1. **Explain the purpose of constraints and how they help maintain data integrity in a database. Provide examples of common types of constraints.**

Constraints are rules enforced on database tables to ensure the accuracy, validity, and consistency of the data. They define how data can be inserted, updated, or deleted, helping maintain data integrity by preventing invalid data from being stored in the database.

* + Primary key
  + Foreign key
  + Not Null
  + Unique
  + Check
  + Default

1. **Why would you apply the NOT NULL constraint to a column? Can a primary key contain NULL values? Justify your answer.**

The **NOT NULL constraint** is applied to ensure essential fields are never left blank. A **primary key** inherently disallows NULL values because it must uniquely and reliably identify each row in the table.

5. Explain the consequences of attempting to insert, update, or delete data in a way that violates constraints. Provide an example of an error message that might occur when violating a constraint.

Violating constraints leads to errors that prevent the operation from completing, ensuring data integrity. Each type of constraint has specific rules and associated error messages. Understanding these rules helps design and manage databases effectively, avoiding unintended consequences.

ERROR: Column 'emp\_name' cannot be null

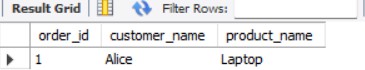
ERROR: Duplicate entry '1' for key 'emp\_id\_UNIQUE'

ERROR: Cannot add or update a child row: a foreign key constraint fails

8. Consider the following three tables:

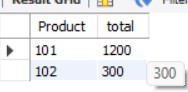
-- Write a query that shows all order\_id, customer\_name, and product\_name, ensuring that all products are

-- listed even if they are not associated with an order ans:



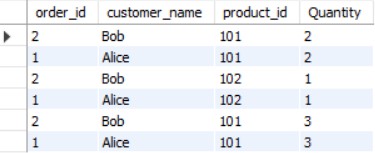
9 Write a query to find the total sales amount for each product using an INNER JOIN and the SUM() function.

ans :



10 . Write a query to display the order\_id, customer\_name, and the quantity of products ordered by each

-- customer using an INNER JOIN between all three tables. ans:

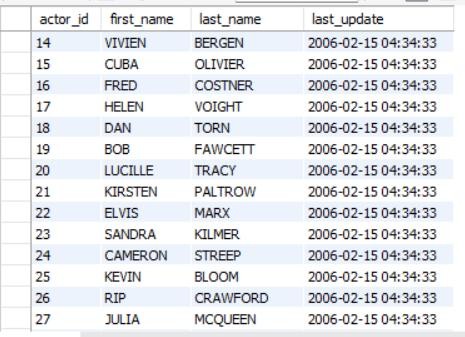


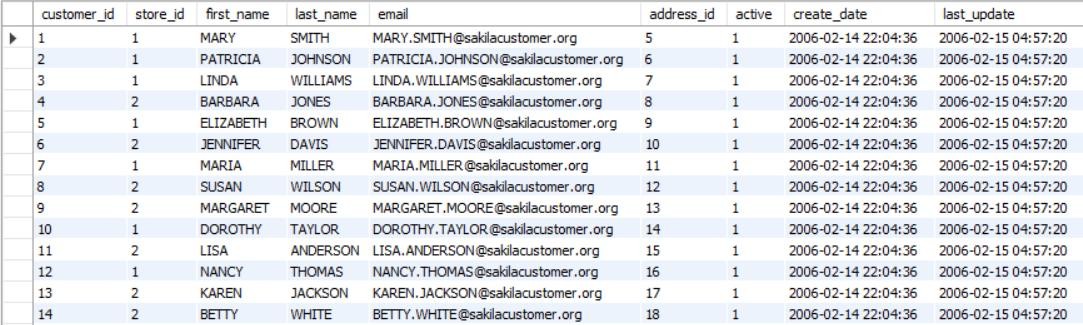
SQL COMMANDS

1 .-Identify the primary keys and foreign keys in maven movies db. Discuss the differences

|  |  |
| --- | --- |
| **Primary key:** | **Foreign key:** |
| actor\_id actor\_award\_id advisor\_id category\_id city\_id country\_id customer\_id film\_id inventory\_id investor\_id language\_id payment\_id rental\_id staff\_id store\_id | Category\_id Film\_id |

