

|  |  |
| --- | --- |
| **Course Number and Name:**  CSE 4308  Database Management Systems Lab | |
| **Student Name:**  Nafisa Maliyat | **Student ID:**  200042133 |
| **Report Submission Date:**  September, 2022 | **Name of Lab Instructor:**  Md. Bakhtiar Hasan, Lecturer, CSE  Zannatun Naim Sristy, Lecturer, CSE |

**Lab 4: Advanced Data Manipulation**

**Overview:**

|  |
| --- |
| This lab required us to perform queries on a given .sql file. The queries required multiple sub queries at times to get the result. The same result was also obtained using two methods at times, as some of the problem statements specified.  On the next pages, I have mentioned the following :   * the problem statement, * the problem analysis, * problems faced (if any) during solution of the tasks, and * the queries and their results on the SQL command line. |

**Task 1**

**Problem Statement:**

|  |
| --- |
| Find the name of the actors/actresses that are also directors (with and without ‘intersect’ clause). |

**Analysis of the problem:**

|  |
| --- |
| The problem statement required us to find the people that are both actors and directors i.e. the people that whose data is in both actor and director table. Using intersect between both tables is one way of doing it. Writing conditions to match the first and last names of the entries in both tables will produce the same result as using ‘intersect’ clause, given that there is no director and actor who are different people but has the same full name. |

**Any problems faced and how it was solved:**

|  |
| --- |
| There were no problems faced since the query was straightforward and simple. |

**Results:**

|  |
| --- |
| **with intersect:**    **without intersect :** |

**Task 2**

**Problem Statement:**

|  |
| --- |
| Find the list of all the first names stored in the database. |

**Analysis of the problem:**

|  |
| --- |
| The list of all the first names stored, which requires to select first names from actor and director table (since these are the only tables with first names) without selecting the duplicate entries in both tables. Use of ‘distinct’ ensured only unique names were selected. |

**Any problems faced and how it was solved:**

|  |
| --- |
| There were no problems faced since the query was straightforward and simple. |

**Results:**

|  |
| --- |
|  |

**Task 3**

**Problem Statement:**

|  |
| --- |
| Find the movie titles that did not receive any ratings (with and without ‘minus’ clause). |

**Analysis of the problem:**

|  |
| --- |
| This required the entries in movie that are not in the rating table. Where minus was not allowed, ‘not in’ was used to produce the same result. |

**Any problems faced and how it was solved:**

|  |
| --- |
| There were no problems faced since the query was straightforward and simple. |

**Results:**

|  |
| --- |
| **with minus:**    **without minus:** |

**Task 4**

**Problem Statement:**

|  |
| --- |
| Find the average rating of all movies. |

**Analysis of the problem:**

|  |
| --- |
| The result is the average of all the values of the attribute rev\_stars in the rating table. |

**Any problems faced and how it was solved:**

|  |
| --- |
| There were no problems faced since the query was straightforward and simple. |

**Results:**

|  |
| --- |
|  |

**Task 5**

**Problem Statement:**

|  |
| --- |
| Find the minimum rating for each movie and display them in descending order of rating. |

**Analysis of the problem:**

|  |
| --- |
| First the group by is used on the mov\_id attribute on rating and the aggregate function min was applied on rev\_stars. Finally, order by was used to display them in descending order according to the minimum rev\_stars. |

**Any problems faced and how it was solved:**

|  |
| --- |
| There were no problems faced since the query was straightforward and simple. |

**Results:**

|  |
| --- |
|  |

**Task 6**

**Problem Statement:**

|  |
| --- |
| Find the last name of actors/actresses and the number of ratings received by the movies that they played a role in. |

**Analysis of the problem:**

|  |
| --- |
| The actors’ last names who had a movie that also had a rating and the number of rating of the movies of those actors were the objective. |

**Any problems faced and how it was solved:**

|  |
| --- |
| Since three tables were involved, knowing which data had to match was a bit confusing. This was solved using natural join to join all three tables to get a table containing actors who had movies that also had ratings. From there, group by was used to form groups according to act\_id and the count of rev\_stars for each act\_id was easily obtained. |

