Movie Rating and Commenting System - Database Design

1. Database Schema

Tables Structure

Users Table

```
CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Movies Table

```
CREATE TABLE Movies (
    movie_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    release_date DATE,
    director VARCHAR(100),
    duration_minutes INT,
    poster_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Genres Table

```
CREATE TABLE Genres (
    genre_id INT PRIMARY KEY AUTO_INCREMENT,
    genre_name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Movie_Genres Table (Many-to-Many Relationship)

```
CREATE TABLE Movie_Genres (
    movie_id INT,
    genre_id INT,
    PRIMARY KEY (movie_id, genre_id),
    FOREIGN KEY (movie_id) REFERENCES Movies(movie_id) ON DELETE CASCADE,
    FOREIGN KEY (genre_id) REFERENCES Genres(genre_id) ON DELETE CASCADE
);
```

Ratings Table

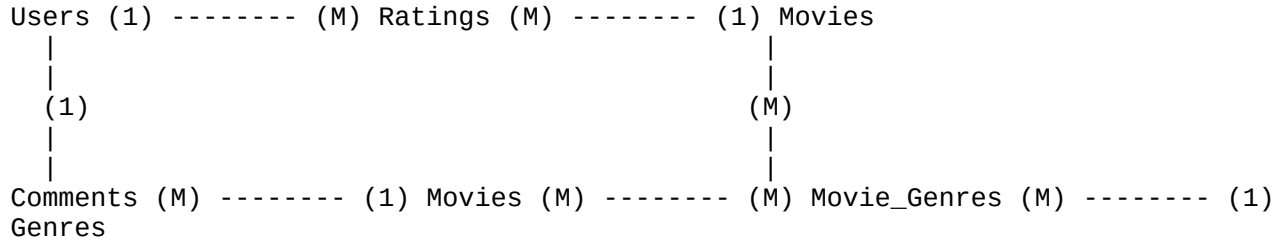
```
CREATE TABLE Ratings (
```

```
    rating_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    movie_id INT NOT NULL,
    rating DECIMAL(2,1) CHECK (rating >= 1.0 AND rating <= 5.0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (movie_id) REFERENCES Movies(movie_id) ON DELETE CASCADE,
    UNIQUE KEY unique_user_movie_rating (user_id, movie_id)
);
```

Comments Table

```
CREATE TABLE Comments (
    comment_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    movie_id INT NOT NULL,
    comment_text TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (movie_id) REFERENCES Movies(movie_id) ON DELETE CASCADE
);
```

2. Entity Relationship Diagram (ERD)



Relationships:

- Users → Ratings (1:M) - One user can give multiple ratings
- Users → Comments (1:M) - One user can write multiple comments
- Movies → Ratings (1:M) - One movie can receive multiple ratings
- Movies → Comments (1:M) - One movie can receive multiple comments
- Movies → Movie_Genres (M:M) - Movies can belong to multiple genres
- Genres → Movie_Genres (1:M) - One genre can be associated with multiple movies

3. Sample Data Insertion

```
-- Insert sample users
INSERT INTO Users (username, email, password_hash, first_name, last_name) VALUES
('john_doe', 'john@email.com', 'hashed_password_1', 'John', 'Doe'),
('jane_smith', 'jane@email.com', 'hashed_password_2', 'Jane', 'Smith'),
('movie_lover', 'lover@email.com', 'hashed_password_3', 'Movie', 'Lover');

-- Insert sample genres
INSERT INTO Genres (genre_name, description) VALUES
('Action', 'High-energy films with physical stunts and chases'),
('Drama', 'Serious, plot-driven films'),
('Comedy', 'Films designed to make audiences laugh'),
('Sci-Fi', 'Science fiction films'),
```

```
('Horror', 'Films intended to frighten and create suspense');

-- Insert sample movies
INSERT INTO Movies (title, description, release_date, director,
duration_minutes) VALUES
('The Matrix', 'A hacker discovers reality is a simulation', '1999-03-31',
'Wachowski Sisters', 136),
('Inception', 'A thief enters dreams to plant ideas', '2010-07-16', 'Christopher
Nolan', 148),
('The Shawshank Redemption', 'Two imprisoned men bond over years', '1994-09-23',
'Frank Darabont', 142);

-- Insert movie-genre relationships
INSERT INTO Movie_Genres (movie_id, genre_id) VALUES
(1, 1), (1, 4), -- The Matrix: Action, Sci-Fi
(2, 1), (2, 4), (2, 2), -- Inception: Action, Sci-Fi, Drama
(3, 2); -- The Shawshank Redemption: Drama

-- Insert sample ratings
INSERT INTO Ratings (user_id, movie_id, rating) VALUES
(1, 1, 4.5),
(2, 1, 5.0),
(3, 1, 4.0),
(1, 2, 4.8),
(2, 2, 4.7),
(1, 3, 5.0);

-- Insert sample comments
INSERT INTO Comments (user_id, movie_id, comment_text) VALUES
(1, 1, 'Amazing visual effects and mind-bending plot!'),
(2, 1, 'A classic that redefined sci-fi cinema.'),
(3, 1, 'Great movie but a bit confusing at times.'),
(1, 2, 'Nolan at his finest. Complex but rewarding.'),
(2, 2, 'Incredible cinematography and storytelling.');
```

4. Sample Queries

Query 1: Retrieve all ratings and comments for a specific movie

