/\* Normalisation and CTE queries\_assignment

Q 1. Identify a table in the Sakila database that violates 1NF. Explain how you would normalize it to achieve 1NF.

--> Actor\_award table violate nf1 formation in mavenmovies database violate 1nf formation we can normalised it by updating

the actor\_award table , avoid multivalued (each column has only atomic values) , Also we can create seperate tables for the

columns which contain multiple values etc

select awards from actor\_award ;

Q 2. 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.

--> select \* from film ;

film table from sakila databse violates 2nf because of the special features column special feature column on the

table violate 1nf and 2nf has a rule that table is in 1nf

Identify Partial Dependencies: all the non-prime attributes like title , discription , release\_year etc are fully dependent on the primary key

which is film id

we can create a another table and make them columns foreign keys and these foreign keys make reference to that film id table

by using these steps we can avoid 2 nf .

Q 3. Identify a table in Sakila that violates 3NF.

Describe the transitive dependencies present and outline the steps to normalize the table to 3NF.

--> if we saw the customer table in the sakila database we get to know that the column name address\_id is linked with store id

and both are non key attribute and 3nf stays that table is in 2 nf from and it ensure that all the non key attribute column on the

table are not related with each other (one non key attribute column related to other non key attribute column) so because of that it

violate 2 and 3 nf

steps to prevent 3nf ;

1, analyse the violation

2, create new table to store data

3 , update customer table (make store id as foreign key )

4 , update address info. (so it reference to the foreign key )

etc

Q 4. Take a specific table in Sakila and guide through the process of normalizing it from the initial

unnormalized form up to at least 2NF.

--> Initial Unnormalized Form:

The film\_actor table in the Sakila database is already in 1NF because it contains no repeating groups.

However, it can be further normalized to 2NF to eliminate partial dependencies.

2NF Violation:

The film\_actor table violates 2NF because the non-key attribute last\_update is transitively dependent on the primary key actor\_id through the actor table.

This means that last\_update can be determined from actor\_id without needing the entire primary key of film\_actor, which is actor\_id and film\_id.

Normalization Steps:

Identify the transitive dependency: Determine that last\_update is transitively dependent on actor\_id through the actor table.

Create a new table: Create a new table named actor\_update to store the actor\_id and last\_update columns.

Modify the original table: Remove the last\_update column from the film\_actor table.

Establish a relationship: Establish a foreign key relationship between the film\_actor table and the actor\_update table using the actor\_id column.

\*/

-- Q 5. Write a query using a CTE to retrieve the distinct list of actor names and the number of films they have

-- acted in from the actor and film\_actor tables.

WITH ActorFilmCount AS (

SELECT

a.actor\_id,

CONCAT(a.first\_name, ' ', a.last\_name) AS actor\_name,

COUNT(fa.film\_id) AS film\_count

FROM

actor a

JOIN

film\_actor fa ON a.actor\_id = fa.actor\_id

GROUP BY

a.actor\_id, actor\_name

)

SELECT

actor\_name,

film\_count

FROM

ActorFilmCount

ORDER BY

film\_count DESC, actor\_name ;

-- Q 6. Use a recursive CTE to generate a hierarchical list of categories and their subcategories from the category table in Sakila.

WITH RECURSIVE CategoryHierarchy AS (

SELECT

c.category\_id,

c.name AS category\_name,

NULL AS parent\_category\_id,

0 AS level

FROM

category c

WHERE

NOT EXISTS (

SELECT 1

## FROM film\_category fc