**Results:**

|  |
| --- |
|  |

**Task 7**

**Problem Statement:**

|  |
| --- |
| Find the last name and average runtime of movies of different actors/actresses. Do not include any actor/actress who worked with ‘James Cameron’ (with and without ‘having’ clause). |

**Analysis of the problem:**

|  |
| --- |
| Firstly, a distinct list of actors was used to get act\_lastname and a sub query was used to find the average runtime of the movies performed by that actor. The sub query selected all the mov\_time of the movie provided that the mov\_id was associated with that particular actor in the casts table and an average function was used.  From these results, some of the rows were eliminated if found to be common with a second query which contained list of actors whose id were associated with the dir\_id (in the direction table) that had name ‘James Cameron’ in the director table.  For having, the first query was grouped by act\_id and the second query was in the having clause. In without having, the first query was made using matching of act\_id from casts and act\_id and where clause was used instead of having. |

**Any problems faced and how it was solved:**

|  |
| --- |
| Using group by, act\_lastname could not be selected because it was not within the group by function. This was solved using a dummy function max(act\_lastname) which would return the same thing since the act\_lastname in group formed by act\_id was same.  The number of rows in the query without ‘having’ clause was 29 while the second was 28 and upon further inspection, a name (Spacey) was in the list twice. Using ‘distinct’ for this solved the problem. |

**Results:**

|  |
| --- |
| **without having:**        **With having:** |

**Task 8**

**Problem Statement:**

|  |
| --- |
| Find the first name and last name of the director of the movie having the highest average rating (with and without ‘all’ clause). |

**Analysis of the problem:**

|  |
| --- |
| First a list of average rating of each movie is produced from rating table. Next the maximum value of the average rating is obtained using max aggregate function. The list of first and last names of the directors who had directed a movie is obtained whose directed movie has that maximum average rating.  Using all, the average rating of each movie is checked to find the average rating that is greater or equal to all of the average rating .Without all, the average rating is matched against the max of the average rating. |

**Any problems faced and how it was solved:**

|  |
| --- |
| Since there were multiple sub queries, it was difficult to organize the order. The problem was solved by thinking in steps and writing the smaller sub queries and keep nesting them one by one. |

**Results:**

|  |
| --- |
| **with all:** |

|  |
| --- |
| **without all:** |

**Task 9**

**Problem Statement:**

|  |
| --- |
| Find all the movie related information of movies acted and directed by the same person. |

**Analysis of the problem:**

|  |
| --- |
| First the dir\_id of those directors were selected who are also actors by matching their names. Next all movie related information is selected of those movies whose mov\_id was associated with those dir\_id (found by query in the first part) in the direction table. |

**Any problems faced and how it was solved:**

|  |
| --- |
| Since there were many selections, the view of the obtained result was broken to accommodate all the information. This problem could not be solved. |

**Results:**

|  |
| --- |
|  |

**Task 10**

**Problem Statement:**

|  |
| --- |
| Find the title and average rating of the movies that have average rating more than 7 (with and without using ‘having’ clause). |

**Analysis of the problem:**

|  |
| --- |
| First the mov\_id was chosen from rating table whose average rating was more than 7 and the mov\_id was used to find the mov\_title from movie table.  Using having, it was easily done by grouping the mov\_id in the rating table and applying the condition using having.  Without having, the same thing was done with more steps. The mov\_id and average rating was chosen after applying group by. Next only those mov\_id was selected whose average rating matched the condition. This was done using where clause which could be used because the table produced in the subquery had the columns mov\_id and avg\_rating. |

**Any problems faced and how it was solved:**

|  |
| --- |
| At first, it seemed that group by could not be used since having could not be used but later, with the help of online resources, it was discovered that if the average rating and movie id was considered as a table, conditions could be imposed on average rating using ‘where’ clause. Thus nested queries were used to accomplish this. |

**Results:**

|  |
| --- |
| **with having:** |

|  |
| --- |
| **without having:** |

**Task 11**

**Problem Statement:**

|  |
| --- |
| Find the title of the movies having average rating higher than the average rating of all the movies. |