1. List all details of actors



1. **-List all customer information from DB.**
2. **-List different countries**

|  |
| --- |
| Afghanistan |
| Algeria |
| American Samoa |
| Angola |
| Anguilla |
| Argentina |
| Armenia |
| Australia |
| Austria |
| Azerbaijan |
| Bahrain |
| Bangladesh |
| Belarus |
| Bolivia |
| Brazil |
| Brunei |
| Bulgaria |
| Cambodia |
| Cameroon |
| Canada |
| Chad |
| Chile |
| China |
| Colombia |
| Congo, The Democratic Republic of the |
| Czech Republic |
| Dominican Republic |
| Ecuador |
| Egypt |
| Estonia |
| Ethiopia |
| Faroe Islands |
| Finland |
| France |

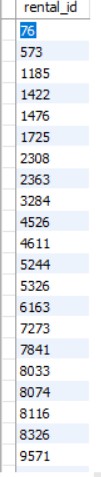
|  |
| --- |
| French Guiana |
| French Polynesia |
| Gambia |
| Germany |
| Greece |
| Greenland |
| Holy See (Vatican City State) |
| Hong Kong |
| Hungary |
| India |
| Indonesia |
| Iran |
| Iraq |
| Israel |
| Italy |
| Japan |
| Kazakstan |
| Kenya |
| Kuwait |
| Latvia |
| Liechtenstein |
| Lithuania |
| Madagascar |
| Malawi |
| Malaysia |
| Mexico |
| Moldova |
| Morocco |
| Mozambique |
| Myanmar |
| Nauru |
| Nepal |
| Netherlands |
| New Zealand |
| Nigeria |
| North Korea |
| Oman |
| Pakistan |
| Paraguay |
| Peru |
| Philippines |
| Poland |
| Puerto Rico |
| Romania |
| Runion |
| Russian Federation |

|  |
| --- |
| Saint Vincent and the Grenadines |
| Saudi Arabia |
| Senegal |
| Slovakia |
| South Africa |
| South Korea |
| Spain |
| Sri Lanka |
| Sudan |
| Sweden |
| Switzerland |
| Taiwan |
| Tanzania |
| Thailand |
| Tonga |
| Tunisia |
| Turkey |
| Turkmenistan |
| Tuvalu |
| Ukraine |
| United Arab Emirates |
| United Kingdom |
| United States |
| Venezuela |
| Vietnam |
| Virgin Islands, U.S. |
| Yemen |
| Yugoslavia |
| Zambia |

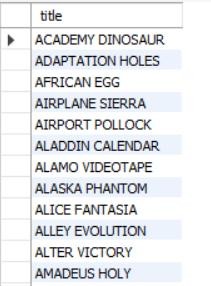
1. **-Display all active customers.**



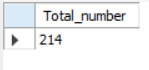
6. List of all rental IDs for customer with ID 1.

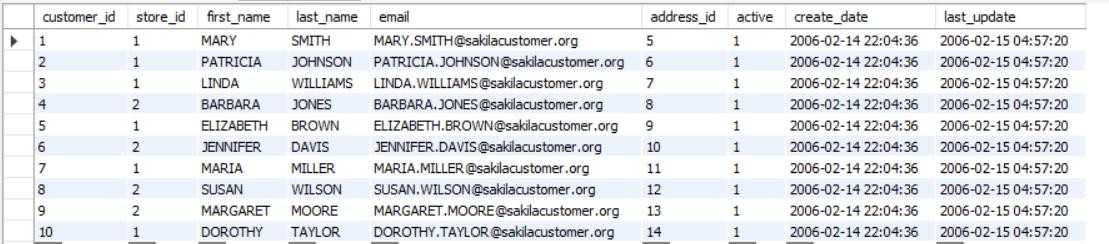


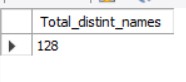
1. **- Display all the films whose rental duration is greater than 5 .**



