CINEMA- THE MOVIE RECOMMENDATION SYSTEM

A PROJECT REPORT

(21CSC205P – Database Management Systems)

Submitted by

Divy Jain [RA2311028010113]

Tejas Mishra [RA2311028010122]

Under the Guidance of

Dr. Kayalvizhi R

Associate Professor, Department of Networking and Communications

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

With specialization in Cloud Computing

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified to be the Bonafide work done by **Divy Jain (RA2311028010113) and Tejas Mishra (RA2311028010122)** of II-year B. Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date:

Faculty in Charge
Dr. Kayalvizhi R
Associate Professor
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur,
Chennai – 603 203

HEAD OF THE DEPARTMENT
Dr. M. Lakshmi
Professor & Head
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur,
Chennai – 603 203

ABSTRACT

A movie recommendation system is an intelligent application that helps users discover movies based on their preferences and viewing history. These systems utilize various filtering techniques, such as content-based filtering, collaborative filtering, and hybrid approaches, to analyze user behavior and suggest relevant movies. Content-based filtering recommends movies with similar characteristics to those previously watched, while collaborative filtering identifies patterns from multiple users with similar interests. Hybrid models combine both techniques for improved accuracy. With advancements in artificial intelligence and machine learning, modern recommendation systems leverage deep learning, natural language processing, and big data analytics to enhance personalization. This project explores the architecture, methodologies, and challenges of movie recommendation systems, highlighting their impact on user engagement and satisfaction in the entertainment industry.

PROBLEM STATEMENT

In today's world of streaming services and digital entertainment, users face an overwhelming choice of movies to watch. The lack of personalized suggestions often leads to decision fatigue and dissatisfaction, reducing user engagement. On the other hand, movie platforms struggle to deliver precise. Recommendations that align with users' unique preferences, viewing history, and behavior.

This project aims to build a Movie Recommendation System that provides tailored suggestions to users based on their interactions, preferences, and social feedback. The system will incorporate advanced techniques such as collaborative filtering, content-based filtering, and hybrid recommendation methods to improve accuracy and user satisfaction. It will also offer features like user reviews, ratings, watchlists, and activity tracking to enhance user experience.

By addressing these challenges, the system will simplify the movie discovery process, improve user retention, and create a personalized viewing experience.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROBLEM UNDERSTANDING

A **Movie Recommendation System** is an advanced software application that suggests movies to users based on their interests, preferences, and past interactions. It plays a crucial role in enhancing user experience by filtering and presenting personalized movie recommendations from a vast collection of films. With the ever-growing number of movies available on streaming platforms, it becomes overwhelming for users to decide what to watch. This project helps us by:

- Reducing decision fatigue
- Enhancing user engagement
- Increasing watch time on platforms
- Personalizing content based on user preferences

1.1.1 OBJECTIVE:

This project aims to create a **structured, database-driven** system that addresses the existing issues. The key objectives are:

1. Personalised Recommendations

Suggests movies based on past watch history. Allows users to create custom watchlists for tailored suggestions.

2. User-Friendly Interface

Simple navigation with intuitive search and filters. Responsive design for seamless experience on all devices.

3. Short Movie to Generate Interest

Provides a teaser or mini-story related to the main movie. Engages viewers with high-quality visuals and storytelling.

4. Secured Payment Methods

Supports encrypted transactions for user safety. Offers multiple trusted payment options (credit card, PayPal, etc.).

5. Reviews and Ratings

Displays user-generated ratings for better decision-making. Allows written reviews and video testimonials for authenticity.

1.1.2 USER REQUIREMENT:

1. Personalized Recommendations

Suggests movies based on watch history, genre preferences, and ratings. Provides AI-driven recommendations using collaborative filtering and content-based filtering.

2. User-Friendly Interface

Simple and intuitive navigation with minimal clicks. Advanced search and filter options (genre, release year, language, rating, etc.).

3. Short Movie Previews (Teasers)

Engaging movie teasers to generate interest. Personalized trailers based on user preferences.

4. Secure Payment Methods

Multiple secure payment gateways (Credit/Debit Card, PayPal, Digital Wallets).

5. Reviews and Ratings

Allows users to submit ratings and write reviews. Displays aggregated ratings for better decision-making.

6. Social and Community Features

Option to follow friends and influencers for movie recommendations. Shareable watchlists and reviews on social media.

7. Multi-Platform Accessibility

Available on web, mobile, and smart TV applications. Cross-platform synchronization for seamless transitions between devices.

8. Parental Controls

Content restriction options based on age and ratings. Kid-friendly recommendations with strict filtering.

9. Offline Viewing and Download Options

Ability to download movies for offline viewing. Storage management tools to optimize downloaded content.

1.1.3 CHALLENGES:

1. Data Collection and Processing

Gathering accurate and relevant user data while maintaining privacy. Processing large datasets to extract meaningful insights for recommendations.

2. Balancing Privacy and Personalization

Ensuring compliance with data protection laws (GDPR, CCPA, etc.). Handling sensitive user data while providing personalized recommendations.

3. Accuracy of Recommendations

Implementing advanced algorithms to deliver precise suggestions. Avoiding biased recommendations that may limit content diversity.

4. Scalability

Handling a growing number of users and movie data efficiently. Ensuring smooth performance during peak usage times.

5. UI/UX Optimization

Designing an interface that appeals to diverse user demographics. Ensuring accessibility features for differently-abled users.

6. Integration with Third-Party Services

Seamless integration with OTT platforms, payment gateways, and social media.

1.2 IDENTIFICATION OF ENTITY AND RELATIONSHIPS

1.2.1 ENTITIES:

1. Movies

- MovieID (Primary Key)
- MovieName
- Language
- Duration
- Rating
- AgeRestriction
- ReleaseDate

2. Genre

- GenreID (Primary Key)
- GenreName

3. MovieGenre

- MovieID (Foreign Key → Movies)
- GenreID (Foreign Key → Genre)

4. Cast

- CastID (Primary Key)
- Name
- Bio
- DOB

5. Directors

- DirectorID (Primary Key)
- Name
- Bio
- DOB

6. Ratings

- RatingID (Primary Key)
- MovieID (Foreign Key → Movies)
- MotionPictures

- TSeries
- IMDB

7. Awards

- AwardID (Primary Key)
- MovieID (Foreign Key → Movies)
- AwardType
- AwardName
- AwardYear

8. Region

- RegionID (Primary Key)
- RegionName

9. MovieRegion

- MovieID (Foreign Key → Movies)
- RegionID (Foreign Key → Region)

10. Users

- UserID (Primary Key)
- Name
- DOB

11. Subscription

- SubscriptionID (Primary Key)

- UserID (Foreign Key → Users)
- PlanName
- StartDate
- EndDate
- PlanPrice
- PlanStatus
- Details

12. Device

- DeviceID (Primary Key)
- OS
- Type
- UserID (Foreign Key → Users)

13. Reviews

- ReviewID (Primary Key)
- UserID (Foreign Key → Users)
- MovieID (Foreign Key → Movies)
- ReviewText
- Rating
- ReviewDate

14. Watchlist

- WatchlistID (Primary Key)
- UserID (Foreign Key → Users)
- MovieID (Foreign Key → Movies)
- AddedDate

15. Movie_Trailer

- TrailerID (Primary Key)
- MovieID (Foreign Key → Movies)
- TrailerLink
- UploadDate

16. Movie_Cast

- MovieID (Foreign Key → Movies)
- CastID (Foreign Key → Cast)
- Role

17. Movie_Director

- MovieID (Foreign Key → Movies)
- DirectorID (Foreign Key → Directors)

1.2.2 RELATIONSHIPS:

1. Movies and Related Tables

- Movies have Ratings (1:M)

- Movies have Movie Trailers (1:M)
- Movies win Awards (1:M)
- Movies belong to Movie Genre (M:M)
- Movie Genre defines Genre (1:M)
- Movies available Movie Region (M:M)
- Movie Region represents Region (1:M)

2. **Cast and Directors**

- Movies have Movie Cast (M:M)
- Movie Cast Has Cast (M:M)
- Movies directed by Movie Director (M:M)
- Movie Director has Directors (1:M)

3. **Users and Related Tables**

- Users write Reviews (1:M)
- Reviews reviewed Movies (M:1)
- Users have Subscription (1:1)
- Users add Watchlist (M:M)
- Movies are Watchlist(M:M)
- Users use Device (1:M)

1.3 CONSTRUCTION OF DB USING E-R MODEL

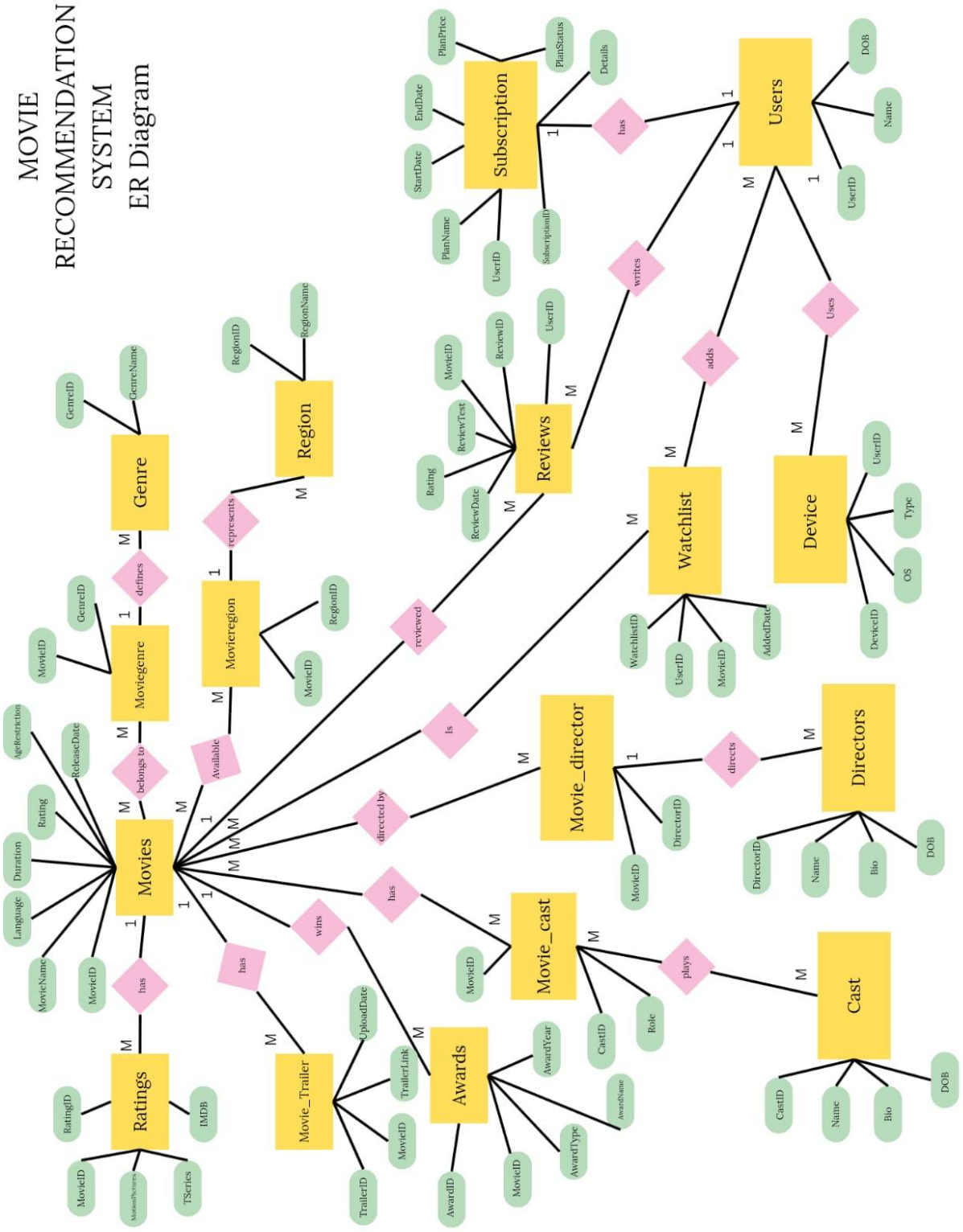


Fig. 1.1 ER Diagram

Constructing a database using the Entity-Relationship (ER) diagram for a Movie Recommendation System

Here's how the database construction process can be explained:

This ER diagram represents a **Movie Recommendation System**, showing the relationships between key entities such as **Movies, Users, Directors, Cast, Subscription, Genre, Awards, and Watchlist**. The **Movies** entity is central, connecting with **Directors, Cast, Genre, Ratings, and User Watchlists**. Users interact with movies through watch history, ratings, reviews, and subscriptions, which are linked to **payment plans and device usage**. The **Subscription** entity manages user memberships, while **Awards** track movie achievements. **Downloads and Watchlists** help users save and access content.

CHAPTER 2

RELATIONAL MODEL

2.1 DESIGN OF RELATIONAL SCHEMA

The process of converting an ER diagram to a relational model is crucial in database design. The ER diagram represents entities, attributes, and relationships graphically, while the relational model structures these into tables. This report outlines the systematic approach to transforming an ER diagram into a relational schema model.