**Analysis of the problem:**

|  |
| --- |
| First the average rating of all movies was found (also done in task 4). This was used to produce a list of mov\_id whose average rating was higher than this value.  Next the movie’s title and average rating was selected. Average rating was selected using a sub-query and mov\_id of the rating’s mov\_id and movie’s mov\_id was matched so average of the rating of a particular movie would be done and not all the movies.  From this list of movie name and average mov\_id, only those were selected that had their mov\_id in the list produced in the first step. |

**Any problems faced and how it was solved:**

|  |
| --- |
| In this problem, selecting the average rating was the problem since using group by would not allow movie title or any other attribute to be selected that were not grouped by. This was solved using sub query and this method was then followed for the rest of the previous queries as needed. |

**Results:**

|  |
| --- |
|  |

**Task 12**

**Problem Statement:**

|  |
| --- |
| Find the title and average rating of the movies without using the group by statement. |

**Analysis of the problem:**

|  |
| --- |
| From the movie and rating table, the mov\_title (who had a rating) and their average rating was selected.  This was done without grouping by taking the same table rating twice and assigning them alias R and R2 where the mov\_id matched in both table (which produces the effect of grouping) and this selected mov\_id matched the mov\_id from movie table so the correct mov\_title was selected. From this query tables with average ratings of a particular mov\_id was produced.  Finally only those mov\_title was included with ratings present in the rating table otherwise, mov\_title with no titles would show up with blank in average rating. |

**Any problems faced and how it was solved:**

|  |
| --- |
| The problems regarding this task was how to do it without group by. Internet resources were used to solve the problem. |

**Results:**

|  |
| --- |
|  |

**Task 13**

**Problem Statement:**

|  |
| --- |
| Find the actresses with the same first name. |

**Analysis of the problem:**

|  |
| --- |
| Only those act\_firstname in the actor table were selected that appeared more than once. This was done by grouping the first names and checking if their count was more than one (if they appeared more than once). |

**Any problems faced and how it was solved:**

|  |
| --- |
| There were no problems faced since the query was straightforward and simple. |

**Results:**

|  |
| --- |
|  |

**Task 14**

**Problem Statement:**

|  |
| --- |
| Find the title and maximum rating of the movies that has at least 10 reviews and has a female actress. One of the reviewers of the movie should be ‘Neal Wruck’. Do not include any movie that received less than 4 stars rating or any movies from directors that have did not direct more than one movie. |

**Analysis of the problem:**

|  |
| --- |
| The problem can be broken into conditions:   * **Movie title and maximum rating of the movies that has at least 10 reviews**: using movie table and selecting the maximum rating of those movies that has count of rev\_stars of at least 10. This is done using group by function to select the maximum rating and mov\_id and then nesting it to select the rating of those movies that matches the mov\_id of the mov\_title selected. * **Has a female actress**: the mov\_id is checked to see if it is in the list of movies that had female actresses (using cast and actor table). * **One of the reviewers of the movie should be ‘Neal Wruck’**: the mov\_id is checked to see if it is in the list of movies directed by Neal Wruck. (using rating table to find rev\_id and checking corresponding rev\_name in the reviewer table). * **Do not include any movie that received less than 4 stars rating**: using ‘not in’ clause to ensure that mov\_id is not in the list of those movies that has a less than 4 star rating (using rating table to produce a list of mov\_id that has a less than 4 star rating). * **Do not include any movies from directors that have did not direct more than one movie**: First those dir\_id with a count of mov\_id more than one are selected (using direction table and group by function). Next the movie id of those dir\_id is obtained and the original mov\_id is checked to ensure it is in that list.   Finally the mov\_id that satisfies all of these (using ‘and’) and its corresponding maximum rating is the result. |

**Any problems faced and how it was solved:**

|  |
| --- |
| The problem statement stated a lot of conditions. The interconnection of tables made it all a bit confusing to decide where to start. This was solved by breaking down the problem (as shown above) which made it easier to think. |

**Results:**

|  |
| --- |
|  |