1. **- List the total number of films whose replacement cost is greater than $15 and less than $20.**

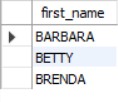


1. **- Display the count of unique first names of actors**

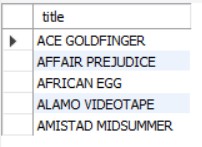


10- Display the first 10 records from the customer table .

1. **- Display the first 3 records from the customer table whose first name starts with ‘b’.**



1. **-Display the names of the first 5 movies which are rated as ‘G’.**



1. **Find all customers whose first name starts with "a".**

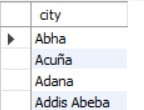
|  |
| --- |
| ANGELA |
| AMY |
| ANNA |
| AMANDA |
| ANN |
| ALICE |
| ASHLEY |
| ANDREA |
| ANNE |
| ANNIE |
| ANITA |
| AMBER |
| APRIL |
| ALICIA |
| AUDREY |
| ANNETTE |
| ANA |
| ALMA |
| AGNES |
| ARLENE |
| ALLISON |
| ANTHONY |
| ANDREW |
| ARTHUR |
| ALBERT |
| ADAM |
| AARON |
| ALAN |

|  |
| --- |
| ANTONIO |
| ALLEN |
| ALFRED |
| ALEXANDER |
| ALEX |
| ALVIN |
| ANGEL |
| ANDRE |
| ARNOLD |
| ADRIAN |
| ALLAN |
| ARMANDO |
| ALFREDO |
| ALBERTO |
| ANDY |
| AUSTIN |

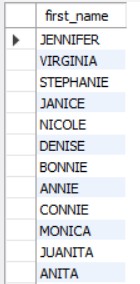
1. **Find all customers whose first name ends with "a".**

|  |
| --- |
| PATRICIA |
| LINDA |
| BARBARA |
| MARIA |
| LISA |
| SANDRA |
| DONNA |
| LAURA |
| JESSICA |
| CYNTHIA |
| ANGELA |
| MELISSA |
| BRENDA |
| ANNA |
| REBECCA |
| VIRGINIA |
| PAMELA |
| MARTHA |
| DEBRA |
| AMANDA |
| TERESA |
| GLORIA |
| CHRISTINA |
| THERESA |
| ANDREA |
| SARA |

1. **Display the list of first 4 cities which start and end with ‘a’ .**



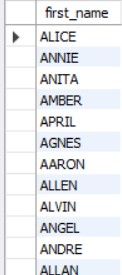
1. **Find all customers whose first name have "NI" in any position.**



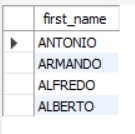
1. **Find all customers whose first name have "r" in the second position .**



18 - Find all customers whose first name starts with "a" and are at least 5 characters in length.



19- Find all customers whose first name starts with "a" and ends with "o".



1. **- Get the films with pg and pg-13 rating using IN operator.**



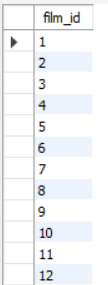
1. **- Get the films with length between 50 to 100 using between operator.**



1. **- Get the top 50 actors using limit operator.**



1. **- Get the distinct film ids from inventory table.**



Question 1:

-- Retrieve the total number of rentals made in the Sakila database.

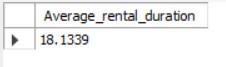
-- Hint: Use the COUNT() function.



Question 2:

-- Find the average rental duration (in days) of movies rented from the Sakila database.

-- Hint: Utilize the AVG() function.



Question 3:

-- Display the first name and last name of customers in uppercase.