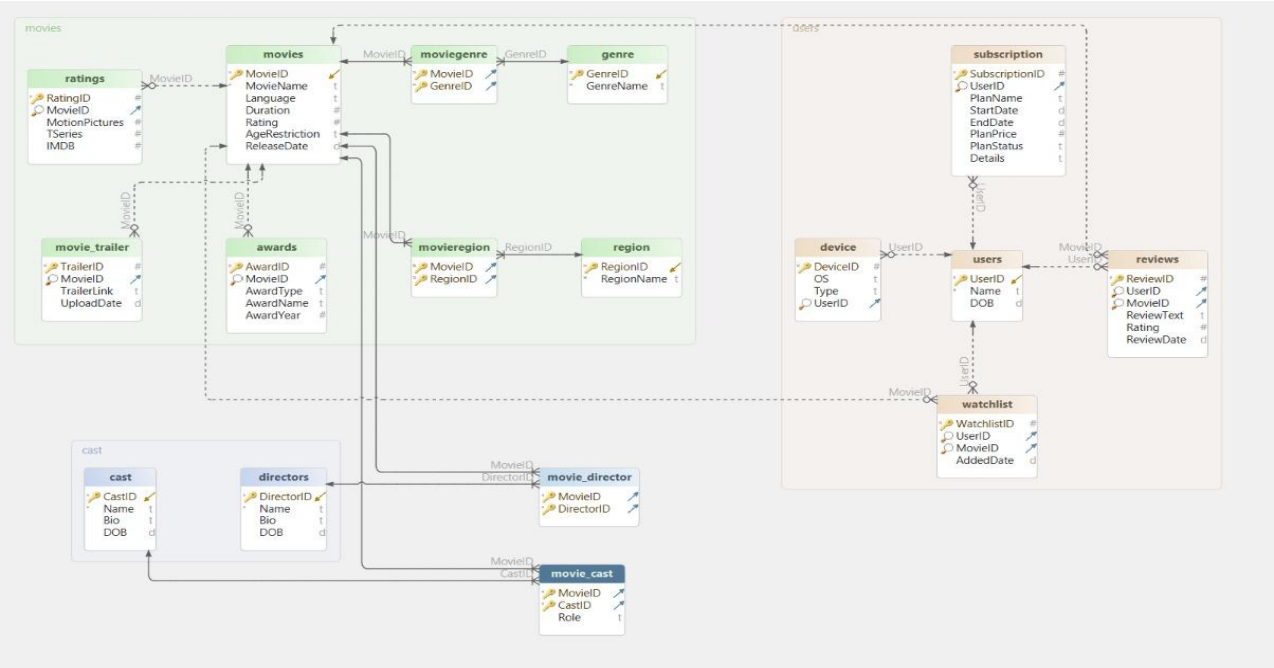


Fig. 2.1 Relational Schema

2.2 IDENTIFICATION OF ENTITIES

Entities represent real-world objects or concepts that need to be stored in a database. These can be classified as:

Strong Entities: Exist independently with a unique identifier (Primary Key)

Weak Entities: Depend on a strong entity and require a foreign key.

2.3 CONVERSION OF ENTITIES INTO TABLES

Each entity in the ER diagram is mapped to a relational table:

One example from our ER Diagram is:

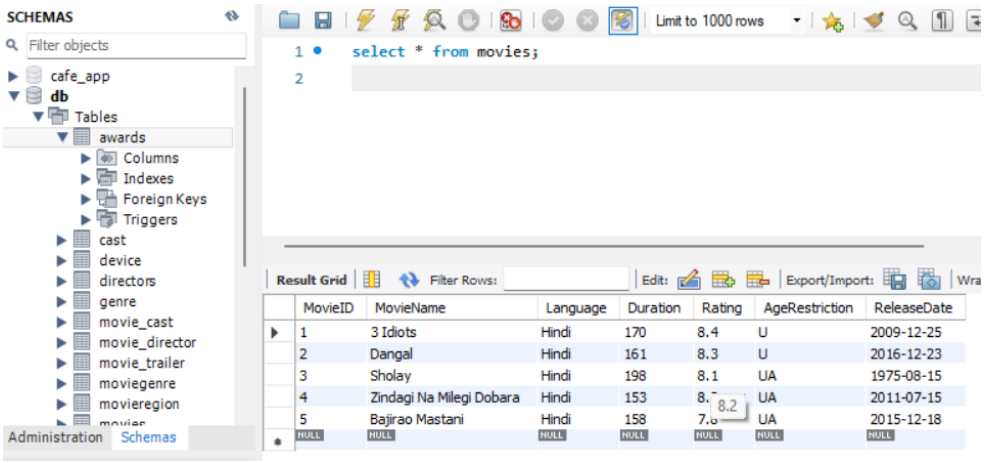


Fig. 2.2 Table Movie

2.4 MAPPING RELATIONSHIPS:

Relationships between Entities are implemented using foreign keys or separate tables for many-to-many relationships. In our ER Diagram there are:

ONE-ONE Relationship (1:1): A one-to-one relationship means that each entity in one table is related to a single entity in another table.

ONE-MANY Relationship (1:M): Relationships between entities are implemented using foreign keys or separate tables for many-to-many relationships**.**

2.6 QUERY AND THE OUTPUT FOR EVERY ENTITY-RELATION MODULE:

CREATE TABLE Movies (MovieID INT PRIMARY KEY AUTO_INCREMENT, MovieName VARCHAR (255) NOT NULL, Language VARCHAR (50), Duration INT, Rating FLOAT, AgeRestriction VARCHAR (10),

ReleaseDate DATE);

INSERT INTO movies (MovieName, Language, Duration, Rating, AgeRestriction, ReleaseDate) VALUES ('3 Idiots', 'Hindi', 170, 8.4, 'U', '2009-12-25'), ('Dangal', 'Hindi', 161, 8.3, 'U', '2016-12-23'), ('Sholay', 'Hindi', 198, 8.1, 'UA', '1975-08-15'), ('Zindagi Na Milegi Dobara', 'Hindi', 153, 8.2, 'UA', '2011-07-15'), ('Bajirao Mastani', 'Hindi', 158, 7.8, 'UA', '2015-12-18');

Select * from movies;

MovieID	MovieName	Language	Duration	Rating	AgeRestriction	ReleaseDate
1	3 Idiots	Hindi	170	8.4	U	2009-12-25
2	Dangal	Hindi	161	8.3	U	2016-12-23
3	Sholay	Hindi	198	8.1	UA	1975-08-15
4	Zindagi Na Milegi Dobara	Hindi	153	8.2	UA	2011-07-15
5	Bajirao Mastani	Hindi	158	7.8	UA	2015-12-18
6	3 Idiots	Hindi	170	8.4	U	2009-12-25
7	Dangal	Hindi	161	8.3	U	2016-12-23
8	Sholay	Hindi	198	8.1	UA	1975-08-15
9	Zindagi Na Milegi Dobara	Hindi	153	8.2	UA	2011-07-15
10	Bajirao Mastani	Hindi	158	7.8	UA	2015-12-18

Fig. 2.3 Table Movie

CREATE TABLE Genre (GenreID INT PRIMARY KEY AUTO_INCREMENT, GenreName VARCHAR (100) NOT NULL);

INSERT INTO genre (GenreID, GenreName) VALUES (1, 'Comedy'), (2, 'Drama'), (3, 'Sports'), (4, 'Action'), (5, 'Adventure'), (6, 'Road Trip'), (7, 'Historical');

Select * from genre;

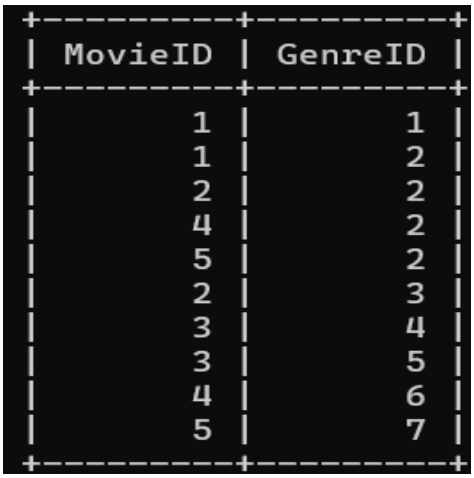
GenreID	GenreName
1	Comedy
2	Drama
3	Sports
4	Action
5	Adventure
6	Road Trip
7	Historical

Fig. 2.4 Table Genre

CREATE TABLE MovieGenre (MovieID INT, GenreID INT, PRIMARY KEY (MovieID, GenreID), FOREIGN KEY (MovieID) REFERENCES Movies(MovieID), FOREIGN KEY (GenreID) REFERENCES Genre(GenreID));

INSERT INTO moviegenre (MovieID, GenreID) VALUES (1, 1), (1, 2), (2, 2), (2, 3), (3, 4), (3, 5), (4, 2), (4, 6), (5, 2), (5, 7);

Select * from moviegenre;



MovieID	GenreID
1	1
1	2
2	2
4	2
5	2
2	3
3	4
3	5
4	6
5	7

Fig. 2.5 Table MovieGenre

CREATE TABLE Cast (CastID INT PRIMARY KEY AUTO_INCREMENT, Name VARCHAR(255) NOT NULL, Bio TEXT, DOB DATE);

INSERT INTO cast (CastID, Name, Bio, DOB) VALUES (1, 'Aamir Khan', 'Indian actor, director, and producer known for his work in Hindi films.', '1965-03-14'), (2, 'R. Madhavan', 'Indian actor known for his work in Hindi and Tamil cinema.', '1970-06-01'), (3, 'Sharman Joshi', 'Indian actor known for his roles in comedy and drama films.', '1979-04-28'), (4, 'Fatima Sana Shaikh', 'Indian actress who gained fame for her role in Dangal.', '1992-01-11'), (5, 'Sanya Malhotra', 'Indian actress known for her work in Bollywood.', '1992-02-25'), (6, 'Amitabh Bachchan', 'Legendary Indian actor known for his deep voice and intense performances.', '1942-10-11'), (7, 'Dharmendra', 'Veteran Bollywood actor known for his action and romantic roles.',

'1935-12-08'), (8, 'Hema Malini', 'Indian actress and politician, known as the Dream Girl of Bollywood.', '1948-10-16'), (9, 'Hrithik Roshan', 'Indian actor known for his dance skills and action-packed roles.', '1974-01-10'), (10, 'Farhan Akhtar', 'Indian actor, director, and singer known for multi-talented performances.', '1974-01-09'), (11, 'Priyanka Chopra', 'Indian actress, singer, and producer who gained international fame.', '1982-07-18'), (12, 'Deepika Padukone', 'Indian actress known for her work in Bollywood and Hollywood.', '1986-01-05');

Select * from cast;

CastID	Name	Bio	DOB
1	Aamir Khan	Indian actor, director, and producer known for his work in Hindi films.	1965-03-14
2	R. Madhavan	Indian actor known for his work in Hindi and Tamil cinema.	1970-06-01
3	Sharman Joshi	Indian actor known for his roles in comedy and drama films.	1979-04-28
4	Fatima Sana Shaikh	Indian actress who gained fame for her role in Dangal.	1992-01-11
5	Sanya Malhotra	Indian actress known for her work in Bollywood.	1992-02-25
6	Amitabh Bachchan	Legendary Indian actor known for his deep voice and intense performances.	1942-10-11
7	Dharmendra	Veteran Bollywood actor known for his action and romantic roles.	1935-12-08
8	Hema Malini	Indian actress and politician, known as the Dream Girl of Bollywood.	1948-10-16
9	Hrithik Roshan	Indian actor known for his dance skills and action-packed roles.	1974-01-10
10	Farhan Akhtar	Indian actor, director, and singer known for multi-talented performances.	1974-01-09
11	Priyanka Chopra	Indian actress, singer, and producer who gained international fame.	1982-07-18
12	Deepika Padukone	Indian actress known for her work in Bollywood and Hollywood.	1986-01-05

Fig. 2.6 Table Cast

CREATE TABLE Directors (DirectorID INT PRIMARY KEY AUTO_INCREMENT, Name VARCHAR(255) NOT NULL, Bio TEXT, DOB DATE);

INSERT INTO directors (DirectorID, Name, Bio, DOB) VALUES (1, 'Rajkumar Hirani', 'Indian film director known for his socially relevant and entertaining films.', '1962-11-20'), (2, 'Nitesh Tiwari', 'Indian filmmaker known for directing Dangal, one of the highest-grossing Indian films.', '1973-05-26'), (3, 'Ramesh Sippy', 'Veteran Indian film director known for directing the classic film Sholay.', '1947-01-23'), (4, 'Zoya Akhtar', 'Indian filmmaker known for making realistic and stylish Bollywood films.', '1972-10-14'), (5, 'Sanjay Leela Bhansali', 'Acclaimed Indian film director known for his visually stunning films.', '1963-02-24');

Select * from directors;

```
mysql> Select * from directors;
+------------+--------------------+-----------------------------------------------------------------------------------------+------------+
| DirectorID | Name               | Bio                                                                                     | DOB        |
+------------+--------------------+-----------------------------------------------------------------------------------------+------------+
|          1 | Rajkumar Hirani    | Indian film director known for his socially relevant and entertaining films.            | 1962-11-20 |
|          2 | Nitesh Tiwari      | Indian filmmaker known for directing Dangal, one of the highest-grossing Indian films.  | 1973-05-26 |
|          3 | Ramesh Sippy       | Veteran Indian film director known for directing the classic film Sholay.               | 1947-01-23 |
|          4 | Zoya Akhtar        | Indian filmmaker known for making realistic and stylish Bollywood films.                | 1972-10-14 |
|          5 | Sanjay Leela Bhansali | Acclaimed Indian film director known for his visually stunning films.                 | 1963-02-24 |
+------------+--------------------+-----------------------------------------------------------------------------------------+------------+
5 rows in set (0.02 sec)
```

Fig. 2.7 Table Directors

CREATE TABLE Ratings (RatingID INT PRIMARY KEY AUTO_INCREMENT, MovieID INT, MotionPictures FLOAT, TSeries FLOAT, IMDB FLOAT, FOREIGN KEY (MovieID) REFERENCES Movies(MovieID));

INSERT INTO ratings (MovieID, MotionPictures, TSeries, IMDB) VALUES (1, 8.5, 8.3, 8.4), (2, 8.2, 8.0, 8.3), (3, 8.0, 7.8, 8.1), (4, 8.3, 8.1, 8.2), (5, 7.9, 7.7, 7.8);

