CSA0593

DATABASE MANAGEMENT SYSTEM.

NAME : S.DEEPIKA

REG NO. : 192324269

ASSIGNMENT – 3

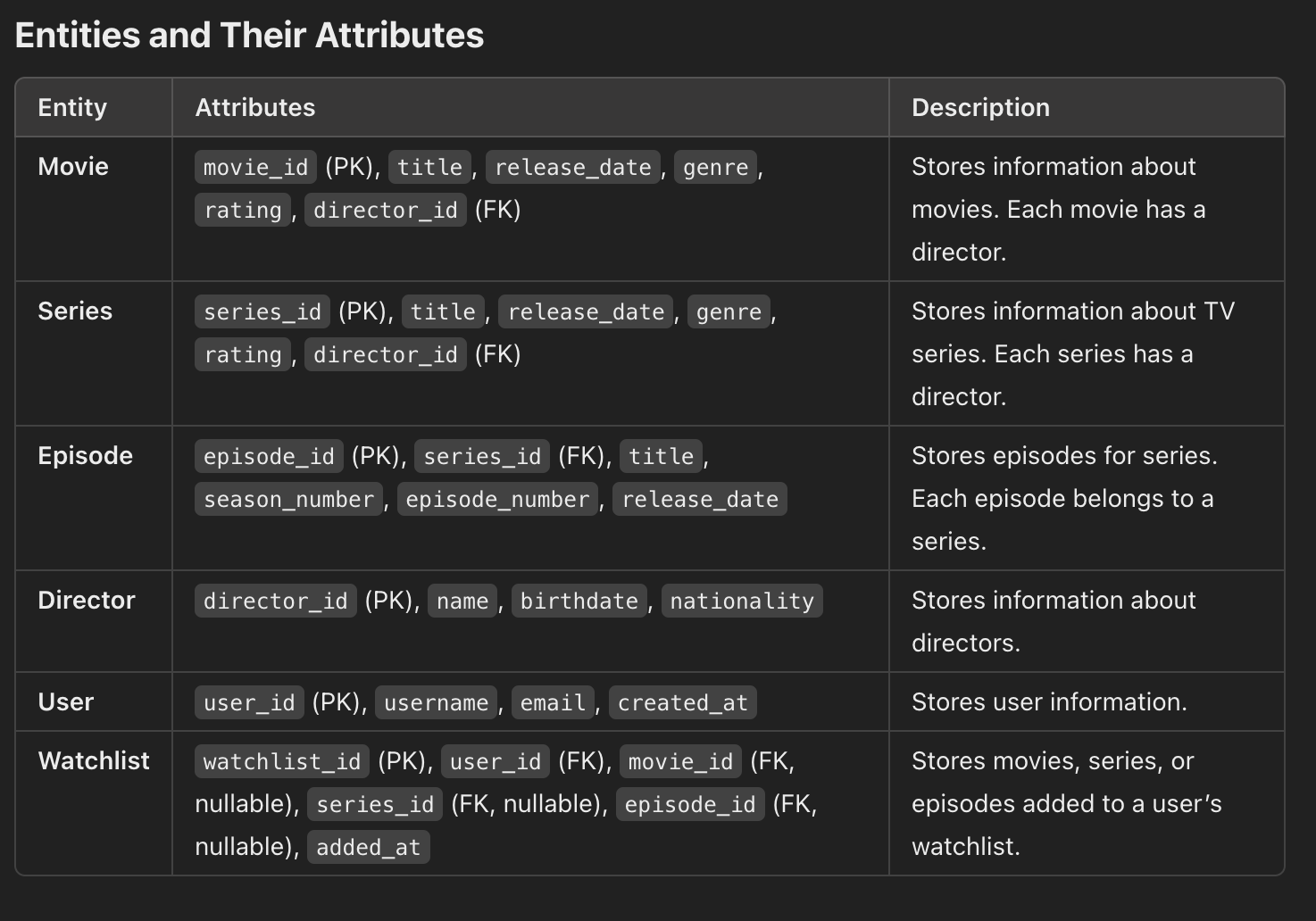
**Design a database to manage movies, series, directors, and user watchlists.**

* **Model tables for movies, series, directors, genres, and watchlists.**
* **Write stored procedures for adding movies or series to user watchlists and recording user ratings.**
* **Implement triggers to update view counts and movie rankings based on ratings or watch count.**
* **Write SQL queries to analyse movie/series popularity, director ratings, and user viewing habits.**

**Design a database to manage movies, series, directors, and user watchlists**

**Entity-Relationship Model (ER Diagram)**

To design a **database for managing movies, series, directors, and user watchlists**, we will identify the **entities** and their **relationships** in tabular form. The entities include **Movies**, **Series**, **Episodes**, **Directors**, **Users**, and **Watchlists**.