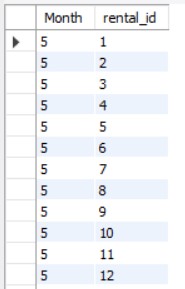
-- Hint: Use the UPPER () function.



Question 4:

-- Extract the month from the rental date and display it alongside the rental ID.

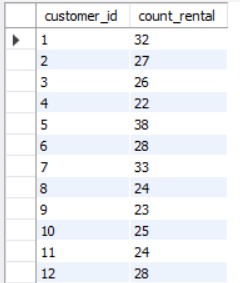
-- Hint: Employ the MONTH() function.



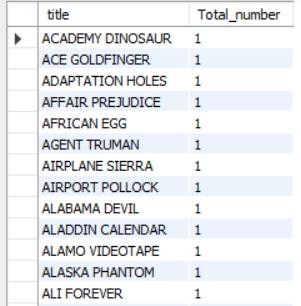
Question 5:

-- Retrieve the count of rentals for each customer (display customer ID and the count of rentals).

-- Hint: Use COUNT () in conjunction with GROUP BY.



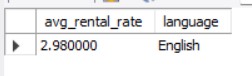
Question 7:-- Determine the total number of rentals for each category of movies.-- Hint: JOIN film\_category, film, and rental tables, then use COUNT () and GROUP BY.



-- Question 8:

-- Find the average rental rate of movies in each language.

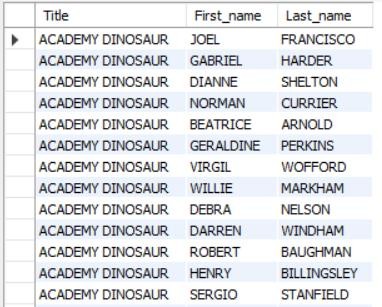
-- Hint: JOIN film and language tables, then use AVG () and GROUP BY.



Questions 9 -

-- Display the title of the movie, customer s first name, and last name who rented it.

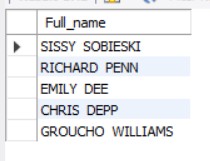
-- Hint: Use JOIN between the film, inventory, rental, and customer tables.



-- Question 10:

-- Retrieve the names of all actors who have appeared in the film "Gone with the Wind."

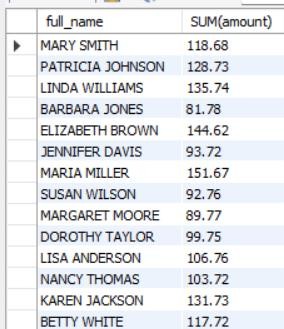
-- Hint: Use JOIN between the film actor, film, and actor tables.



-- Question 11:

-- Retrieve the customer names along with the total amount they've spent on rentals.

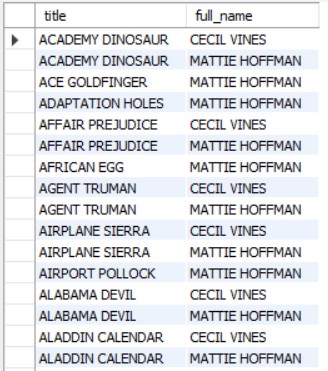
-- Hint: JOIN customer, payment, and rental tables, then use SUM() and GROUP BY



Question 12:

-- List the titles of movies rented by each customer in a particular city (e.g., 'London').

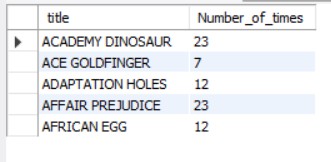
-- Hint: JOIN customer, address, city, rental, inventory, and film tables, then use GROUP BY.



-- Question 13:

-- Display the top 5 rented movies along with the number of times they've been rented.

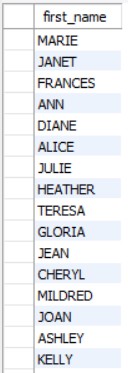
-- Hint: JOIN film, inventory, and rental tables, then use COUNT () and GROUP BY, and limit the results.