Select * from ratings;

```
mysql> select * from ratings;
+----------+---------+---------------+---------+------+
| RatingID | MovieID | MotionPictures | TSeries | IMDB |
+----------+---------+---------------+---------+------+
|        1 |       1 |           8.5 |     8.3 |  8.4 |
|        2 |       2 |           8.2 |       8 |  8.3 |
|        3 |       3 |             8 |     7.8 |  8.1 |
|        4 |       4 |           8.3 |     8.1 |  8.2 |
|        5 |       5 |           7.9 |     7.7 |  7.8 |
+----------+---------+---------------+---------+------+
5 rows in set (0.03 sec)
```

Fig. 2.8 Table Ratings

CREATE TABLE Awards (AwardID INT PRIMARY KEY AUTO_INCREMENT, MovieID INT, AwardType VARCHAR(100), AwardName VARCHAR(255), AwardYear INT, FOREIGN KEY (MovieID) REFERENCES Movies(MovieID));

INSERT INTO awards (MovieID, AwardType, AwardName, AwardYear) VALUES (1, 'National Film Award', 'Best Popular Film', 2010), (2, 'Filmfare

Award', 'Best Film', 2017), (3, 'Filmfare Award', 'Best Film of 50 Years', 2005), (4, 'IIFA Award', 'Best Film', 2012), (5, 'National Film Award', 'Best Director', 2016);

Select * from awards;

```
mysql> select * from awards;
+---------+---------+-------------------+----------------------+-----------+
| AwardID | MovieID | AwardType         | AwardName            | AwardYear |
+---------+---------+-------------------+----------------------+-----------+
|       1 |       1 | National Film Award | Best Popular Film    |      2010 |
|       2 |       2 | Filmfare Award    | Best Film            |      2017 |
|       3 |       3 | Filmfare Award    | Best Film of 50 Years |      2005 |
|       4 |       4 | IIFA Award        | Best Film            |      2012 |
|       5 |       5 | National Film Award | Best Director        |      2016 |
+---------+---------+-------------------+----------------------+-----------+
5 rows in set (0.02 sec)
```

Fig. 2.9 Table Awards

CREATE TABLE Region (RegionID INT PRIMARY KEY AUTO_INCREMENT, RegionName VARCHAR(100) NOT NULL);

INSERT INTO region (RegionID, RegionName) VALUES (1, 'India'),(2, 'USA'), (3, 'UK'), (4, 'Canada'), (5, 'Australia');

Select * from region;

```
mysql> Select * from region;
+----------+------------+
| RegionID | RegionName |
+----------+------------+
|        1 | India      |
|        2 | USA        |
|        3 | UK         |
|        4 | Canada     |
|        5 | Australia  |
+----------+------------+
5 rows in set (0.04 sec)
```

Fig. 2.10 Table Region

CREATE TABLE MovieRegion (MovieID INT, RegionID INT, PRIMARY KEY (MovieID, RegionID), FOREIGN KEY (MovieID) REFERENCES

Movies(MovieID), FOREIGN KEY (RegionID) REFERENCES Region(RegionID));

INSERT INTO movieregion (MovieID, RegionID) VALUES (1, 1), (2, 1), (3, 1), (4, 1), (5, 1);

Select * from movieregion;

MovieID	RegionID
1	1
2	1
3	1
4	1
5	1

Fig. 2.11 Table MovieRegion

CREATE TABLE Users (UserID INT PRIMARY KEY AUTO_INCREMENT,Name VARCHAR(255) NOT NULL,DOB DATE);

INSERT INTO users (UserID, Name, DOB) VALUES (1, 'Amit Sharma', '1995-04-12'), (2, 'Priya Verma', '1998-09-25'), (3, 'Rahul Mehta', '1990-06-15'), (4, 'Sneha Kapoor', '2000-01-30'), (5, 'Vikram Joshi', '1993-11-08');

Select * from users;

```
mysql> Select * from users;
+--------+---------------+------------+
| UserID | Name          | DOB        |
+--------+---------------+------------+
|      1 | Johnathan Doe | 1995-04-12 |
|      2 | Priya Verma   | 1998-09-25 |
|      3 | Rahul Mehta   | 1990-06-15 |
|      4 | Sneha Kapoor  | 2000-01-30 |
|      5 | Vikram Joshi  | 1993-11-08 |
|      6 | John Doe      | 1990-01-01 |
+--------+---------------+------------+
6 rows in set (0.03 sec)
```

Fig. 2.12 Table Users

CREATE TABLE Subscription (SubscriptionID INT PRIMARY KEY AUTO_INCREMENT, UserID INT, PlanName VARCHAR(100), StartDate DATE, EndDate DATE, PlanPrice DECIMAL(10,2), PlanStatus VARCHAR(50), Details TEXT, FOREIGN KEY (UserID) REFERENCES Users(UserID));

INSERT INTO subscription (UserID, PlanName, StartDate, EndDate, PlanPrice, PlanStatus, Details) VALUES (1, 'Basic', '2024-01-01', '2024-06-30', 9.99, 'Active', 'Basic plan with limited access'), (2, 'Premium', '2024-02-01', '2024-07-31', 14.99, 'Active', 'Premium plan with HD streaming'), (3, 'Family', '2024-03-01', '2024-08-31', 19.99, 'Active', 'Family plan with multiple devices'), (4, 'Student', '2024-04-01', '2024-09-30', 4.99, 'Inactive', 'Discounted plan for students'), (5, 'Annual', '2024-05-01', '2025-04-30', 99.99, 'Active', 'Annual subscription with full access');

Select * from subscription;

```
mysql> Select * from subscription;
+----------------+--------+----------+------------+------------+-----------+------------+-----------------------------------+
| SubscriptionID | UserID | PlanName | StartDate  | EndDate    | PlanPrice | PlanStatus | Details                           |
+----------------+--------+----------+------------+------------+-----------+------------+-----------------------------------+
|              1 |      1 | Basic    | 2024-01-01 | 2024-06-30 |      9.99 | Active     | Basic plan with limited access    |
|              2 |      2 | Premium  | 2024-02-01 | 2024-07-31 |     14.99 | Active     | Premium plan with HD streaming    |
|              3 |      3 | Family   | 2024-03-01 | 2024-08-31 |     19.99 | Active     | Family plan with multiple devices |
|              4 |      4 | Student  | 2024-04-01 | 2024-09-30 |      4.99 | Inactive   | Discounted plan for students      |
|              5 |      5 | Annual   | 2024-05-01 | 2025-04-30 |     99.99 | Active     | Annual subscription with full access |
+----------------+--------+----------+------------+------------+-----------+------------+-----------------------------------+
5 rows in set (0.02 sec)
```

Fig. 2.13 Table Subscription

CREATE TABLE Device (DeviceID INT PRIMARY KEY AUTO_INCREMENT, OS VARCHAR(100), Type VARCHAR(100), UserID INT, FOREIGN KEY (UserID) REFERENCES Users(UserID));

INSERT INTO device (OS, Type, UserID) VALUES ('Android', 'Smartphone', 1), ('iOS', 'Tablet', 2), ('Windows', 'Laptop', 3), ('macOS', 'Desktop', 4), ('Linux', 'Smart TV', 5);

Select * from device;

DeviceID	OS	Type	UserID
1	Android	Smartphone	1
2	iOS	Tablet	2
3	Windows	Laptop	3
4	macOS	Desktop	4
5	Linux	Smart TV	5

Fig. 2.14 Table Device

CREATE TABLE reviews (ReviewID INT AUTO_INCREMENT PRIMARY KEY,m UserID INT, MovieID INT, ReviewText TEXT, Rating DECIMAL(3,1), ReviewDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (UserID) REFERENCES users(UserID) ON DELETE CASCADE, FOREIGN KEY (MovieID) REFERENCES movies(MovieID) ON DELETE CASCADE);

INSERT INTO reviews (UserID, MovieID, ReviewText, Rating, ReviewDate) VALUES (1, 1, 'A brilliant and heartwarming film!', 9.0, '2024-03-15 10:30:00'), (2, 2, 'An inspiring story with great performances.', 8.5, '2024-03-16 14:20:00'), (3, 3, 'A true classic of Indian cinema.', 9.5, '2024-03-17 18:45:00'), (4, 4, 'Beautiful cinematography and a great adventure.', 8.0, '2024-03-18 21:10:00'), (5, 5, 'A visually stunning period drama.', 8.8, '2024-03-19 12:05:00');

Select * from reviews;

ReviewID	UserID	MovieID	ReviewText	Rating	ReviewDate
1	1	1	A brilliant and heartwarming film!	9.0	2024-03-15 10:30:00
2	2	2	An inspiring story with great performances.	8.5	2024-03-16 14:20:00
3	3	3	A true classic of Indian cinema.	9.5	2024-03-17 18:45:00
4	4	4	Beautiful cinematography and a great adventure.	8.0	2024-03-18 21:10:00
5	5	5	A visually stunning period drama.	8.8	2024-03-19 12:05:00

Fig. 2.15 Table Reviews

CREATE TABLE watchlist (WatchlistID INT AUTO_INCREMENT PRIMARY KEY, UserID INT, MovieID INT, AddedDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (UserID) REFERENCES users(UserID) ON DELETE CASCADE, FOREIGN KEY (MovieID) REFERENCES movies(MovieID) ON DELETE CASCADE);

INSERT INTO watchlist (UserID, MovieID, AddedDate) VALUES (1, 3, '2024-03-15 10:30:00'), (2, 1, '2024-03-16 14:20:00'), (3, 5, '2024-03-17 18:45:00'), (4, 2, '2024-03-18 21:10:00'), (5, 4, '2024-03-19 12:05:00');

Select * from watchlist;

```
mysql> Select * from watchlist;
+-------------+--------+---------+---------------------+
| WatchlistID | UserID | MovieID | AddedDate           |
+-------------+--------+---------+---------------------+
|           1 |      1 |       3 | 2024-03-15 10:30:00 |
|           2 |      2 |       1 | 2024-03-16 14:20:00 |
|           3 |      3 |       5 | 2024-03-17 18:45:00 |
|           4 |      4 |       2 | 2024-03-18 21:10:00 |
|           5 |      5 |       4 | 2024-03-19 12:05:00 |
+-------------+--------+---------+---------------------+
5 rows in set (0.02 sec)
```

Fig. 2.16 Table Watchlist

CREATE TABLE movie_trailer (TrailerID INT AUTO_INCREMENT PRIMARY KEY, MovieID INT, TrailerLink VARCHAR(255), UploadDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (MovieID) REFERENCES movies(MovieID) ON DELETE CASCADE);

INSERT INTO movie_trailer (MovieID, TrailerLink, UploadDate) VALUES (1, ‘https://www.youtube.com/watch?v=K0eDlFX9GMc’, ‘2009-07-16 10:00:00’), (2, ‘https://www.youtube.com/watch?v=x_7YlGv9u1g’, ‘2016-12-01 12:00:00’), (3, ‘https://www.youtube.com/watch?v=Vx5k9vPZyUI’, ‘1975-08-15 14:00:00’), (4, ‘https://www.youtube.com/watch?v=FJrpcDgC3zU’, ‘2011-06-15 18:00:00’), (5, ‘https://www.youtube.com/watch?v=eHOc-4D7MjY’, ‘2015-11-20 20:00:00’);

Select * from movie_trailer;

```
mysql> Select * from movie_trailer;
+-----------+---------+-----------------------------------------+---------------------+
| TrailerID | MovieID | TrailerLink                             | UploadDate          |
+-----------+---------+-----------------------------------------+---------------------+
|         1 |       1 | https://www.youtube.com/watch?v=K0eDlFX9GMc | 2009-07-16 10:00:00 |
|         2 |       2 | https://www.youtube.com/watch?v=x_7YlGv9u1g | 2016-12-01 12:00:00 |
|         3 |       3 | https://www.youtube.com/watch?v=Vx5k9vPZyUI | 1975-08-15 14:00:00 |
|         4 |       4 | https://www.youtube.com/watch?v=FJrpcDgC3zU | 2011-06-15 18:00:00 |
|         5 |       5 | https://www.youtube.com/watch?v=eHOc-4D7MjY | 2015-11-20 20:00:00 |
+-----------+---------+-----------------------------------------+---------------------+
5 rows in set (0.03 sec)
```

Fig. 2.17 Table Movie_trailer

CREATE TABLE movie_cast (MovieID INT, CastID INT, Role VARCHAR(255), PRIMARY KEY (MovieID, CastID), FOREIGN KEY (MovieID) REFERENCES movies(MovieID) ON DELETE CASCADE, FOREIGN KEY (CastID) REFERENCES `cast`(CastID) ON DELETE CASCADE);

INSERT INTO movie_cast (MovieID, CastID, Role) VALUES (1, 1, 'Rancho'), (1, 2, 'Farhan'), (1, 3, 'Raju'), (2, 1, 'Mahavir Singh Phogat'), (2, 4, 'Geeta Phogat'), (2, 5, 'Babita Phogat'), (3, 6, 'Jai'), (3, 7, 'Veeru'), (3, 8, 'Basanti'), (4, 9, 'Arjun'), (4, 10, 'Imran'), (5, 11, 'Kashibai'), (5, 12, 'Mastani');

Select * from movie_cast;

```
mysql> Select * from movie_cast;
+---------+--------+---------------------+
| MovieID | CastID | Role                |
+---------+--------+---------------------+
|       1 |      1 | Rancho              |
|       1 |      2 | Farhan              |
|       1 |      3 | Raju                |
|       2 |      1 | Mahavir Singh Phogat |
|       2 |      4 | Geeta Phogat        |
|       2 |      5 | Babita Phogat       |
|       3 |      6 | Jai                 |
|       3 |      7 | Veeru               |
|       3 |      8 | Basanti             |
|       4 |      9 | Arjun               |
|       4 |     10 | Imran               |
|       5 |     11 | Kashibai            |
|       5 |     12 | Mastani             |
+---------+--------+---------------------+
13 rows in set (0.02 sec)
```