```
-- Get all ratings and comments for "The Matrix" (movie_id = 1)
SELECT
    m.title AS movie_title,
    u.username,
    u.first_name,
    u.last_name,
    r.rating,
    c.comment_text,
    r.created_at AS rating_date,
    c.created_at AS comment_date
FROM Movies m
LEFT JOIN Ratings r ON m.movie_id = r.movie_id
LEFT JOIN Comments c ON m.movie_id = c.movie_id
LEFT JOIN Users u ON (r.user_id = u.user_id OR c.user_id = u.user_id)
WHERE m.movie_id = 1
ORDER BY r.created_at DESC, c.created_at DESC;

-- Alternative: Separate queries for better organization
-- Get all ratings for a specific movie
SELECT
    m.title,
    u.username,
    u.first_name,
```

```
    u.last_name,
    r.rating,
    r.created_at
FROM Movies m
JOIN Ratings r ON m.movie_id = r.movie_id
JOIN Users u ON r.user_id = u.user_id
WHERE m.movie_id = 1
ORDER BY r.created_at DESC;

-- Get all comments for a specific movie
SELECT
    m.title,
    u.username,
    u.first_name,
    u.last_name,
    c.comment_text,
    c.created_at
FROM Movies m
JOIN Comments c ON m.movie_id = c.movie_id
JOIN Users u ON c.user_id = u.user_id
WHERE m.movie_id = 1
ORDER BY c.created_at DESC;
```

Query 2: Calculate the average rating for a movie

```
-- Calculate average rating for a specific movie
SELECT
    m.title,
    ROUND(AVG(r.rating), 2) AS average_rating,
    COUNT(r.rating) AS total_ratings
FROM Movies m
JOIN Ratings r ON m.movie_id = r.movie_id
WHERE m.movie_id = 1
GROUP BY m.movie_id, m.title;

-- Calculate average rating for all movies
SELECT
    m.movie_id,
    m.title,
    ROUND(AVG(r.rating), 2) AS average_rating,
    COUNT(r.rating) AS total_ratings
FROM Movies m
LEFT JOIN Ratings r ON m.movie_id = r.movie_id
GROUP BY m.movie_id, m.title
ORDER BY average_rating DESC;
```

Query 3: Retrieve all movies associated with a particular genre

```
-- Get all movies in "Action" genre
SELECT
    m.movie_id,
    m.title,
    m.description,
    m.release_date,
    m.director,
    g.genre_name
FROM Movies m
JOIN Movie_Genres mg ON m.movie_id = mg.movie_id
JOIN Genres g ON mg.genre_id = g.genre_id
WHERE g.genre_name = 'Action'
ORDER BY m.release_date DESC;
```

```
-- Get all movies with their genres (showing multiple genres per movie)
SELECT
    m.movie_id,
    m.title,
    GROUP_CONCAT(g.genre_name ORDER BY g.genre_name SEPARATOR ', ') AS genres
FROM Movies m
LEFT JOIN Movie_Genres mg ON m.movie_id = mg.movie_id
LEFT JOIN Genres g ON mg.genre_id = g.genre_id
GROUP BY m.movie_id, m.title
ORDER BY m.title;
```

5. Additional Useful Queries

Get movie details with ratings and genre information

```
SELECT
    m.title,
    m.director,
    m.release_date,
    ROUND(AVG(r.rating), 2) AS avg_rating,
    COUNT(r.rating) AS rating_count,
    COUNT(c.comment_id) AS comment_count,
    GROUP_CONCAT(DISTINCT g.genre_name ORDER BY g.genre_name SEPARATOR ', ') AS
genres
FROM Movies m
LEFT JOIN Ratings r ON m.movie_id = r.movie_id
LEFT JOIN Comments c ON m.movie_id = c.movie_id
LEFT JOIN Movie_Genres mg ON m.movie_id = mg.movie_id
LEFT JOIN Genres g ON mg.genre_id = g.genre_id
GROUP BY m.movie_id, m.title, m.director, m.release_date
ORDER BY avg_rating DESC;
```

Get top-rated movies by genre

```
SELECT
    g.genre_name,
    m.title,
    ROUND(AVG(r.rating), 2) AS avg_rating,
    COUNT(r.rating) AS rating_count
FROM Genres g
JOIN Movie_Genres mg ON g.genre_id = mg.genre_id
JOIN Movies m ON mg.movie_id = m.movie_id
LEFT JOIN Ratings r ON m.movie_id = r.movie_id
GROUP BY g.genre_id, g.genre_name, m.movie_id, m.title
HAVING COUNT(r.rating) >= 2  -- Only movies with at least 2 ratings
ORDER BY g.genre_name, avg_rating DESC;
```

6. Indexes for Performance Optimization

```
-- Create indexes for better query performance
CREATE INDEX idx_ratings_movie_id ON Ratings(movie_id);
CREATE INDEX idx_ratings_user_id ON Ratings(user_id);
CREATE INDEX idx_comments_movie_id ON Comments(movie_id);
CREATE INDEX idx_comments_user_id ON Comments(user_id);
CREATE INDEX idx_movie_genres_movie_id ON Movie_Genres(movie_id);
CREATE INDEX idx_movie_genres_genre_id ON Movie_Genres(genre_id);
CREATE INDEX idx_movies_title ON Movies(title);
CREATE INDEX idx_movies_release_date ON Movies(release_date);
```

7. Constraints and Business Rules

1. **Rating Constraints**: Ratings must be between 1.0 and 5.0
2. **Unique Ratings**: Each user can only rate a movie once
3. **Cascade Deletes**: When a user or movie is deleted, related ratings and comments are also deleted
4. **Required Fields**: All foreign keys are NOT NULL to maintain referential integrity
5. **Genre Flexibility**: Movies can belong to multiple genres through the junction table