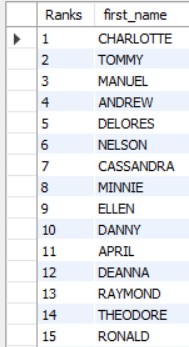
-- Question 14:

-- Determine the customers who have rented movies from both stores (store ID 1 and store ID 2).

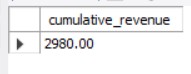
-- Hint: Use JOINS with rental, inventory, and customer tables and consider COUNT() and GROUP BY.



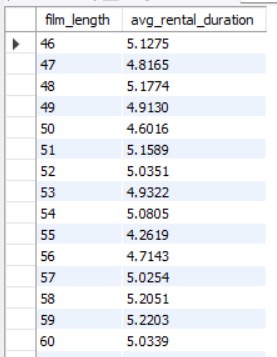
-- 1. Rank the customers based on the total amount they've spent on rentals.



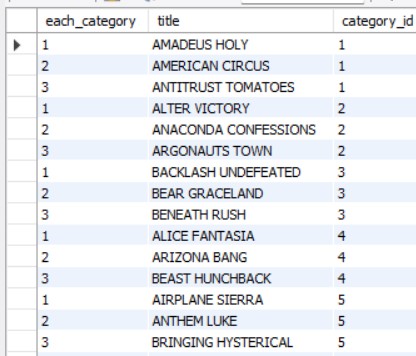
-- 2. Calculate the cumulative revenue generated by each film over time.



1. **Determine the average rental duration for each film, considering films with similar lengths.**

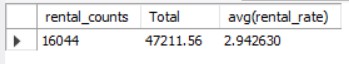


1. **Identify the top 3 films in each category based on their rental counts.**

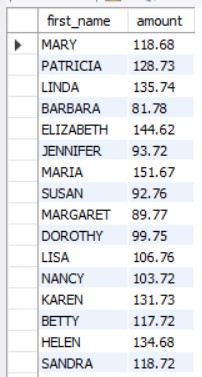


1. **Calculate the difference in rental counts between each customer's total rentals and the average rentals**

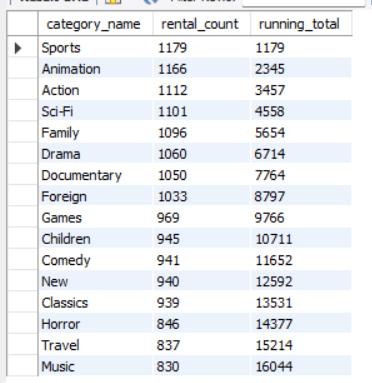
-- across all customers.



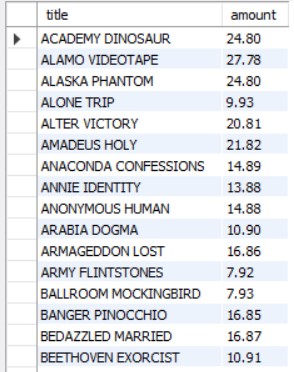
1. **Find the monthly revenue trend for the entire rental store over time.**
2. **Identify the customers whose total spending on rentals falls within the top 20% of all custommer;**



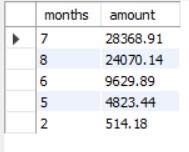
1. **Calculate the running total of rentals per category, ordered by rental count.**



9 Find the films that have been rented less than the average rental count for their respective categories.



10. Identify the top 5 months with the highest revenue and display the revenue generated in each month.



Normalisation and CTE

1. First Normal Form (1NF):
   1. Identify a table in the Sakila database that violates 1NF. Explain how you would normalize it to achieve 1NF.

**Ans**: each table cell contain a single value No repeating values in a group

No repeating groups

Each record(row) is unique

violating First Normal Form (1NF) could be the film table if it contains a repeating group or multivalued fields.