Fig. 2.18 Table Movie_cast

CREATE TABLE movie_director (MovieID INT, DirectorID INT, PRIMARY KEY (MovieID, DirectorID), FOREIGN KEY (MovieID) REFERENCES movies(MovieID) ON DELETE CASCADE, FOREIGN KEY (DirectorID) REFERENCES directors(DirectorID) ON DELETE CASCADE);

INSERT INTO movie_director (MovieID, DirectorID) VALUES (1, 1), (2, 2), (3, 3), (4, 4), (5, 5);

Select * from movie_director;

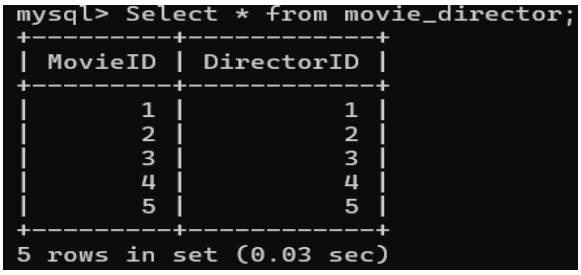


Fig. 2.19 Table Movie_director

2.7 CONCLUSION

Converting an ER diagram to a relational model ensures efficient data storage and integrity. By identifying entities, defining relationships, and applying constraints, we create a structured and reliable database system. This process enhances data consistency and optimizes query performance**.**

CHAPTER 3

COMPLEX QUERIES

3.1 VIEWS

3.1.1 View to Show Movie Awards

Code:

```
CREATE VIEW movie_awards_view AS
SELECT m.MovieName, a.AwardType, a.AwardName, a.AwardYear
FROM movies m
INNER JOIN awards a ON m.MovieID = a.MovieID;
SELECT * FROM movie_awards_view;
```

```
mysql> SELECT * FROM movie_awards_view;
+-------------------------+--------------------+----------------------+-----------+
| MovieName               | AwardType          | AwardName            | AwardYear |
+-------------------------+--------------------+----------------------+-----------+
| 3 Idiots                | National Film Award | Best Popular Film    |      2010 |
| Dangal                  | Filmfare Award     | Best Film            |      2017 |
| Sholay                  | Filmfare Award     | Best Film of 50 Years |      2005 |
| Zindagi Na Milegi Dobara | IIFA Award         | Best Film            |      2012 |
| Bajirao Mastani         | National Film Award | Best Director        |      2016 |
+-------------------------+--------------------+----------------------+-----------+
5 rows in set (0.02 sec)
```

Fig. 3.1 View Movie Awards

3.1.2 View for Movies with Rating > 8

Code:

```
CREATE VIEW movies_rating_above_8 AS
SELECT m.MovieID, m.MovieName, r.MotionPictures, r.TSeries, r.IMDB
FROM movies m
JOIN ratings r ON m.MovieID = r.MovieID
WHERE r.MotionPictures > 8 OR r.Tseries > 8 OR r.IMDB > 8;
```

SELECT * FROM movies_rating_above_8;

MovieID	MovieName	MotionPictures	TSeries	IMDB
1	3 Idiots	8.5	8.3	8.4
2	Dangal	8.2	8	8.3
3	Sholay	8	7.8	8.1
4	Zindagi Na Milegi Dobara	8.3	8.1	8.2

Fig. 3.2 View Movie Ratings

3.1.3 MoviesWithGenres: Shows movies along with their genres

Code:

```
CREATE VIEW MoviesWithGenres AS
SELECT m.MovieName, g.GenreName
FROM movies m
JOIN moviegenre mg ON m.MovieID = mg.MovieID
JOIN genre g ON mg.GenreID = g.GenreID;
SELECT * FROM MoviesWithGenres;
```

MovieName	GenreName
3 Idiots	Comedy
Bajirao Mastani	Drama
Zindagi Na Milegi Dobara	Drama
Dangal	Drama
3 Idiots	Drama
Dangal	Sports
Sholay	Action
Sholay	Adventure
Zindagi Na Milegi Dobara	Road Trip
Bajirao Mastani	Historical

Fig. 3.3 View Movie with Genre

3.1.4 AverageRatings: Displays movies along with their average ratings from the ratings table

Code:

```
CREATE VIEW AverageRatings AS
SELECT m.MovieName, (r.MotionPictures + r.TSeries + r.IMDB) / 3 AS AverageRating
FROM movies m
JOIN ratings r ON m.MovieID = r.MovieID;
SELECT * FROM AverageRatings;
```

```
mysql> SELECT * FROM AverageRatings;
+-------------------------+------------------+
| MovieName               | AverageRating    |
+-------------------------+------------------+
| 3 Idiots                | 8.399999936421713 |
| Dangal                  | 8.166666666666666 |
| Sholay                  |  7.96666685740153 |
| Zindagi Na Milegi Dobara | 8.200000127156576 |
| Bajirao Mastani         | 7.800000031789144 |
+-------------------------+------------------+
5 rows in set (0.04 sec)
```

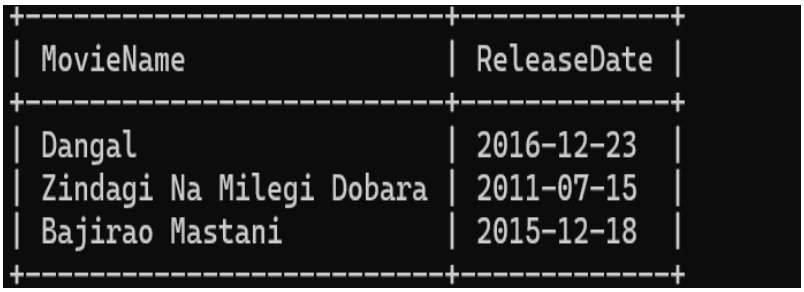
Fig. 3.4 View Average Ratings

3.1.5 MoviesReleasedAfter2010: Lists all movies released after 2010

Code:

```
CREATE VIEW MoviesReleasedAfter2010 AS
SELECT MovieName, ReleaseDate
FROM movies
WHERE ReleaseDate > '2010-01-01';
```

SELECT * FROM MoviesReleasedAfter2010;



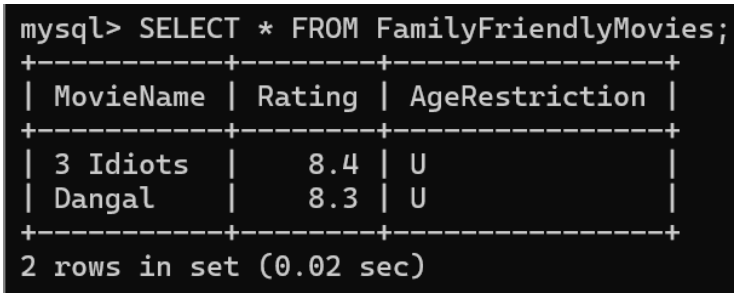
MovieName	ReleaseDate
Dangal	2016-12-23
Zindagi Na Milegi Dobara	2011-07-15
Bajirao Mastani	2015-12-18

Fig. 3.5 View Movie Released After 2010

3.1.6 FamilyFriendlyMovies : Displays movies with a rating above 7 and age restriction as 'U'.

Code:

```
CREATE VIEW FamilyFriendlyMovies AS
SELECT MovieName, Rating, AgeRestriction
FROM movies
WHERE Rating > 7 AND AgeRestriction = 'U';
SELECT * FROM FamilyFriendlyMovies;
```



```
mysql> SELECT * FROM FamilyFriendlyMovies;
+-----------+--------+----------------+
| MovieName | Rating | AgeRestriction |
+-----------+--------+----------------+
| 3 Idiots  |    8.4 | U              |
| Dangal    |    8.3 | U              |
+-----------+--------+----------------+
2 rows in set (0.02 sec)
```

Fig. 3.6 View Family Movies

Dropping the View:

If you no longer need a view, you can drop it using the DROP VIEW statement.

Code:

```
DROP VIEW movie_awards_view;
```

3.2 TRIGGERS :

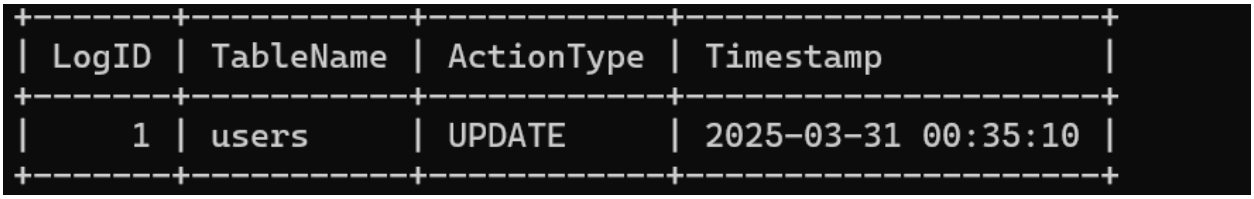
Create a log Table:

CREATE TABLE log (LogID INT AUTO_INCREMENT PRIMARY KEY, TableName VARCHAR(255), ActionType VARCHAR(255), Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

3.2.1 Trigger for Logging Updates:

Code:

```
DELIMITER //
CREATE TRIGGER log_user_update
AFTER UPDATE ON users
FOR EACH ROW
BEGIN
INSERT INTO log (TableName, ActionType)
VALUES ('users', 'UPDATE');
END; //
DELIMITER ;
INSERT INTO users (Name, DOB) VALUES ('John Doe', '1990-01-01');
UPDATE users SET Name = 'Johnathan Doe' WHERE UserID = 1;
SELECT * FROM log;
```



LogID	TableName	ActionType	Timestamp
1	users	UPDATE	2025-03-31 00:35:10

Fig. 3.7 Trigger Logging Update

3.2.2 Trigger for Auto-Creation of Subscription Record After User Registration (for users and subscription tables):

Code:

```
DELIMITER //
CREATE TRIGGER CreateDefaultSubscription
AFTER INSERT ON users
FOR EACH ROW
BEGIN
INSERT INTO subscription (UserID, PlanName, StartDate, EndDate, PlanPrice, PlanStatus)
VALUES (NEW.UserID, 'Basic Plan', CURDATE(), DATE_ADD (CURDATE(), INTERVAL 1 YEAR), 99.99, 'Active');
END; //
DELIMITER ;
INSERT INTO users (UserID, Name, DOB)
VALUES (7, 'Mark John', '1980-08-08');
select * from subscription;
```

```
mysql> select * from subscription;
+----------------+--------+------------+------------+------------+-----------+------------+--------------------------
---------+
| SubscriptionID | UserID | PlanName   | StartDate  | EndDate    | PlanPrice | PlanStatus | Details
        |
+----------------+--------+------------+------------+------------+-----------+------------+--------------------------
---------+
|              1 |      1 | Basic      | 2024-01-01 | 2024-06-30 |      9.99 | Active     | Basic plan with limited acce
ss      |
|              2 |      2 | Premium    | 2024-02-01 | 2024-07-31 |     14.99 | Active     | Premium plan with HD streami
ng      |
|              3 |      3 | Family     | 2024-03-01 | 2024-08-31 |     19.99 | Active     | Family plan with multiple de
vices   |
|              4 |      4 | Student    | 2024-04-01 | 2024-09-30 |      4.99 | Inactive   | Discounted plan for students
        |
|              5 |      5 | Annual     | 2024-05-01 | 2025-04-30 |     99.99 | Active     | Annual subscription with ful
l access |
|              6 |      7 | Basic Plan | 2025-04-01 | 2026-04-01 |     99.99 | Active     | NULL
        |
+----------------+--------+------------+------------+------------+-----------+------------+--------------------------
---------+
```

Fig. 3.8 Trigger Default Subscription

3.2.3 Trigger for Deleting Subscriptions When a User is Deleted (For users and subscription tables):

Code:

```
DELIMITER //
CREATE TRIGGER DeleteUserSubscriptions
BEFORE DELETE ON users
FOR EACH ROW
BEGIN
DELETE FROM subscription WHERE UserID = OLD.UserID;
END;//
DELIMITER ;
Delete from users where UserID = 7;
select * from subscription;
```

```
mysql> select
    -> * from subscription;
+----------------+--------+----------+------------+------------+-----------+-----------+-----------------------------------+
| SubscriptionID | UserID | PlanName | StartDate  | EndDate    | PlanPrice | PlanStatus | Details                           |
+----------------+--------+----------+------------+------------+-----------+-----------+-----------------------------------+
|              1 |      1 | Basic    | 2024-01-01 | 2024-06-30 |      9.99 | Active     | Basic plan with limited access    |
|              2 |      2 | Premium  | 2024-02-01 | 2024-07-31 |     14.99 | Active     | Premium plan with HD streaming    |
|              3 |      3 | Family   | 2024-03-01 | 2024-08-31 |     19.99 | Active     | Family plan with multiple devices |
|              4 |      4 | Student  | 2024-04-01 | 2024-09-30 |      4.99 | Inactive   | Discounted plan for students      |
|              5 |      5 | Annual   | 2024-05-01 | 2025-04-30 |     99.99 | Active     | Annual subscription with full access |
+----------------+--------+----------+------------+------------+-----------+-----------+-----------------------------------+
5 rows in set (0.00 sec)
```

Fig. 3.9 Trigger Delete Subscription

3.2.4 Prevent Duplicates for the Same Movie and Cast

Code:

```
DELIMITER //
CREATE TRIGGER PreventDuplicateCast
```

BEFORE INSERT ON movie_cast

FOR EACH ROW

BEGIN

IF EXISTS (SELECT 1 FROM movie_cast WHERE MovieID = NEW.MovieID AND CastID = NEW.CastID) THEN

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This cast is already associated with the movie';

END IF;

END;//

DELIMITER ;

INSERT INTO movie_cast (MovieID, CastID, Role) VALUES (1, 1, 'Rancho');

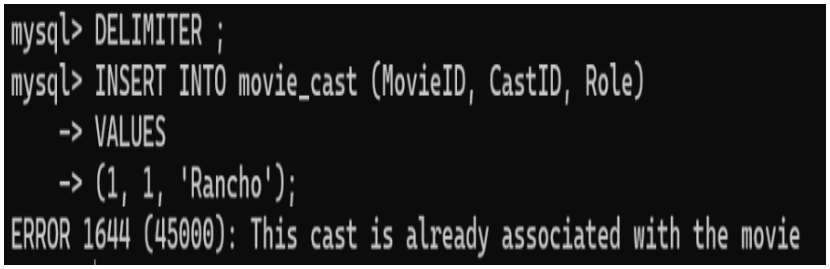


Fig. 3.10 Trigger Prevent Duplicate Cast

3.2.5 Trigger to Prevent Inserting Movies with a Negative Rating

Code:

DELIMITER //

CREATE TRIGGER PreventNegativeRating

BEFORE INSERT ON movies

FOR EACH ROW

BEGIN

IF NEW.Rating < 0 THEN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE_TEXT = ‘Rating cannot be negative’;

END IF;

END; //

DELIMITER ;

INSERT INTO movies (MovieName, Language, Duration, Rating, AgeRestriction, ReleaseDate)

VALUES (‘Test Movie’, ‘English’, 120, -1.0, ‘U’, ‘2023-01-01’);

```
mysql> DELIMITER ;
mysql> INSERT INTO movies (MovieName, Language, Duration, Rating, AgeRestriction, ReleaseDate)
    -> VALUES ('Test Movie', 'English', 120, -1.0, 'U', '2023-01-01');
ERROR 1644 (45000): Rating cannot be negative
```

Fig. 3.11 Trigger Prevent Negative Ratings

3.2.6 Enforce Minimum Duration for Movies

Code:

DELIMITER //

CREATE TRIGGER EnforceMinimumDuration

BEFORE INSERT ON movies

FOR EACH ROW

BEGIN

IF NEW.Duration < 30 THEN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE_TEXT = ‘Movie duration must be at least 30 minutes’;

END IF;

END;//

DELIMITER ;

INSERT INTO movies (MovieName, Language, Duration, Rating, AgeRestriction, ReleaseDate) VALUES (‘Test Movie’, ‘English’, 20,1.0, ‘U’, ‘2023-01-01’);

```
mysql> INSERT INTO movies (MovieName, Language, Duration, Rating, AgeRestriction, ReleaseDate) VALUES ('Test Movie', 'En
glish', 20,1.0, 'U', '2023-01-01');
ERROR 1644 (45000): Movie duration must be at least 30 minutes
```

Fig. 3.12 Trigger Minimum Duration

3.3 CURSORS

3.3.1 Cursor to retrieve all username from database

Code:

```
DELIMITER //
CREATE PROCEDURE fetch_users()
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE user_name VARCHAR(255);
DECLARE user_cursor CURSOR FOR
SELECT Name FROM users;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN user_cursor;
read_loop: LOOP
FETCH user_cursor INTO user_name;
IF done THEN
LEAVE read_loop;
END IF;
SELECT user_name;
END LOOP;
```

```
CLOSE user_cursor;
END //
DELIMITER ;
CALL fetch_users();
```

```
mysql> CALL fetch_users();
+---------------+
| user_name     |
+---------------+
| Johnathan Doe |
+---------------+
1 row in set (0.02 sec)

+-------------+
| user_name   |
+-------------+
| Priya Verma |
+-------------+
1 row in set (0.02 sec)

+-------------+
| user_name   |
+-------------+
| Rahul Mehta |
+-------------+
1 row in set (0.03 sec)

+--------------+
| user_name    |
+--------------+
| Sneha Kapoor |
+--------------+
1 row in set (0.03 sec)

+--------------+
| user_name    |
+--------------+
| Vikram Joshi |
+--------------+
1 row in set (0.03 sec)
```

Fig. 3.13 Cursor Fetch Users

3.3.2 Cursor for Iterating Over movie names

Code:

```
DELIMITER //
CREATE PROCEDURE GetMovieNames()
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE movie_name VARCHAR(255);
DECLARE movie_cursor CURSOR FOR SELECT MovieName FROM movies;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN movie_cursor;
read_loop: LOOP
FETCH movie_cursor INTO movie_name;
IF done THEN
LEAVE read_loop;
END IF;
SELECT movie_name;
END LOOP;
CLOSE movie_cursor;
END; //
DELIMITER ;
CALL GetMovieNames()
```

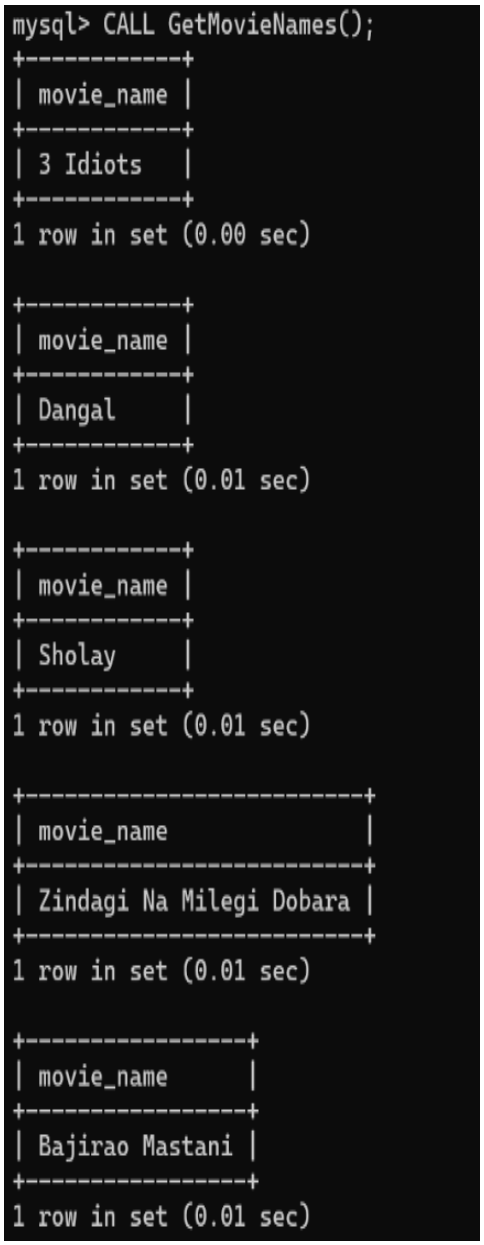


Fig. 3.14 Cursor Get Movies Name

3.3.3 Cursor to retrieve all username from database

Code:

```
DELIMITER //
CREATE PROCEDURE CalculateAverageRating()
BEGIN
```

```
DECLARE done INT DEFAULT 0;
DECLARE movie_id INT;
DECLARE movie_rating FLOAT;
DECLARE total_rating FLOAT DEFAULT 0;
DECLARE movie_count INT DEFAULT 0;
DECLARE avg_rating FLOAT;
DECLARE movie_cursor CURSOR FOR SELECT MovieID, Rating FROM
movies;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN movie_cursor;
read_loop: LOOP
FETCH movie_cursor INTO movie_id, movie_rating;
IF done THEN
LEAVE read_loop;
END IF;
SET total_rating = total_rating + movie_rating;
SET movie_count = movie_count + 1;
END LOOP;
CLOSE movie_cursor;
IF movie_count > 0 THEN
SET avg_rating = total_rating / movie_count;
SELECT avg_rating AS AverageRating;
ELSE
SELECT ‘No movies available’ AS Message;
END IF;
END; //
DELIMITER ;
```

CALL CalculateAverageRating();

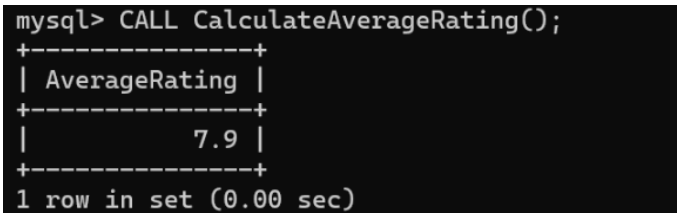


Fig. 3.15 Cursor Calculate Average Rating

3.3.4 Cursor to list cast members for a movie

Code:

```
DELIMITER //
CREATE PROCEDURE GetMovieCast(IN movie_id INT)
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE cast_id INT;
DECLARE role VARCHAR(255);
DECLARE cast_name VARCHAR(255);
DECLARE cast_cursor CURSOR FOR
SELECT c.Name, mc.Role
FROM cast c
JOIN movie_cast mc ON c.CastID = mc.CastID
WHERE mc.MovieID = movie_id;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN cast_cursor;
read_loop: LOOP
FETCH cast_cursor INTO cast_name, role;
IF done THEN
LEAVE read_loop;
```

```
END IF;
SELECT cast_name, role;
END LOOP;
CLOSE cast_cursor;
END; //
DELIMITER ;
CALL GetMovieCast(1);
CALL GetMovieCast(4);
```

```
mysql> CALL GetMovieCast(1);
+------------+--------+
| cast_name  | role   |
+------------+--------+
| Aamir Khan | Rancho |
+------------+--------+
1 row in set (0.01 sec)

+-------------+--------+
| cast_name   | role   |
+-------------+--------+
| R. Madhavan | Farhan |
+-------------+--------+
1 row in set (0.02 sec)

+---------------+------+
| cast_name     | role |
+---------------+------+
| Sharman Joshi | Raju |
+---------------+------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

mysql> CALL GetMovieCast(4);
+----------------+-------+
| cast_name      | role  |
+----------------+-------+
| Hrithik Roshan | Arjun |
+----------------+-------+
1 row in set (0.00 sec)

+---------------+-------+
| cast_name     | role  |
+---------------+-------+
| Farhan Akhtar | Imran |
+---------------+-------+
1 row in set (0.00 sec)
```

Fig. 3.16 Cursor Get Movie Cast

3.3.5 Cursor to count movies by genre

Code:

```
DELIMITER //
CREATE PROCEDURE CountMoviesByGenre(IN genre_id INT)
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE movie_count INT;
DECLARE genre_cursor CURSOR FOR
SELECT COUNT(*) FROM moviegenre WHERE GenreID = genre_id;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN genre_cursor;
read_loop: LOOP
FETCH genre_cursor INTO movie_count;
IF done THEN
LEAVE read_loop;
END IF;
SELECT movie_count AS MoviesInGenre;
END LOOP;
CLOSE genre_cursor;
END; //
DELIMITER ;
CALL CountMoviesByGenre(2);
```

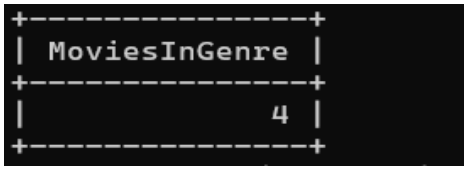


Fig. 3.17 Cursor Count Movie By Genre

3.3.6 Cursor to find movie released before a certain date

Code:

```
DELIMITER //
CREATE PROCEDURE GetMoviesBeforeDate(IN release_date DATE)
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE movie_name VARCHAR(255);
DECLARE movie_release_date DATE;
DECLARE movie_cursor CURSOR FOR
SELECT MovieName, ReleaseDate FROM movies WHERE ReleaseDate <
release_date;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN movie_cursor;
read_loop: LOOP
FETCH movie_cursor INTO movie_name, movie_release_date;
IF done THEN
LEAVE read_loop;
END IF;
SELECT movie_name, movie_release_date;
END LOOP;
CLOSE movie_cursor;
END; //
DELIMITER ;
CALL GetMoviesBeforeDate('2010-01-01');
```

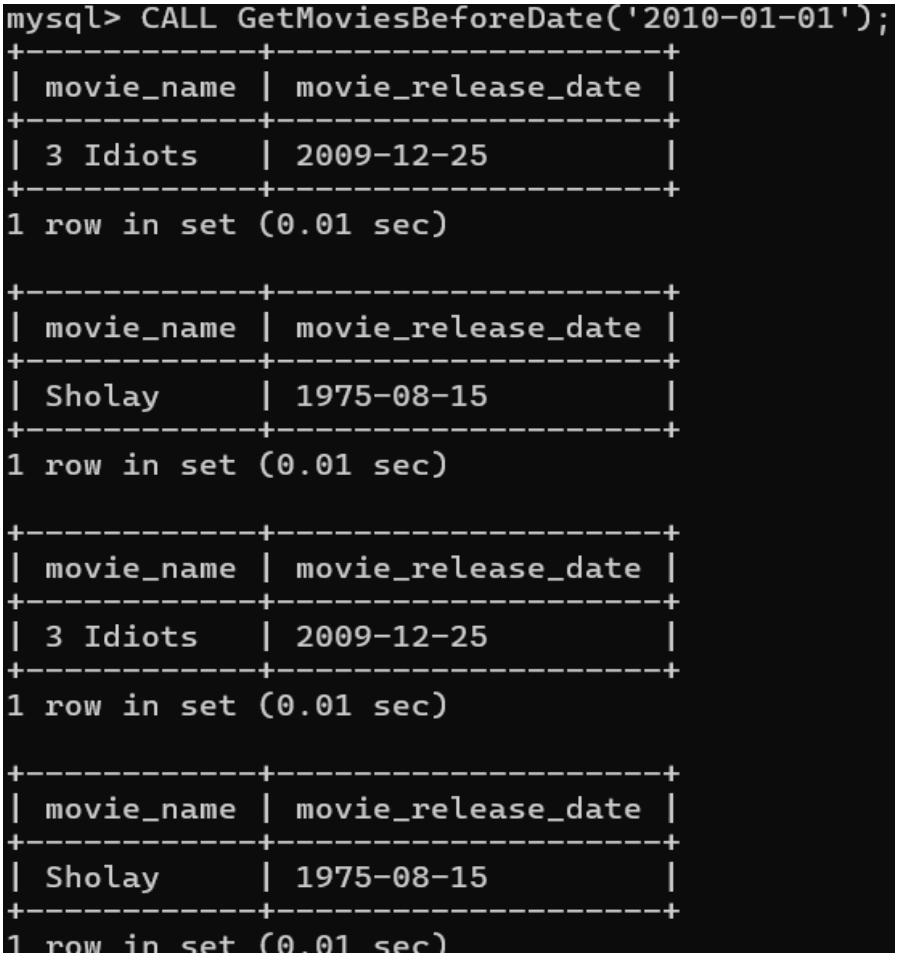


Fig. 3.18 Cursor Get Movies Before Date

CHAPTER 4

PITFALLS, DEPENDENCIES AND NORMALIZATION

4.1 ANALYZING THE PITFALLS

Normalization is a systematic approach to organizing data in a database to reduce redundancy and improve data integrity. In the context of the movie recommendation system, we can analyze the pitfalls, identify dependencies, and apply normalization to ensure that the database design is efficient and effective.