1. **Model tables for movies, series, directors, genres, and watchlists**

**1. Movies Table**

This table will store information specific to individual movies.

CREATE TABLE Movies (

movie\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(255) NOT NULL,

release\_date DATE,

director\_id INT,

genre\_id INT,

duration INT,

language VARCHAR(50),

rating DECIMAL(3, 1),

FOREIGN KEY (director\_id) REFERENCES Directors(director\_id),

FOREIGN KEY (genre\_id) REFERENCES Genres(genre\_id)

);

### 2. ****Series Table****

This table will store information for TV series, including the number of seasons.

CREATE TABLE Series (

series\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(255) NOT NULL,

release\_date DATE,

director\_id INT,

genre\_id INT,

seasons INT,

language VARCHAR(50),

rating DECIMAL(3, 1),

FOREIGN KEY (director\_id) REFERENCES Directors(director\_id),

FOREIGN KEY (genre\_id) REFERENCES Genres(genre\_id)

);

### 3. ****Directors Table****

This table will store information about directors.

CREATE TABLE Directors (

director\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

birth\_date DATE,

nationality VARCHAR(100)

);

### 4. ****Genres Table****

This table will store the different genres for movies or series.

CREATE TABLE Genres (

genre\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL

);

### 5. ****Watchlists Table****

This table will store user-specific watchlists, where users can add movies or series they plan to watch.

CREATE TABLE Watchlists (

watchlist\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT

movie\_id INT,

series\_id INT,

added\_on TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id),

FOREIGN KEY (movie\_id) REFERENCES Movies(movie\_id),

FOREIGN KEY (series\_id) REFERENCES Series(series\_id)

);

**b.)** **Write stored procedures for adding movies or series to user watchlists and recording user ratings :**

### 1. Stored Procedure for Adding Movies or Series to User Watchlist

This procedure adds a specified movie or series to a user’s watchlist by inserting a record in the Watchlist table.

DELIMITER //

CREATE PROCEDURE AddToWatchlist(

IN p\_user\_id INT,

IN p\_media\_id INT,

IN p\_media\_type ENUM('movie', 'series')

)

BEGIN

IF EXISTS (SELECT 1 FROM Watchlist WHERE user\_id = p\_user\_id AND media\_id = p\_media\_id AND media\_type = p\_media\_type) THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'This item is already in the watchlist.';

ELSE

INSERT INTO Watchlist (user\_id, media\_id, media\_type, added\_date)

VALUES (p\_user\_id, p\_media\_id, p\_media\_type, NOW());

END IF;

END //

DELIMITER ;

### 2. Stored Procedure for Recording User Ratings

This procedure allows users to rate movies or series by inserting or updating a rating in the Ratings table.

DELIMITER //

CREATE PROCEDURE RecordUserRating(

IN p\_user\_id INT,

IN p\_media\_id INT,

IN p\_media\_type ENUM('movie', 'series'),

IN p\_rating DECIMAL(2, 1)

)

BEGIN

IF p\_rating < 1.0 OR p\_rating > 5.0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Rating must be between 1.0 and 5.0.';

ELSE

IF EXISTS (SELECT 1 FROM Ratings WHERE user\_id = p\_user\_id AND media\_id = p\_media\_id AND media\_type = p\_media\_type) THEN

UPDATE Ratings

SET rating = p\_rating, rated\_date = NOW()

WHERE user\_id = p\_user\_id AND media\_id = p\_media\_id AND media\_type = p\_media\_type;

ELSE

INSERT INTO Ratings (user\_id, media\_id, media\_type, rating, rated\_date)

VALUES (p\_user\_id, p\_media\_id, p\_media\_type, p\_rating, NOW());

END IF;

END IF;

END //

DELIMITER ;

**c.) Implement triggers to update view counts and movie rankings based on ratings or watch count :**

### 1. Trigger to Update View Count

This trigger updates the view count of a movie or series whenever it is added to a user’s watchlist. Each addition to the watchlist indicates that the movie or series has been "viewed" or "interested," so we increment the view count accordingly.