1. Second Normal Form (2NF):
   1. Choose a table in Sakila and describe how you would determine whether it is in 2NF.

If it violates 2NF, explain the steps to normalize it.

**Ans:** Should be INF No partial Dependency

Occur when there is composite Key

Steps to Normalize the Table to Achieve 2NF

To remove the partial dependency, split the table into two or more tables where each non-key attribute is fully dependent on the entire primary key.

Step 1: Split the film\_actor Table Original film\_actor Table:

film\_id actor\_id last\_update Split it into two tables:

A table to store the relationship between films and actors. A table to store the last\_update attribute.

Step 2: Create New Tables

film\_actor Table (for relationships):

film\_id actor\_id

film\_actor\_update Table (for last\_update):

film\_id last\_update

Step 3: Populate the New Tables

Remove last\_update from the original table and move it to film\_actor\_update, ensuring the relationship between film\_id and last\_update remains intact.

1. Third Normal Form (3NF):
   1. Identify a table in Sakila that violates 3NF. Describe the transitive dependencies

present and outline the steps to normalize the table to 3NF.

ANS:

Step 1: Identify Dependencies Direct Dependencies:

payment\_id → customer\_id, staff\_id, rental\_id, amount, payment\_date, customer\_name

Transitive Dependency:

payment\_id → customer\_id → customer\_name Step 2: Decompose the Table

To remove the transitive dependency, create a new table for customer information and move customer\_name to this table.

Step 3: Create New Tables

1. Normalization Process:
   1. Take a specific table in Sakila and guide through the process of normalizing it from the initial

unnormalized form up to at least 2NF.

**Normalization Process:**

Let’s take the **rental** table from the Sakila database and guide it through the normalization process.

**Step 1: Analyze the Initial Unnormalized Form (UNF)**

An **unnormalized form (UNF)** table may have repeating groups or multivalued attributes. Assume the

rental table in its UNF form looks like this:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **rental\_id** | **rental\_date** | **inventory\_id** | **customer\_id** | **return\_date** | **staff\_id** | **film\_title** | **category** |
| 1 | 2024-01-01  10:00:00 | 101 | 201 | 2024-01-03  15:00:00 | 1 | "Avengers" | Action |
| 2 | 2024-01-01  12:00:00 | 102 | 202 | 2024-01-03  17:00:00 | 2 | "Finding Nemo" | Animation |
| 3 | 2024-01-02  09:00:00 | 103 | 203 | 2024-01-04  11:00:00 | 1 | "Avengers" | Action |

**Violations in UNF:**

* + 1. **Repeating Group**: Columns like film\_title and category are attributes of the inventory\_id, not the rental\_id.
    2. **Multivalued Dependencies**: Data about the films (e.g., film\_title and category) is repeated for every rental.

**Step 2: Convert to First Normal Form (1NF)**

To convert the table to **1NF**:

1. Ensure atomicity (no multivalued attributes or repeating groups).
2. Break down repeating groups into separate rows or tables.

**Revised Table in 1NF:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **rental\_id** | **rental\_date** | **inventory\_id** | **customer\_id** | **return\_date** | **staff\_id** |
| 1 | 2024-01-01 10:00:00 | 101 | 201 | 2024-01-03 15:00:00 | 1 |
| 2 | 2024-01-01 12:00:00 | 102 | 202 | 2024-01-03 17:00:00 | 2 |
| 3 | 2024-01-02 09:00:00 | 103 | 203 | 2024-01-04 11:00:00 | 1 |

Create a separate table for film information (film table):

|  |  |  |
| --- | --- | --- |
| **inventory\_id** | **film\_title** | **category** |
| 101 | Avengers | Action |
| 102 | Finding Nemo | Animation |
| 103 | Avengers | Action |

**Step 3: Convert to Second Normal Form (2NF)**

To achieve **2NF**:

1. Ensure the table is already in **1NF**.
2. Eliminate **partial dependencies**, where non-key attributes depend on only part of a composite primary key.