Pitfalls in the Current Design

1. **Data Redundancy:** Repeated data in multiple places can lead to unnecessary storage usage and inconsistencies. For example, storing movie details like the genre or director in every cast or review record.
2. **Update Anomalies:** When the data is duplicated in several tables, updating it in one place without updating others can lead to inconsistencies.
3. **Insertion Anomalies:** Certain records may not be insertable due to missing dependent data.
4. **Deletion Anomalies:** Deleting a record might unintentionally remove important related data.

4.2 IDENTIFYING DEPENDENCIES

The functional dependencies in Movie Recommendation schema:

- **Movie:** MovieID → MovieName, Language, Duration
- **User:** UserID → Name, DOB, Password, Username
- **Device:** DeviceID → OS, Type, UserID

- **MovieCast:** MovieID → Role, CastID
- **Genre:** GenreID → GenreName
- **Awards:** AwardID → MovieID, AwardType, AwardName, AwardYear
- **Cast:** CastID → Name, Bio, DOB
- **Ratings:** RatingID → MovieID, MotionPictures, TSeries, IMDB
- **Directors:** DirectorID → Name, Bio, DOB
- **Region:** RegionID → RegionName
- **Reviews:** ReviewID → UserID(FK), MovieID, ReviewText, Rating, ReviewDate
- **Subscription:** SubscriptionID → UserID, PlanName, StartDate, EndDate, PlanPrice, PlanStatus, Details
- **Watchlist:** WatchlistID → UserID, MovieID, AddedDate
- **MovieTrailer:** TrailerID → MovieID, TrailerLink, UploadDate

4.3 APPLYING NORMALIZATION

Normalization typically involves several normal forms (NF). Here, we will apply the first three normal forms (1NF, 2NF, and 3NF).

First Normal Form (1NF)

1NF ensures that each column contains atomic values, and each record is unique. There are no repeating groups or multi-valued attributes.

- Each table already has a primary key (e.g., MovieID, UserID).
- No repeating groups or arrays are present.

Second Normal Form (2NF)

2NF ensures that all non-key attributes are fully dependent on the primary key and removes partial dependencies.

Example: MovieGenre now has a composite key of MovieID and GenreID, separating the genre data from the movie data.

Revised Tables for 2NF:

1. **Movie:** MovieID, MovieName, Language, Duration
2. **Device:** DeviceID, OS, Type, UserID
3. **MovieCast:** MovieID, CastID, Role
4. **Awards:** AwardID, MovieID, AwardType, AwardName, AwardYear
5. **MovieGenre:** MovieID, GenreID
6. **Ratings:** RatingID, MovieID, MotionPictures, TSeries, IMDB
7. **Watchlist:** WatchlistID, UserID, MovieID, AddedDate
8. **Subscription:** SubscriptionID, UserID, PlanName, StartDate, EndDate, PlanPrice, PlanStatus, Details

Third Normal Form (3NF)

3NF ensures that no transitive dependencies exist, meaning non-key attributes cannot depend on other non-key attributes.

In the Reviews table, attributes such as ReviewText, Rating, and ReviewDate depend directly on the composite key of MovieID and UserID, which are part of the primary key.

Boyce-Codd Normal Form (BCNF)

Stronger version of 3NF. For every functional dependency X → Y, X must be a super key.

Example: If in the Device table, UserID → DeviceID, and DeviceID → UserID, then both are candidate keys. The table is in BCNF if all determinants are candidate keys.

All tables in the final normalized schema above satisfy BCNF as each determinant is either a primary key or a candidate key.

Fourth Normal Form (4NF)

4NF removes multi-valued dependencies. A table should not have more than one multi-valued independent attribute.

Example: If a Movie has multiple Genres and multiple Trailers, storing both in one table introduces multi-valued dependency.

Keep separate tables:

MovieGenre(MovieID, GenreID)

MovieTrailer(MovieID, TrailerID, TrailerLink, UploadDate)

This ensures each table holds only one independent multi-valued attribute.

Fifth Normal Form (5NF) / Project-Join Normal Form (PJNF)

Decomposed relations should be reconstructed without data loss. Deals with join dependencies.

Example: If a movie is available in multiple regions, languages, and formats:

- Region: RegionID → RegionName

- Language: LanguageID → LanguageName
- Format: FormatID → FormatType

Instead of one large table like MovieID, RegionID, LanguageID, FormatID, split into:

- MovieRegion(MovieID, RegionID)
- MovieLanguage(MovieID, LanguageID)
- MovieFormat(MovieID, FormatID)

These tables can be joined without loss of information, satisfying 5NF.

4.4 Final Normalized Schema:

After applying normalization, the final schema might look like this:

1. **Movie:**
   - MovieID (PK)
   - MovieName
   - Language
   - Duration
2. **User:**
   - UserID (PK)
   - Name
   - DOB
   - Password
   - Username

3. **Device:**

- DeviceID (PK)
- OS
- Type
- UserID (FK)

4. **MovieCast:**

- MovieID (PK, FK)
- CastID (PK, FK)
- Role

5. **Genre:**

- GenreID (PK)
- GenreName

6. **MovieGenre:**

- MovieID (PK, FK)
- GenreID (PK, FK)

7. **Awards:**

- AwardID (PK)
- MovieID (FK)
- AwardType
- AwardName
- AwardYear

8. **Cast:**

- CastID (PK)
- Name
- Bio
- DOB

9. **Ratings:**

- RatingID (PK)
- MovieID (FK)
- MotionPictures
- TSeries
- IMDB

10. **Directors:**

- DirectorID (PK)
- Name
- Bio
- DOB

11. **Region:**

- RegionID (PK)
- RegionName

12. **Reviews:**

- ReviewID (PK)
- UserID (FK)
- MovieID (FK)

- ReviewText
- Rating
- ReviewDate

13. Subscription:

- SubscriptionID (PK)
- UserID (FK)
- PlanName
- StartDate
- EndDate
- PlanPrice
- PlanStatus
- Details

14. Watchlist:

- WatchlistID (PK)
- UserID (FK)
- MovieID (FK)
- AddedDate

15. MovieTrailer:

- TrailerID (PK)
- MovieID (FK)
- TrailerLink
- UploadDate

4.5 Benefits of Normalization

1. **Reduced Redundancy:** Eliminate redundant data, saving storage and ensuring consistency.
2. **Improved Data Integrity:** Maintain data consistency and minimize anomalies.
3. **Easier Maintenance:** Changes to one part of the system will not affect other parts, simplifying maintenance.
4. **Enhanced Query Performance:** Optimized structure for querying efficiency, reducing unnecessary joins and making data retrieval faster.

By normalizing the database schema, we have minimized redundancy, improved data integrity, and optimized the structure for efficient queries.

CHAPTER 5

CONCURRENCY CONTROL

5.1 INTRODUCTION

The mechanisms implemented to ensure both concurrent transactions in the Movie Recommendation System and recovery mechanisms in case of system failure. The goal is to ensure that the system functions smoothly with multiple users and can recover to a consistent state if failures occur.

5.2 CONCURRENCY CONTROL

Concurrency control is a crucial aspect of any system where multiple users can interact with the database simultaneously. In the case of the Movie Recommendation System, several techniques have been implemented to ensure that transactions execute in a way that maintains data consistency and prevents conflicts.

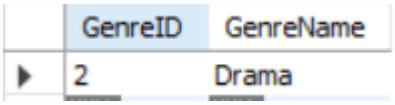
5.2.1 LOCKING MECHANISM

Locking mechanisms are used to prevent conflicts during simultaneous access to shared resources.

- **Shared/Exclusive Lock:** A shared lock allows multiple transactions to read the same data, but only one transaction can modify the data using an exclusive lock. This ensures that while one transaction reads, others can also read, but no other transaction can modify the data concurrently.

  Example Queries:

  SELECT * FROM genre WHERE GenreID = 5 LOCK IN SHARE MODE;



GenreID	GenreName
2	Drama

Fig. 5.1 Shared and Executive Lock

- **Row-Level Locking:** When a specific row of data is locked instead of the entire table, it reduces the contention and allows more concurrent transactions to proceed.

  Example Query:

  START TRANSACTION;

  UPDATE movies SET Rating = 4.5 WHERE MovieID = 15;

  COMMIT;

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE movies SET Rating = 4.5 WHERE MovieID = 15;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Fig. 5.2 Row-Level Lock

  This prevents any other transaction from modifying the same row while it's being updated.

- **Deadlock Detection and Resolution:** A **deadlock** occurs when two or more transactions are waiting for each other to release locks on resources, and thus they are stuck indefinitely.

  Example Query:

  START TRANSACTION;

  UPDATE movies SET Rating = 4.5 WHERE MovieID = 10;

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE movies SET Rating = 4.5 WHERE MovieID = 10;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Fig. 5.3 Deadlock Detection

  If the system detects that both transactions are waiting for each other, one will be rolled back to resolve the deadlock.

5.2.2 ISOLATION LEVELS

Isolation levels define the degree to which the changes made by one transaction are visible to other transactions. They help in preventing issues such as dirty reads, non-repeatable reads, and phantom reads.

- **Read Committed:** At this isolation level, transactions can only read committed data. This prevents dirty reads (reading data that hasn't been committed yet).
- **Repeatable Read:** This isolation level ensures that if a transaction reads a row, no other transaction can modify it before the current transaction completes. This avoids non-repeatable reads.
- **Serializable:** The highest isolation level ensures that transactions are executed one after another, as if they were executed in serial order. This eliminates any possibility of dirty reads, non-repeatable reads, or phantom reads.

5.3 RECOVERY MECHANISMS

In case of system failure (such as a crash, power failure, or network issue), **recovery mechanisms** ensure that the database can be restored to a consistent state. These mechanisms include transaction logging, backup strategies, and replication.

5.3.1 Transaction Logging

Transaction logs are essential for **recovery** as they record all changes made to the database before they are written to disk.

- **Write-Ahead Logging:** WAL ensures that all changes are logged before the data is modified. In case of a crash, the system can replay the logs to restore data to a consistent state.

- **Checkpointing:** Checkpointing involves periodically flushing all in-memory changes to disk. This minimizes the amount of log data to replay during recovery.

5.3.2 Backup and Recovery

Backup and recovery mechanisms ensure data can be restored to a previous state if necessary.

- **Full Backups:** A full backup captures the entire database, ensuring that no data is lost if a failure occurs.

  Backup Command:

  mysqldump -u root -p movie_db > full_backup.sql
- **Incremental Backups:** Incremental backups only capture the changes made since the last full or incremental backup. This reduces the backup time and storage needed.

  Backup Command:

  mysqldump -u root -p --incremental movie_db > incremental_backup.sql
- **Point-in-Time Recovery:** Using transaction logs and backups, the database can be restored to a specific point in time, providing more flexibility in recovery.

  Recovery Command:

  mysqlbinlog --start-datetime="2025-04-23 10:00:00" --stop-datetime="2025-04-23 11:00:00" mysql-bin.000001 | mysql -u root -p

5.3.3 REPLICATION AND FAILOVER

Replication ensures that multiple copies of the database are available, minimizing downtime in case of failure.

- **Data Replication:** Replication creates a real-time copy of the database on another server.
- **Load Balancing:** Load balancing can be used to distribute requests between multiple servers, improving performance and availability.

5.3.4 ERROR HANDLING AND ROLLBACK

In case of errors during transaction execution, error handling mechanisms ensure that the database maintains consistency.

- **Exception Handling:** Implement robust exception handling mechanisms to catch and handle errors or exceptional conditions that may occur during transactions.
- **Rollback Mechanism:** Rollback ensures that in case of errors or transaction failure, any changes made are undone, maintaining data consistency.

5.4 CONCLUSION

By implementing these concurrency control and recovery mechanisms, the Movie Recommendation System is equipped to handle multiple users accessing and modifying the database simultaneously while ensuring data consistency and providing recovery solutions in case of system failures.

CHAPTER 6

CODE

6.1 FRONTEND

Cińemà

<h2>Welcome, {{ name }}!</h2>

<h3> 🎬 Movies For You</h3>

<h4>{{ movie[1] }}</h4>

Genre: {{ movie[2] }}

Duration: {{ movie[3] }} mins

Avg MotionPictures: ⭐ {{ movie[4]|round(1) }}

Avg TSeries: ⭐ {{ movie[5]|round(1) }}

Avg IMDB: ⭐ {{ movie[6]|round(1) }}

No ratings yet.

<img src="https://img.youtube.com/vi/{{ movie[7].split('v=')[1] }}/0.jpg" alt="YouTube Thumbnail">

🎬 Watch Trailer

<h3>No recommendations available yet.</h3>

Please select a genre to receive personalized movie recommendations.

🚪 Logout

CHAPTER 7

RESULT AND DISCUSSION

7.1 USER LOGIN PAGE

The login page serves as the entry point to the webpage, ensuring secure access and user authentication.

