**Assignment 4**

**cte and normalisation**

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

-- Ans 1 . 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 ;

-- que 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 ;

-- ans 2 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 .

-- que 3 Identify a table in Sakila that violates 3NF. Describe the transitive dependencies present and outline the

-- steps to normalize the table to 3NF

ans 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

\*/

-- qur 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

-- ans

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 ;

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

-- ans

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

WHERE fc.category\_id = c.category\_id

)

UNION ALL

SELECT

c.category\_id,

c.name AS category\_name,

fc.category\_id AS parent\_category\_id,

ch.level + 1 AS level

FROM

category c

JOIN

film\_category fc ON c.category\_id = fc.category\_id

JOIN

CategoryHierarchy ch ON fc.film\_id = ch.category\_id

)

SELECT

category\_id,

category\_name,

parent\_category\_id,

level

FROM

CategoryHierarchy

ORDER BY

level, category\_id ;

-- que 7 Create a CTE that combines information from the film and language tables to display the films title , language , rental rate

-- ans

WITH FilmLanguageInfo AS (

SELECT

f.title AS film\_title,

l.name AS language,

f.rental\_rate

FROM

film f

JOIN

language l ON f.language\_id = l.language\_id

)

SELECT

film\_title,

language,

rental\_rate

FROM

FilmLanguageInfo;

-- que 8 Write a query using a CTE to find the total revenue generated by each customer (sum of payments) from customer and payment table

-- ans

WITH CustomerRevenue AS (

SELECT

c.customer\_id,

c.first\_name || ' ' || c.last\_name AS customer\_name,

SUM(p.amount) AS total\_revenue

FROM

customer c

LEFT JOIN

payment p ON c.customer\_id = p.customer\_id

GROUP BY

c.customer\_id, customer\_name

)

SELECT

customer\_id,

customer\_name,

COALESCE(total\_revenue, 0) AS total\_revenue

FROM

CustomerRevenue

ORDER BY

total\_revenue DESC;

-- que 9 Utilize a CTE with a window function to rank films based on their rental duration from the film table

-- ans

WITH RankedFilms AS (

SELECT

film\_id,

title,

rental\_duration,

RANK() OVER (ORDER BY rental\_duration DESC) AS rental\_duration\_rank

FROM

film

)

SELECT

film\_id,

title,

rental\_duration,

rental\_duration\_rank

FROM

RankedFilms

ORDER BY

rental\_duration\_rank;

-- que 10 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

-- ans

WITH CustomerRentals AS (

SELECT

customer\_id,

COUNT(rental\_id) AS rental\_count

FROM

rental

GROUP BY

customer\_id

HAVING

COUNT(rental\_id) > 2

)

SELECT

c.\*,

cr.rental\_count

FROM

customer c

JOIN

CustomerRentals cr ON c.customer\_id = cr.customer\_id

ORDER BY

cr.rental\_count DESC;

-- que 11 Write a query using a CTE to find the total number of rentals made each month, considering the rental date from the rental table

-- ans

WITH MonthlyRentals AS (

SELECT

DATE\_FORMAT(rental\_date, '%Y-%m') AS rental\_month,

COUNT(rental\_id) AS total\_rentals

FROM

rental

GROUP BY

rental\_month

)

SELECT

rental\_month,

total\_rentals

FROM

MonthlyRentals

ORDER BY

rental\_month;

-- que 12 Use a CTE to pivot the data from the payment rental\_date table to display the total payments made by each customer in

-- separate columns for different payment methods

-- ans since we dont have payment method column or any column that specify payment type we calculate total payments made by each customer

WITH CustomerPayments AS (

SELECT

customer\_id,

SUM(amount) AS total\_payments

FROM

payment

GROUP BY

customer\_id

)

SELECT

c.customer\_id,

c.first\_name,

c.last\_name,

cp.total\_payments

FROM

customer c

JOIN

CustomerPayments cp ON c.customer\_id = cp.customer\_id;

-- que 13 Create a CTE to generate a report showing pairs of actors who have appeared in the same film together, using the film\_actor

-- table

-- ans

WITH ActorPairs AS (

SELECT

fa1.actor\_id AS actor1\_id,

fa2.actor\_id AS actor2\_id,

COUNT(\*) AS films\_together

FROM

film\_actor fa1

JOIN film\_actor fa2 ON fa1.film\_id = fa2.film\_id AND fa1.actor\_id < fa2.actor\_id

GROUP BY

fa1.actor\_id, fa2.actor\_id

HAVING

COUNT(\*) > 0

)

SELECT

ap.actor1\_id,

ap.actor2\_id,

ap.films\_together

FROM

ActorPairs ap;