CPSC 321: Assignment 7 *Due: Thursday, Nov. 12*

Reading. Continue the Chapter 6 reading as needed in ZyBooks and complete all of the corresponding reading "*activities*". You should also complete the four "labs" in Chapter 7.

Technical Work. Use the database (`cpsc321`) that has been created on the DB server for the course to answer the following questions. Within the `cpsc321` database are a number of tables that have been created and populated to represent simple film rental data. For reference, the tables are listed below. Note that the film rental_rate attribute is different than the payment amount for a rental. The rental rate is the amount of the rental for a certain duration, whereas the payment amount is for a specific duration and is what the customer actually payed for the rental. As mentioned below, for this homework, we're interested in the actual payment amounts for rentals as opposed to the rental rate for films.

actor(actor_id, first_name, last_name, last_update)

category(category_id, name, last_update)

customer(customer_id, store_id, first_name, last_name, email, address_id, active, create_date, last_update)

film(film_id, title, description, release_year, language_id, original_language_id, rental_duration, rental_rate, length, replacement_cost, rating, special_features, last_update)

film_actor(actor_id, film_id, last_update)

film_category(film_id, category_id, last_update)

inventory(inventory_id, film_id, store_id, last_update)

payment(payment_id, customer_id, staff_id, rental_id, amount, payment_date, last_update)

rental(rental_id, rental_date, inventory_id, customer_id, return_date, staff_id, last_update)

staff(staff_id, first_name, last_name, address_id, picture, email, store_id, active, username, password, last_update)

store(store_id, manager_staff_id, address_id, last_update)

For each question below, write a single SQL query to answer the question and show the result of the query answer against the above database. If the answer has more than 10 rows, you only need to show the first 10 rows (however, your query should work generally over the instance). In addition to the 10 rows listed, be sure to list the total number of rows the query returns. Note that you should only use subqueries when necessary (i.e., for queries that can't be expressed with "plain old" joins instead).

1. Find the number of films acted in by each actor/actress ordered from highest number of films to lowest. For each actor/actress, give their first and last name and the number of films they acted in. Your query must not use comma joins, but can use "join syntax" (e.g., JOIN USING or JOIN ON).

2. Find the total number of films by category ordered from most to least. Give the name of each category along with the number of corresponding films. Your query must not use comma joins, but can use "join syntax" (e.g., JOIN USING or JOIN ON).

3. Find all first and last names of customers that have rented at least four 'PG' rated films that they payed 2.99 to rent (i.e., the payment amount was 2.99). For each customer give the number of such films they've rented. The result should be sorted from most rented to least rented. Your query must not use comma joins, but can use "join syntax" (e.g., JOIN USING or JOIN ON).

4. Find the 'G' rated films that have been rented for the largest rental payment amount across all movie rentals. Return each matching film title and the corresponding rental payment amount. As part of your query, you must find the largest rental payment amount (i.e., you cannot assume this is a static or fixed value known when the query is written).

5. Find the film category (or categories if there is a tie) with the most number of 'PG' rated films. Your query cannot use limit and must only return the categories with the most number of films (i.e., not the second most, third most, and so on). Return the category name and the corresponding number of 'PG' rated films.

6. Find the 'G' rated film (or films if there is a tie) that have been rented more than the average number of times (for 'G' rated movies). Return the film titles and the number of times each film has been rented ordered from most number of rentals to least.

7. Write an SQL query using subqueries to find the actors/actresses that have not acted in a 'G' rated film.

8. Write an SQL query to find the film titles that all stores carry (i.e., in all store's inventories). Assume there can be any number of stores (i.e., you cannot assume a certain number of stores). Your query also cannot use COUNT(). (Hint: it isn't difficult using subqueries to find stores that don't have a particular film.)

9. Write an SQL query to find the percentage of 'G'-rated movies each actor/actress has acted in. Your query should return the id, first name, and last name of each actor/actress and the corresponding percentage. Your results should order actors/actresses from highest to lowest corresponding percentage. For this query, you only need to consider actors/actresses that have acted in at least one 'G'-rated movie.

10. Write an SQL query using an outer join to find all of the film titles that do not have any actors.

11. Write an SQL query using an outer join to find all of the film titles that are in a store's inventory but that have not been rented.

12. Write an SQL query to find the number of actors that acted in each film. Return the film_id and the number of associated actors. Based on your query result, how many films are there without an actor? Note that there should be more than one such film! Hint: `COUNT(columname)` only counts the number of non-NULL values in the values of `columname`.

13. Write a single query to find all of the movies that have the fewest number of actors and all of the movies that have the largest number of actors. Your query should return the film id, the number of actors for each such film, and the length of each film. The films should be sorted from most to fewest actors. Sort those films with the same number of actors in order of their film length (from shortest to longest). Hint: The result should have three different films, each with an actor count of 0.

14. Develop your own interesting "analytics" style query over the database that involves joins, aggregates, and subqueries. Explain the purpose of your query and give the result of executing it (if it is over 10 rows, provide only the first ten rows and the total number of rows of the query).

Create a `hw7.sql` script file that contains your for questions 1–14. You must also include comments in your script file, including a header file comment (with your name, the course, assignment number, and brief description) as well as comments for each query as needed. As part of your query comments include the problem number as well. If you are unsure what

is being asked for above, please ask for clarification on Piazza. Submit your `hw7.sql` file to the GitHub classroom for HW-7 along with a print-out showing your database tables and the results of running your queries (a link will be provided in Piazza).