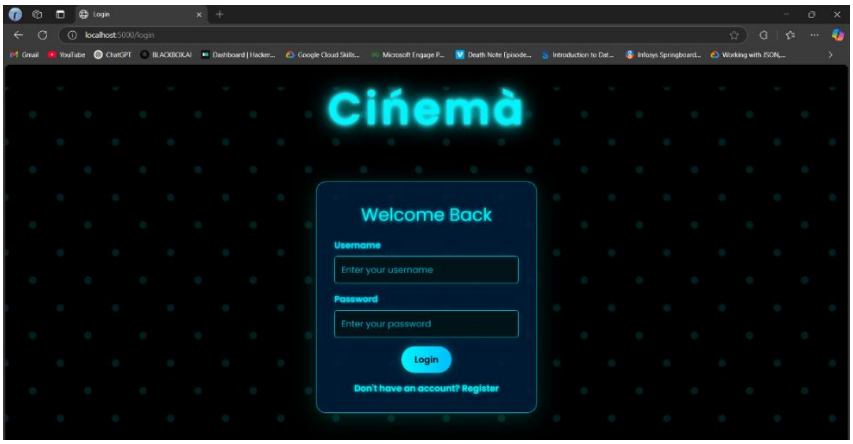


Fig. 7.1 Login Page

Theme: The dark background with glowing blue text and dotted patterns gives a futuristic, cinema-style vibe.

Typography & Effects: The title "Cinéma" uses a glowing neon effect, creating a bold and eye-catching heading. Text elements like "Welcome Back" and button labels maintain a consistent glow-style aesthetic.

Login Form:

- **Username Field** – Allows users to input their username.
- **Password Field** – For secure entry of the user's password.
- **Login Button** – A clickable button to authenticate and log into the application

Navigation Link: Below the form, a prompt says: *"Don't have an account? Register"* – likely redirecting to a registration page for new users.

7.2 USER REGISTER PAGE

The Registration Page of the *Cinéma* web application

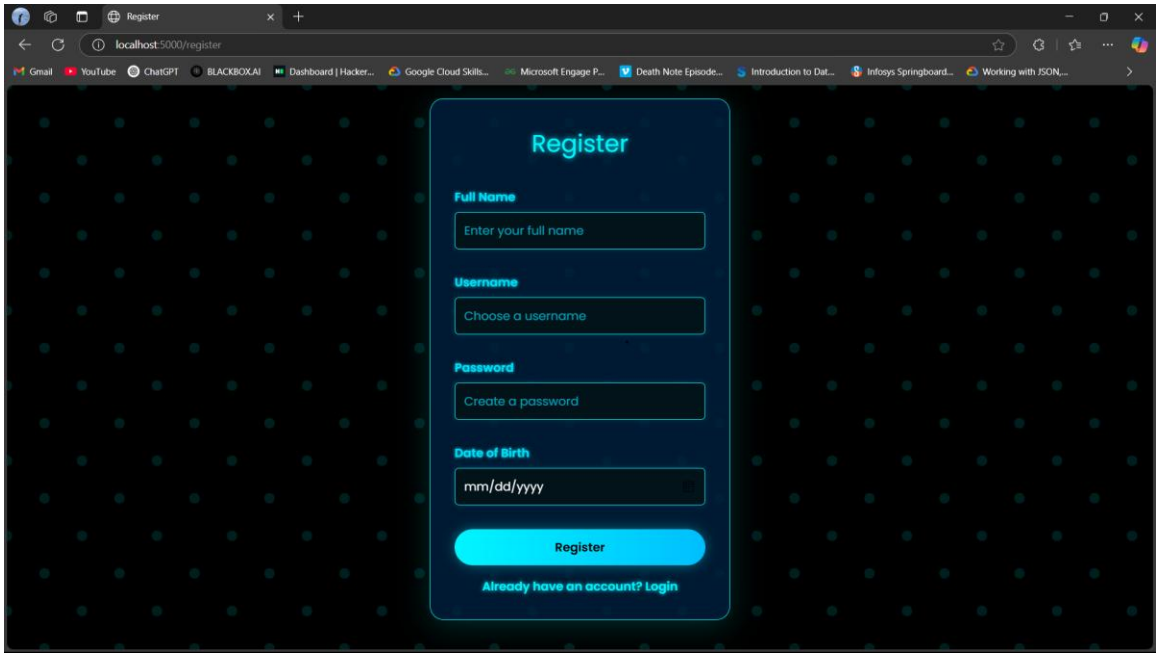


Fig. 7.2 Registration Page

Theme: The dark background with glowing blue text and dotted patterns gives a futuristic, cinema-style vibe

Form Field:

- **Full Name –** Accepts the user's complete name.
- **Username –** Lets the user choose a unique identifier for login.
- **Password –** Secured field to protect user access.
- **Date of Birth –** Standard date input field in *mm/dd/yyyy* format.
- **Register –** Triggers the submission of the registration form, likely sending user data to the backend for account creation.

Navigation Link: Below the form, the prompt says *"Already have an account? Login"* — links users to the login page if they are already registered.

7.1.3 HOME PAGE

The main landing page of the *Cinéma* web app that displays personalized movie recommendations for the logged-in user.

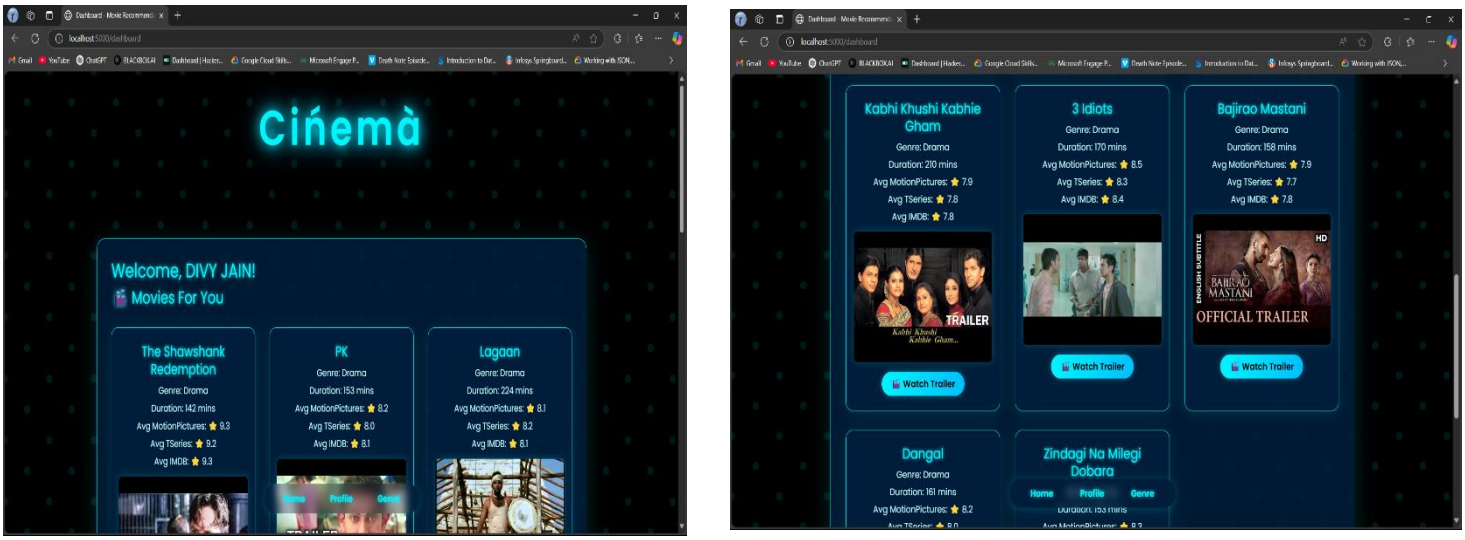


Fig. 7.3 Home Page

Interest-Based Recommendations: Movies displayed are tailored to the user's preferences

Comprehensive Movie Info: The *Cinéma* landing page provides personalized movie recommendations based on user interests. Each movie card displays the title, genre, duration, and average ratings from MotionPictures, TSeries, and IMDB

Watch Trailers: Users can instantly preview any movie by clicking the **Watch Trailer** button linked to the official trailer.

UI Highlights:

- A sleek, **neon-glow aesthetic** consistent with a cinema theme.
- Organized **card-style layout** for easy viewing and comparison.
- Smooth navigation with floating bottom tabs like **Home**, **Profile**, and **Genre**

7.1.4 PROFILE PAGE

The profile page displays the user's personal details including their full name, username, date of birth, and preferred movie genre. It also shows their current subscription status.

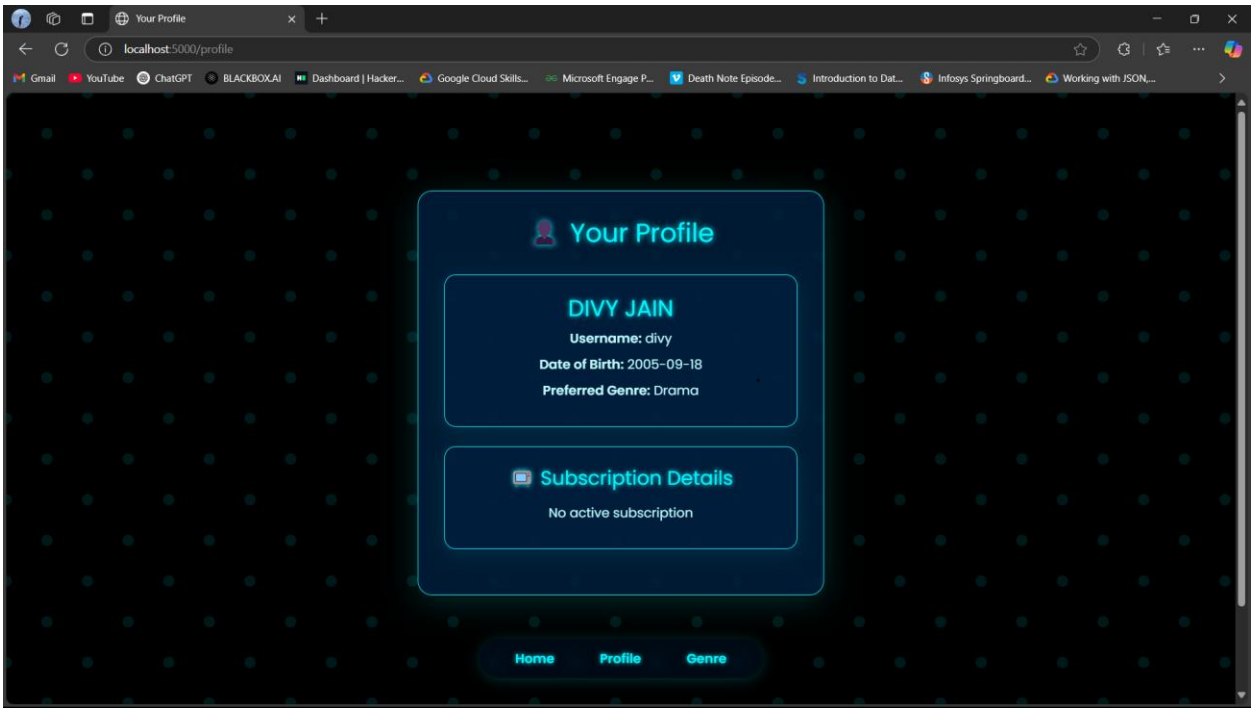


Fig. 7.4 Profile Page

Theme: The layout is clean with a neon theme that matches the overall design of the platform, ensuring a visually consistent and user-friendly experience.

Profile Information Card: The Profile Information Card displays key user details in a concise format. It shows the full name of the user, along with their chosen login username. The registered date of birth is also provided, helping personalize the user experience. Additionally, the card highlights the user's preferred genre, which is likely used to tailor movie recommendations based on individual interests.

- **Subscription Details:** Indicates the user's subscription status.

7.1.5 GENRE Selection PAGE

The page is a genre selection interface, letting users choose their favorite genre.

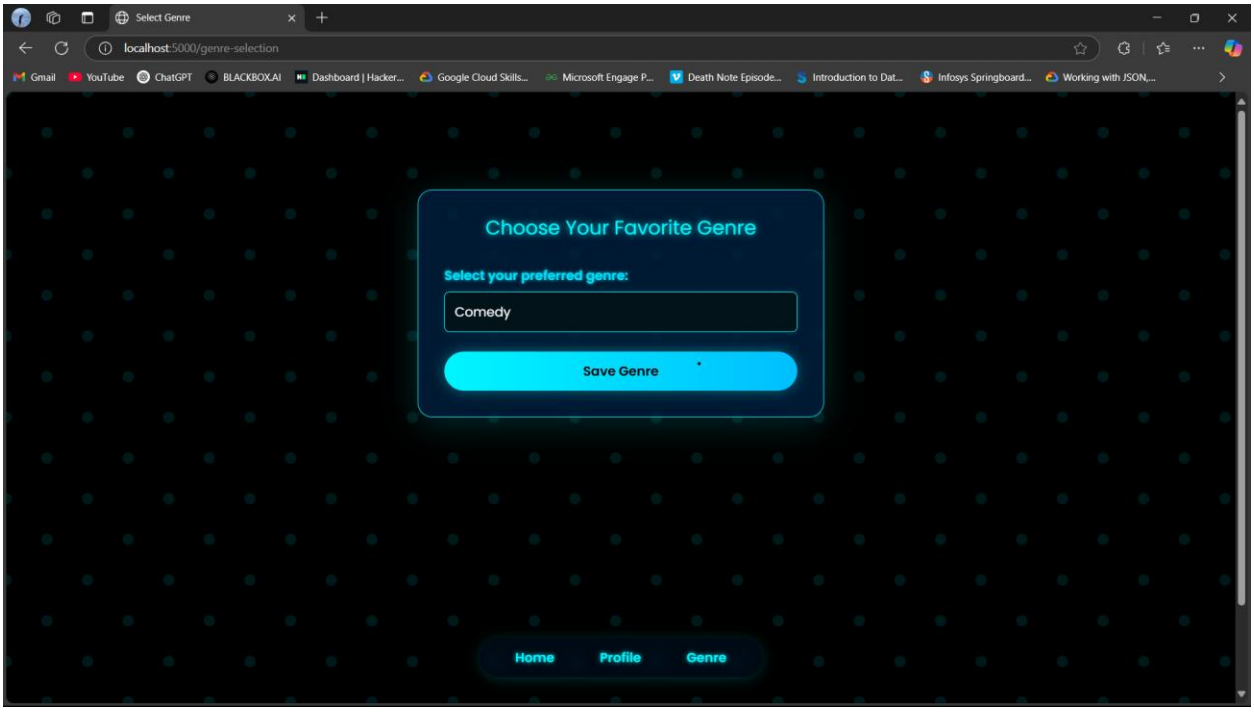


Fig. 7.5 Genre Selection Page

Theme: Neon/cyberpunk vibes glowing blue on a dark background with faint dots gives it a futuristic aesthetic.