**Key Analysis:**

* + **rental table primary key**: rental\_id (single-column key).
  + All columns in the rental table depend fully on rental\_id, so it is in 2NF.
  + **film table primary key**: inventory\_id.

o film\_title and category fully depend on inventory\_id, so it is also in 2NF.

**Normalized Tables (1NF to 2NF)**

1. **rental Table**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **rental\_id** | **rental\_date** | **inventory\_id** | **customer\_id** | **return\_date** | **staff\_id** |
| 1 | 2024-01-01 10:00:00 | 101 | 201 | 2024-01-03 15:00:00 | 1 |
| 2 | 2024-01-01 12:00:00 | 102 | 202 | 2024-01-03 17:00:00 | 2 |
| 3 | 2024-01-02 09:00:00 | 103 | 203 | 2024-01-04 11:00:00 | 1 |

1. **film Table**:

|  |  |  |
| --- | --- | --- |
| **inventory\_id** | **film\_title** | **category** |
| 101 | Avengers | Action |
| 102 | Finding Nemo | Animation |
| 103 | Avengers | Action |

6. CTE with Joins:

a. Create a CTE that combines information from the film and language tables to display the film title,

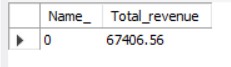
language name, and rental rate.



7 CTE for Aggregation:

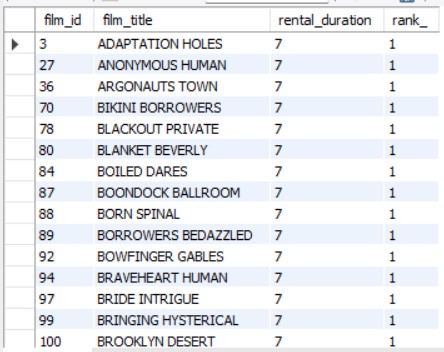
a. Write a query using a CTE to find the total revenue generated by each customer (sum of payments)

from the customer and payment tables.



-- 8 CTE with Window Functions:

-- a. Utilize a CTE with a window function to rank films based on their rental duration from the film table.



-- 9 CTE and Filtering:

-- a. Create a CTE to list customers who have made more than two rentals, and then join this CTE with the

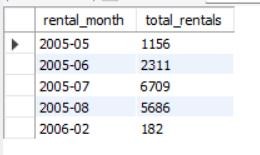
-- customer table to retrieve additional customer details



-- 10 CTE for Date Calculations:

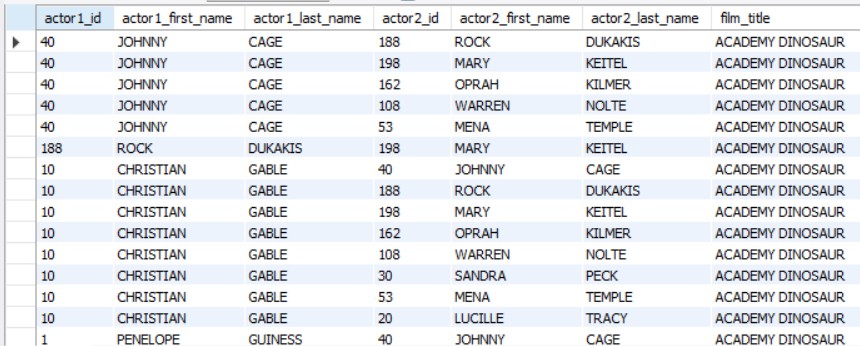
-- a. Write a query using a CTE to find the total number of rentals made each month, considering the

-- rental\_date from the rental table



11' CTE and Self-Join:

-- a. Create a CTE to generate a report showing pairs of actors who have appeared in the same film

-- together, using the film\_actor table.

-- 12. CTE for Recursive Search:

-- a. Implement a recursive CTE to find all employees in the staff table who report to a specific manager,

-- considering the reports\_to column