DELIMITER //

CREATE TRIGGER UpdateViewCount

AFTER INSERT ON Watchlist

FOR EACH ROW

BEGIN

IF NEW.media\_type = 'movie' THEN

UPDATE Movies

SET view\_count = view\_count + 1

WHERE movie\_id = NEW.media\_id;

ELSEIF NEW.media\_type = 'series' THEN

UPDATE Series

SET view\_count = view\_count + 1

WHERE series\_id = NEW.media\_id;

END IF;

END //

DELIMITER ;

### 2. Trigger to Update Movie Ranking Based on Ratings

This trigger recalculates the ranking of a movie or series whenever a new rating is added or an existing rating is updated. Rankings could be based on the average rating or total rating score for each movie or series.

DELIMITER //

CREATE TRIGGER UpdateMovieRanking

AFTER INSERT ON Ratings

FOR EACH ROW

BEGIN

DECLARE avg\_rating DECIMAL(2, 1);

IF NEW.media\_type = 'movie' THEN

SELECT AVG(rating) INTO avg\_rating

FROM Rating

WHERE media\_id = NEW.media\_id AND media\_type = 'movie';

UPDATE Movies

SET average\_rating = avg\_rating

WHERE movie\_id = NEW.media\_id;

ELSEIF NEW.media\_type = 'series' THEN

SELECT AVG(rating) INTO avg\_rating

FROM Ratings

WHERE media\_id = NEW.media\_id AND media\_type = 'series';

UPDATE Series

SET average\_rating = avg\_rating

WHERE series\_id = NEW.media\_id;

END IF;

END //

DELIMITER ;

**d.) Write SQL queries to analyse movie/series popularity, director ratings, and user viewing habits:**

### 1. Analysing Movie/Series Popularity

This query retrieves the most popular movies or series based on view count and average rating. The result is ordered by view count and average rating in descending order.

SELECT

media\_id,

title,

media\_type,

view\_count,

average\_rating

FROM

(

SELECT movie\_id AS media\_id, title, 'movie' AS media\_type, view\_count, average\_rating

FROM Movies

UNION ALL

SELECT series\_id AS media\_id, title, 'series' AS media\_type, view\_count, average\_rating

FROM Series

) AS MediaPopularity

ORDER BY

view\_count DESC,

average\_rating DESC

LIMIT 10;

### 2. Analysing Director Ratings

This query calculates the average rating for each director based on the movies or series they directed. It provides insights into directors' performance by aggregating the ratings of their directed media.

SELECT

d.director\_id,

d.name AS director\_name,

AVG(r.rating) AS average\_rating

FROM

Directors d

JOIN

Movies m ON d.director\_id = m.director\_id

JOIN

Ratings r ON m.movie\_id = r.media\_id AND r.media\_type = 'movie'

GROUP BY

d.director\_id, d.name

UNION ALL

SELECT

d.director\_id,

d.name AS director\_name,

AVG(r.rating) AS average\_rating

FROM

Directors d

JOIN

Series s ON d.director\_id = s.director\_id

JOIN

Ratings r ON s.series\_id = r.media\_id AND r.media\_type = 'series'

GROUP BY

d.director\_id, d.name

ORDER BY

average\_rating DESC;

### 3. Analysing User Viewing Habits

This query examines user viewing habits by counting the number of movies and series each user has added to their watchlist. It helps in understanding user preferences for movies or series.

SELECT

w.user\_id,

u.name AS user\_name,

SUM(CASE WHEN w.media\_type = 'movie' THEN 1 ELSE 0 END) AS movies\_watched,

SUM(CASE WHEN w.media\_type = 'series' THEN 1 ELSE 0 END) AS series\_watched,

COUNT(\*) AS total\_watched

FROM

Watchlist w

JOIN

Users u ON w.user\_id = u.user\_id

GROUP BY

w.user\_id, u.name

ORDER BY

total\_watched DESC;

**CONCLUSION :**

The SQL queries provided enable comprehensive analysis of a movie and series database, focusing on key aspects such as content popularity, director performance, and user viewing habits. By examining view counts, average ratings, and genre preferences, the queries offer valuable insights into what drives user engagement and highlights the top-rated and most-watched content. These analyses can inform content recommendations, platform optimizations, and strategic decisions for enhancing user experience and boosting media engagement.