Typography & UI: Clear and glowing header *Choose Your Favorite Genre*. The *Save Genre* button also glows and has a rounded, modern design.

Navigation Buttons: Located at the bottom with options for Home, Profile, and Genre. All buttons match the glowing theme.

CHAPTER 8

CONCLUSION

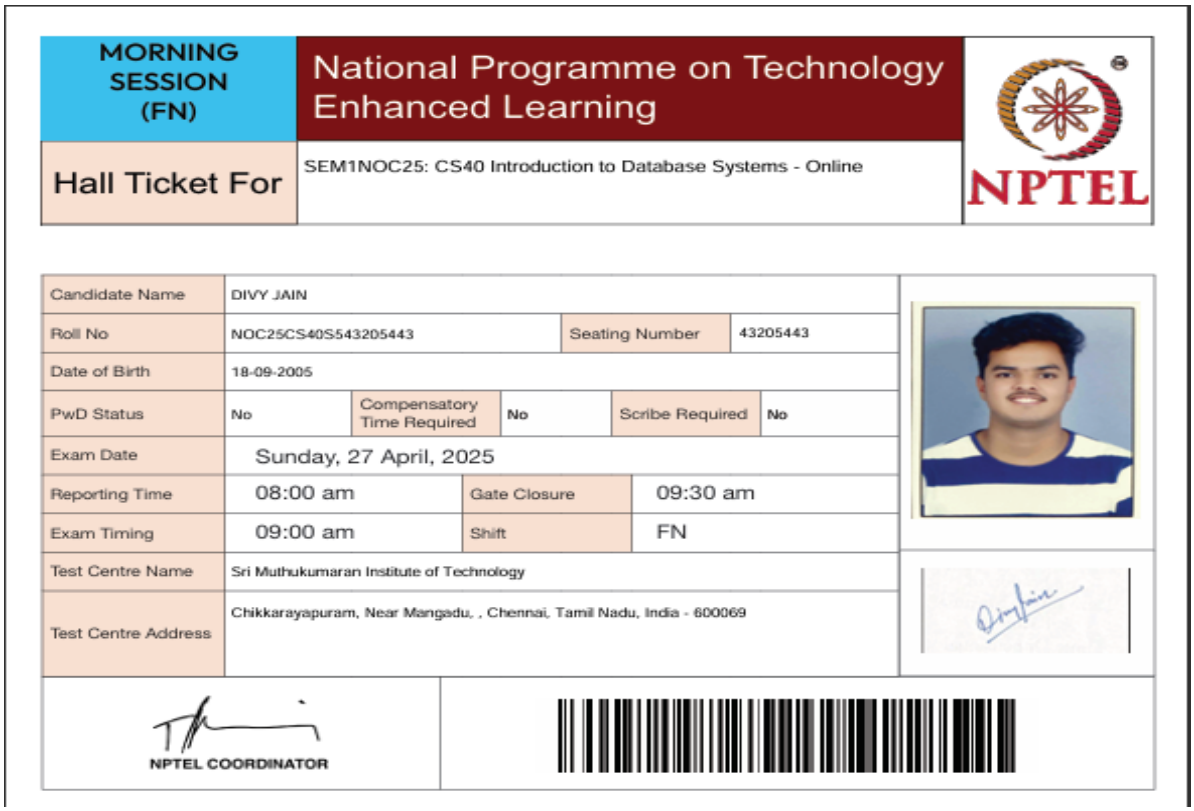
The Movie Recommendation System project successfully demonstrates the design and implementation of a comprehensive and user-centric platform tailored for personalized entertainment experiences. By integrating various features such as collaborative filtering, content-based filtering, and hybrid recommendation methods, the system intelligently suggests movies aligned with users' interests and viewing history. Through the strategic use of entities and attributes—such as Movies, Users, Genres, Watchlists, and Reviews—the system ensures organized and scalable data management, allowing seamless handling of user preferences, ratings, subscriptions, and media content.

Furthermore, the project emphasizes real-time interaction with dynamic data using advanced SQL features like views, triggers, and cursors. These enable enhanced data processing, monitoring, and automation of tasks such as logging changes, managing subscriptions, and enforcing constraints. The frontend complements the backend with an intuitive and visually engaging interface, creating an immersive experience for users. Overall, this system serves as a powerful solution for navigating large movie libraries, fostering higher user satisfaction and engagement through intelligent recommendations and smooth usability.

ONLINE COURSE CERTIFICATION

DIVY JAIN

- **NPTEL HALL TICKET:**

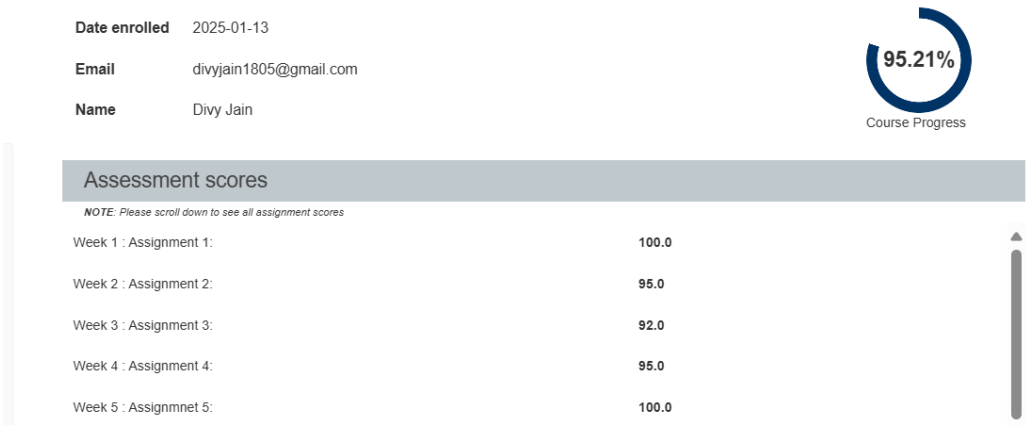


MORNING SESSION (FN)	National Programme on Technology Enhanced Learning	NPTEL
Hall Ticket For	SEM1NOC25: CS40 Introduction to Database Systems - Online	

Candidate Name	DIVY JAIN				
Roll No	NOC25CS40S543205443		Seating Number	43205443	
Date of Birth	18-09-2005				
PwD Status	No	Compensatory Time Required	No	Scribe Required	No
Exam Date	Sunday, 27 April, 2025				
Reporting Time	08:00 am		Gate Closure	09:30 am	
Exam Timing	09:00 am		Shift	FN	
Test Centre Name	Sri Muthukumaran Institute of Technology				
Test Centre Address	Chikkarayapuram, Near Mangadu, , Chennai, Tamil Nadu, India - 600069				

NPTEL COORDINATOR

- **COURSE PROGRESS:**



Date enrolled 2025-01-13

Email divyjain1805@gmail.com

Name Divy Jain

95.21%
Course Progress

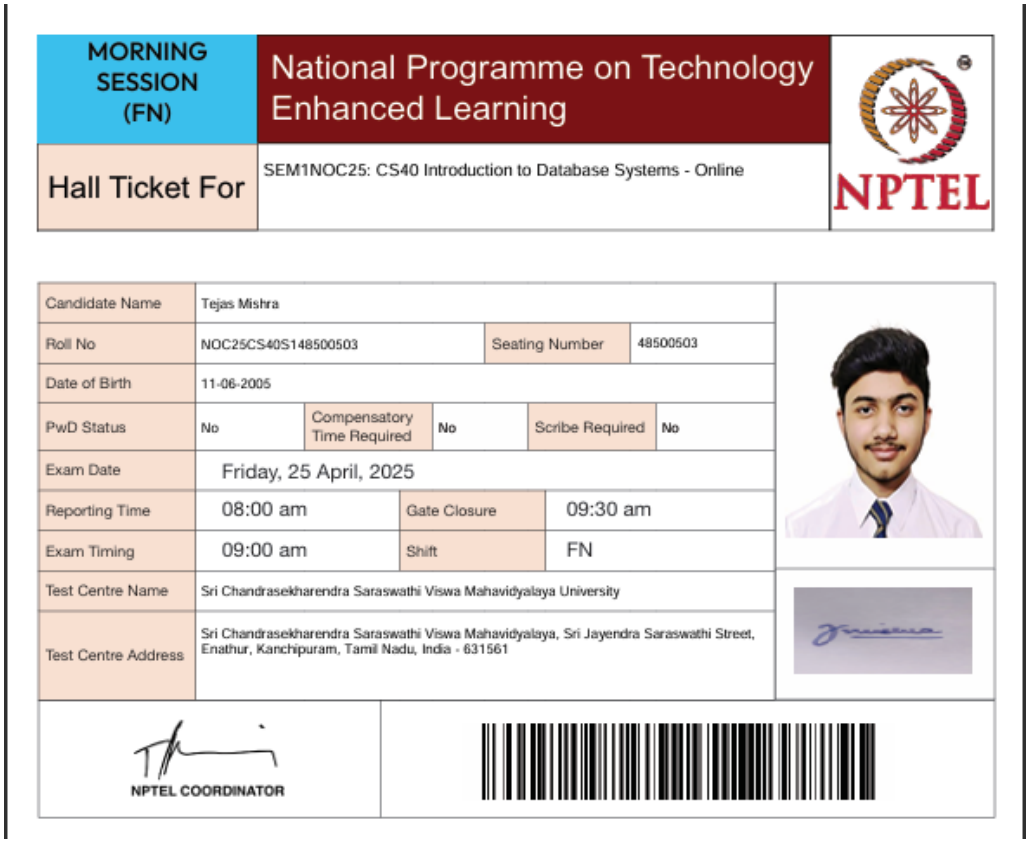
Assessment scores

NOTE: Please scroll down to see all assignment scores

Week 1 : Assignment 1:	100.0
Week 2 : Assignment 2:	95.0
Week 3 : Assignment 3:	92.0
Week 4 : Assignment 4:	95.0
Week 5 : Assignmnet 5:	100.0

TEJAS MISHRA

- **NPTEL HALL TICKET:**

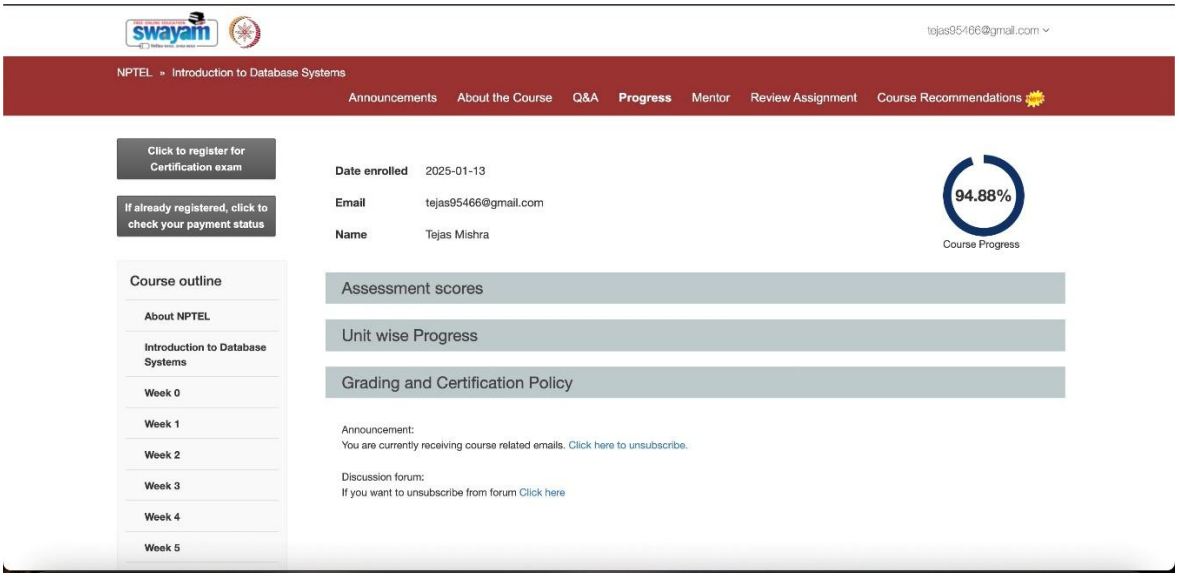


MORNING SESSION (FN)	National Programme on Technology Enhanced Learning	NPTEL
Hall Ticket For	SEM1NOC25: CS40 Introduction to Database Systems - Online	

Candidate Name	Tejas Mishra				
Roll No	NOC25CS40S148500503		Seating Number	48500503	
Date of Birth	11-06-2005				
PwD Status	No	Compensatory Time Required	No	Scribe Required	No
Exam Date	Friday, 25 April, 2025				
Reporting Time	08:00 am		Gate Closure	09:30 am	
Exam Timing	09:00 am		Shift	FN	
Test Centre Name	Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya University				
Test Centre Address	Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya, Sri Jayendra Saraswathi Street, Enathur, Kanchipuram, Tamil Nadu, India - 631561				

NPTEL COORDINATOR

- **COURSE PROGRESS**



CINEMA- THE MOVIE RECOMMENDATION SYSTEM

CONTRIBUTORS:

- DIVY JAIN- divyjain1805@gmail.com
- TEJAS MISHRA- tejas95466@gmail.com

GitHub link: *Divy1809/Cinema*