:



* 1. Results:

