CSE414 - Database Project Report

Ayşe Begüm Nur

June 14, 2024

Contents

1 User Requirements

- **User Registration and Management:**
  - Registration: Users should be able to register with a username, password, and email.
  - Profile Management: Users can update their profile information, including changing their password and email.
  - Account Deletion: Users can delete their accounts.
- **Review System:**
  - Write Reviews: Users can write reviews for TV shows and movies, which include ratings and review text.
  - Edit Reviews: Users can update their previously written reviews.
  - Delete Reviews: Users can delete their reviews.
  - View Reviews: Users can view reviews written by others and see the average rating for each TV show or movie.
- **Watchlist Management:**
  - Create Watchlists: Users can create multiple watchlists.
  - Add/Remove Items: Users can add TV shows and movies to their watchlists and remove them as desired.
  - View Watchlists: Users can view their watchlists and the items within them.
- **Recommendations:**
  - Personalized Recommendations: Users can receive personalized recommendations based on their viewing history and reviews.
  - View Recommendations: Users can view and manage their recommendations.
- **Search and Filter:**
  - Search: Users can search for TV shows and movies by title, genre, rating, and release date.
  - Filter: Users can filter search results based on criteria like genre, rating, and release date.
- **Critic Reviews:**
  - View Critic Reviews: Users can view reviews written by critics, which include the critic's publication and average rating.
  - Critics Writing Reviews: Critics can write, update, and delete their reviews, which are tagged as "Critic Reviews".
- **Cast and Crew Information:**
  - Detailed Information: Users can view detailed information about cast and crew members, including their roles and the shows/movies they've worked on.
  - Search Cast and Crew: Users can search for specific cast and crew members.
- **User Roles:**
  - Regular Users: Can write reviews, create watchlists, and receive recommendations.
  - Critics: Have additional attributes like publication, review count, and average rating. They can write "Critic Reviews".
- **Analytics and Reporting:**
  - User Activity: Admins can view analytics on user activity, such as the most reviewed shows/-movies and user engagement.
  - Review Analytics: Reporting on average ratings by critics and regular users, and trends in reviews over time.

2 Entity-Relationship (E-R) Diagram

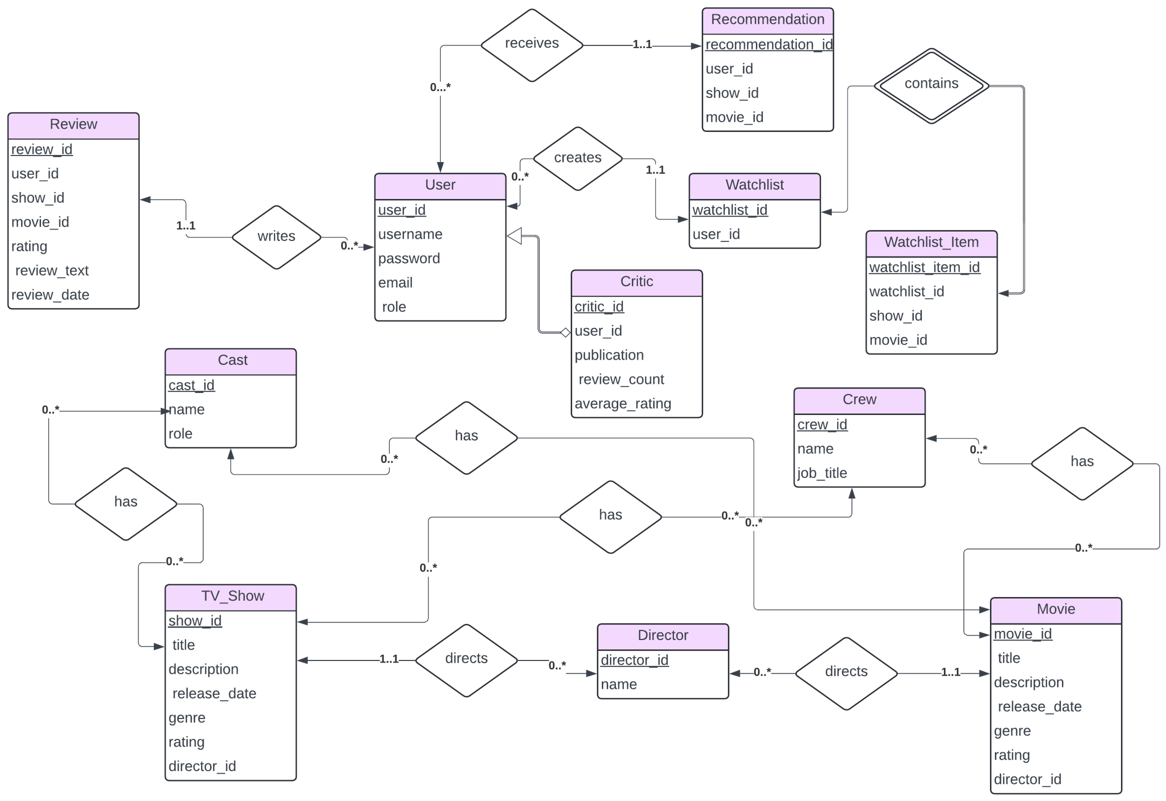


Figure 1: Entity-Relationship Diagram

3 Functional Dependencies

- **User:**

$$\text{user\_id} \rightarrow \text{username, password, email, role}$$

- **Critic:**

$$\text{critic\_id} \rightarrow \text{user\_id, publication, review\_count, average\_rating}$$

- **Review:**

$$\text{review\_id} \rightarrow \text{user\_id, show\_id, movie\_id, rating, review\_text, review\_date}$$

- **Recommendation:**

$$\text{recommendation\_id} \rightarrow \text{user\_id, show\_id, movie\_id}$$

- **Watchlist:**

$$\text{watchlist\_id} \rightarrow \text{user\_id}$$

- **Watchlist_Item:**

$$\text{watchlist\_item\_id} \rightarrow \text{watchlist\_id, show\_id, movie\_id}$$

- **Cast:**

$$\text{cast\_id} \rightarrow \text{name, role}$$

- **Crew:**

$$\text{crew\_id} \rightarrow \text{name, job\_title}$$

- **TV_Show:**

$$\text{show\_id} \rightarrow \text{title, description, release\_date, genre, rating, director\_id}$$

- **Director:**

$$\text{director\_id} \rightarrow \text{name}$$

- **Movie:**

$$\text{movie\_id} \rightarrow \text{title, description, release\_date, genre, rating, director\_id}$$

4 Normalization

4.1 User_Critic Table

4.1.1 Initial Form

```
User_Critic(user_id, username, password, email, role, publication, review_count, average_rating)
```

Functional Dependencies:

$$\text{user\_id} \rightarrow \text{username, password, email, role}$$

$$\text{user\_id} \rightarrow \text{publication, review\_count, average\_rating}$$

4.1.2 1NF

The table is already in 1NF.

4.1.3 2NF

Decomposition to achieve 2NF:

```
User(user_id, username, password, email, role)
Critic(critic_id, user_id, publication, review_count, average_rating)
```

4.1.4 3NF and BCNF

Both tables are in 3NF and BCNF.

4.2 Review Table

4.2.1 Initial Form

```
Review(review_id, user_id, show_id, movie_id, rating, review_text, review_date, username)
```

Functional Dependencies:

$$\text{review\_id} \rightarrow \text{user\_id, show\_id, movie\_id, rating, review\_text, review\_date}$$

$$\text{user\_id} \rightarrow \text{username}$$

4.2.2 1NF

The table is already in 1NF.

4.2.3 2NF

The table is in 2NF.

4.2.4 3NF

Decomposition to achieve 3NF:

```
Review(review_id, user_id, show_id, movie_id, rating, review_text, review_date)
User_Info(user_id, username)
```

4.2.5 BCNF

Both tables are in BCNF.

4.3 TV_Show Table

4.3.1 Initial Form

```
TV_Show(show_id, title, description, release_date, genre, rating, director_id, director_name)
```

Functional Dependencies:

$$\text{show\_id} \rightarrow \text{title, description, release\_date, genre, rating, director\_id, director\_name}$$

$$\text{director\_id} \rightarrow \text{director\_name}$$

4.3.2 1NF

The table is already in 1NF.

4.3.3 2NF

The table is in 2NF.

4.3.4 3NF

Decomposition to achieve 3NF:

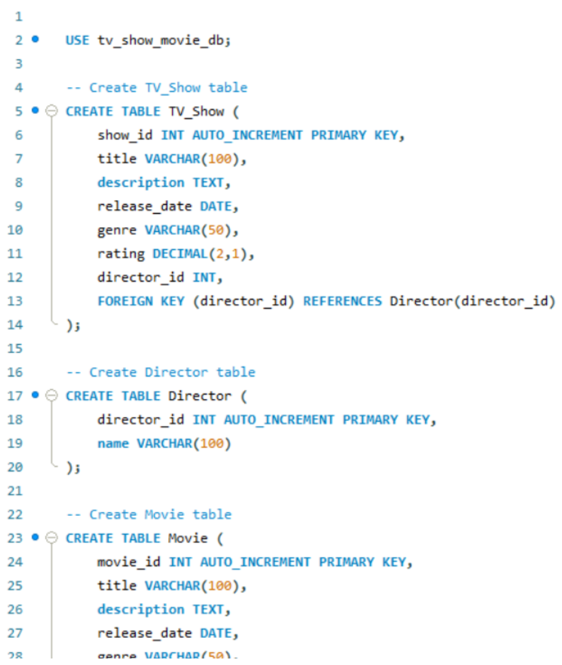
```
TV_Show(show_id, title, description, release_date, genre, rating, director_id)
Director(director_id, director_name)
```

4.3.5 BCNF

Both tables are in BCNF.

5 Database Schema

```
User(user_id, username, password, email, role)
Critic(critic_id, user_id, publication, review_count, average_rating)
Review(review_id, user_id, show_id, movie_id, rating, review_text, review_date)
User_Info(user_id, username)
TV_Show(show_id, title, description, release_date, genre, rating, director_id)
Director(director_id, director_name)
```



```sql

USE tv_show_movie_db;

-- Create TV_Show table
CREATE TABLE TV_Show (
    show_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    description TEXT,
    release_date DATE,
    genre VARCHAR(50),
    rating DECIMAL(2,1),
    director_id INT,
    FOREIGN KEY (director_id) REFERENCES Director(director_id)
);

-- Create Director table
CREATE TABLE Director (
    director_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);

-- Create Movie table
CREATE TABLE Movie (
    movie_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    description TEXT,
    release_date DATE,
    genre VARCHAR(50),
```

Figure 2: Defining the tables in mySql

6 Database Implementation

The implementation of the TV Shows and Movies Management System involved several steps, from database design to deployment.

6.1 Tools and Technologies

- **Database Management System (DBMS):** MySQL
- **Programming Language:** Python
- **Libraries and Frameworks:** MySQL Connector, Flask (for API), Tkinter (for GUI)
- **Development Environment:** PyCharm, MySQL Workbench

6.2 Steps

1. **Database Design**
    - Created the Entity-Relationship (E-R) diagram to visualize the database structure.
    - Identified entities, attributes, and relationships.
2. **Schema Creation**
    - Defined tables, columns, and relationships based on the E-R diagram.
    - Ensured normalization to 3NF and BCNF for data integrity and efficiency.
3. **Database Setup**
    - Created the database and tables in MySQL.
    - Implemented primary keys, foreign keys, and indexes.
4. **Data Insertion**
    - Inserted sample data into tables for testing.
    - Used SQL scripts and Python code to automate data insertion.

5. **Stored Procedures and Triggers**
    - Created stored procedures to handle complex operations (e.g., adding movies with directors, fetching cast and crew).
    - Implemented triggers to maintain data integrity (e.g., updating average ratings, ensuring unique usernames).
6. **Views**
    - Created views to simplify complex queries and reporting (e.g., user reviews, average ratings).
7. **User Interface**
    - Developed a GUI using Tkinter for user interactions.
    - Implemented login forms, review forms, and watchlist management features.
8. **API Development**
    - Developed a RESTful API using Flask to interact with the database.
    - Implemented endpoints for CRUD operations on movies, shows, reviews, and users.
9. **Testing**
    - Conducted extensive testing to ensure all functionalities work as expected.
    - Tested stored procedures, triggers, views, and API endpoints.
10. **Deployment**
    - Deployed the database on a MySQL server.
    - Deployed the application on a local server for demonstration.

6.3 Challenges and Solutions

- **Concurrency Control:**
    - **Challenge:** Ensuring data integrity with concurrent transactions.
    - **Solution:** Implemented locks and transaction isolation levels.
- **Data Retrieval:**
    - **Challenge:** Managing complex relationships and data retrieval.
    - **Solution:** Used views and stored procedures to simplify queries and operations.

7 User Interface

Homepage shows five different buttons: User login, critic login, admin login, listing most watched movies and listing most watched shows.User and critic logins require registration by entering the username and password.

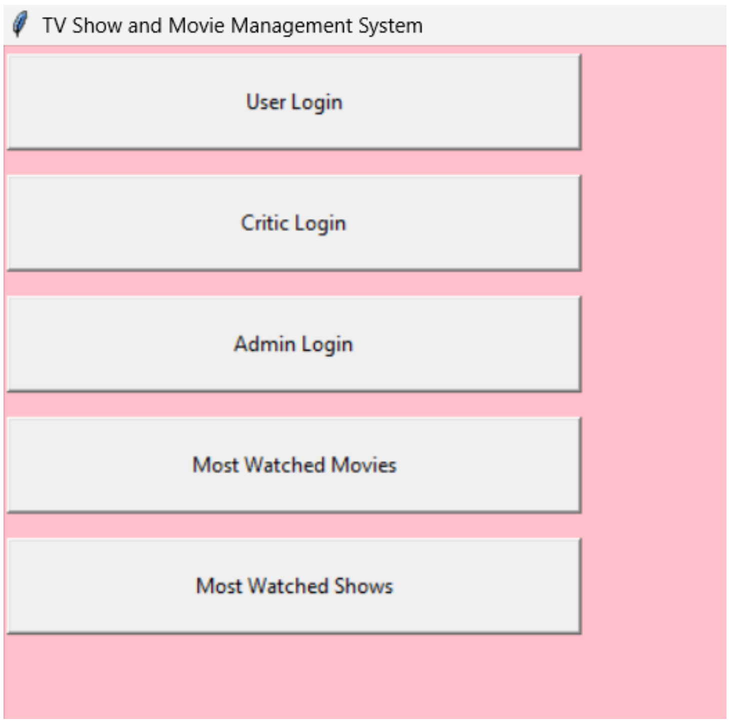


Figure 3: Homepage

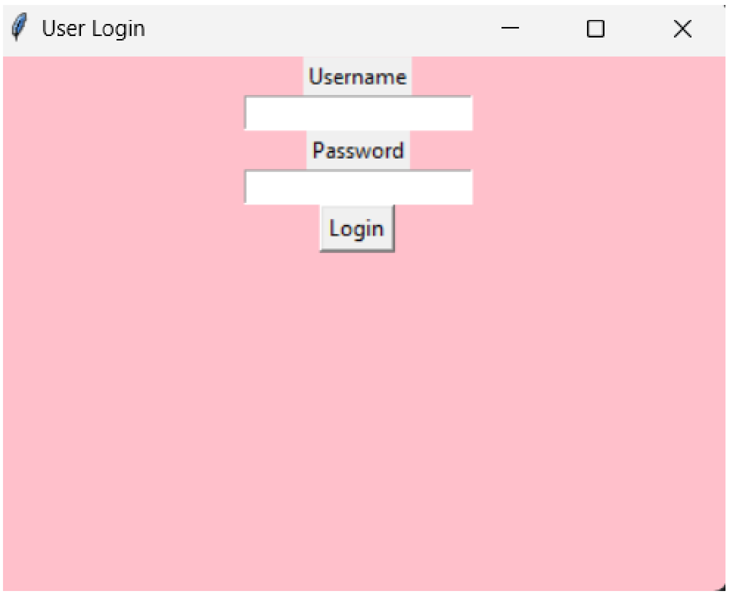


Figure 4: User login

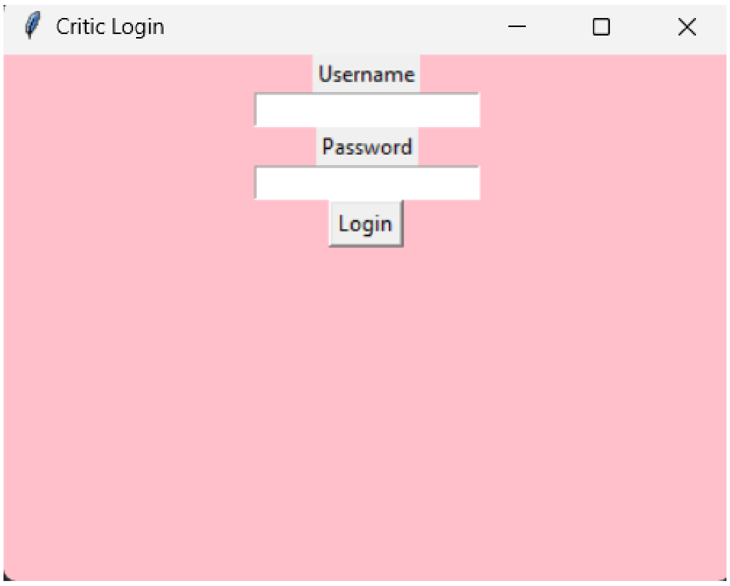
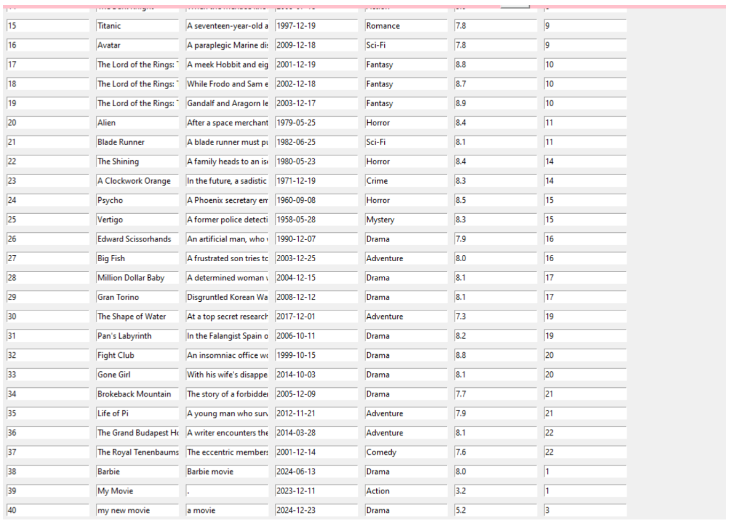


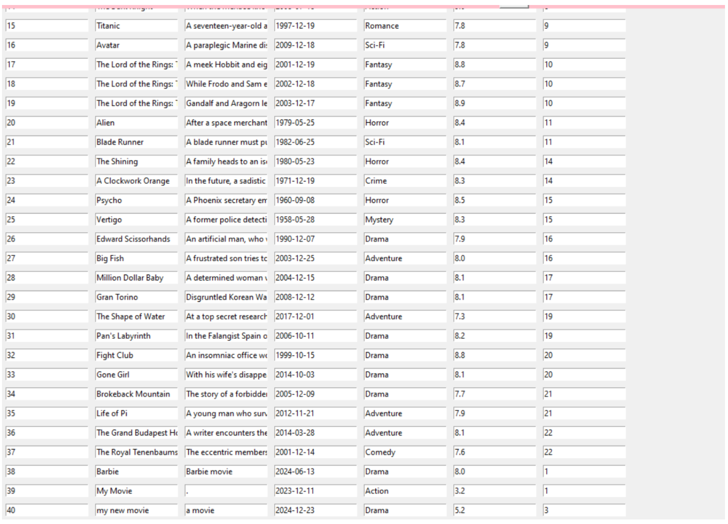
Figure 5: Critic Login

An admin can list shows and movies, add shows and movies, list crew and cast. Also admin can list most watched and most reviewed movies along with worst rated. They can view users and reviews as well.



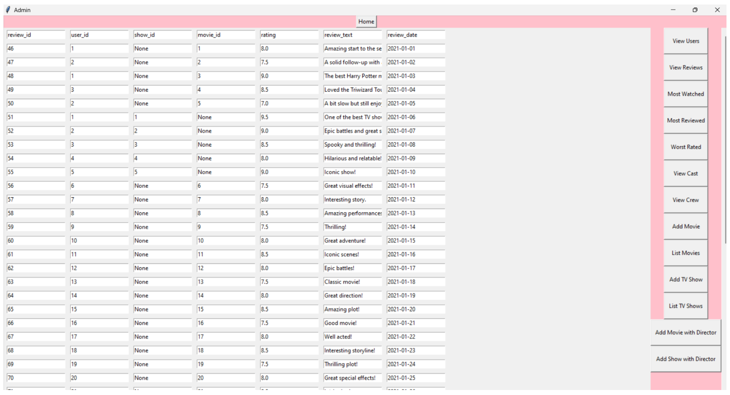
15	Titanic	A seventeen-year-old a	1997-12-19	Romance	7.8	9
16	Avatar	A paraplegic Marine di:	2009-12-18	Sci-Fi	7.8	9
17	The Lord of the Rings: `	A meek Hobbit and eig	2001-12-19	Fantasy	8.8	10
18	The Lord of the Rings: `	While Frodo and Sam e	2002-12-18	Fantasy	8.7	10
19	The Lord of the Rings: `	Gandalf and Aragorn le	2003-12-17	Fantasy	8.9	10
20	Alien	After a space merchant	1979-05-25	Horror	8.4	11
21	Blade Runner	A blade runner must pu	1982-06-25	Sci-Fi	8.1	11
22	The Shining	A family heads to an is	1980-05-23	Horror	8.4	14
23	A Clockwork Orange	In the future, a sadistic	1971-12-19	Crime	8.3	14
24	Psycho	A Phoenix secretary em	1960-09-08	Horror	8.5	15
25	Vertigo	A former police detecti	1958-05-28	Mystery	8.3	15
26	Edward Scissorhands	An artificial man, who	1990-12-07	Drama	7.9	16
27	Big Fish	A frustrated son tries tc	2003-12-25	Adventure	8.0	16
28	Million Dollar Baby	A determined woman	2004-12-15	Drama	8.1	17
29	Gran Torino	Disgruntled Korean Wa	2008-12-12	Drama	8.1	17
30	The Shape of Water	At a top secret research	2017-12-01	Adventure	7.3	19
31	Pan's Labyrinth	In the Falangist Spain o	2006-10-11	Drama	8.2	19
32	Fight Club	An insomniac office wc	1999-10-15	Drama	8.8	20
33	Gone Girl	With his wife's disappe	2014-10-03	Drama	8.1	20
34	Brokeback Mountain	The story of a forbidde	2005-12-09	Drama	7.7	21
35	Life of Pi	A young man who surv	2012-11-21	Adventure	7.9	21
36	The Grand Budapest Hc	A writer encounters the	2014-03-28	Adventure	8.1	22
37	The Royal Tenenbaums	The eccentric members	2001-12-14	Comedy	7.6	22
38	Barbie	Barbie movie	2024-06-13	Drama	8.0	1
39	My Movie	.	2023-12-11	Action	3.2	1
40	my new movie	a movie	2024-12-23	Drama	5.2	3

Figure 6: Adding a movie by admin



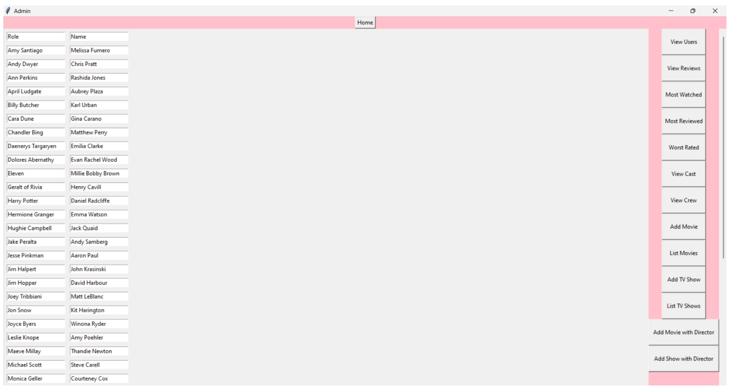
15	Titanic	A seventeen-year-old a	1997-12-19	Romance	7.8	9
16	Avatar	A paraplegic Marine di:	2009-12-18	Sci-Fi	7.8	9
17	The Lord of the Rings: '	A meek Hobbit and eig	2001-12-19	Fantasy	8.8	10
18	The Lord of the Rings: '	While Frodo and Sam e	2002-12-18	Fantasy	8.7	10
19	The Lord of the Rings: '	Gandalf and Aragorn le	2003-12-17	Fantasy	8.9	10
20	Alien	After a space merchant	1979-05-25	Horror	8.4	11
21	Blade Runner	A blade runner must pu	1982-06-25	Sci-Fi	8.1	11
22	The Shining	A family heads to an is	1980-05-23	Horror	8.4	14
23	A Clockwork Orange	In the future, a sadistic	1971-12-19	Crime	8.3	14
24	Psycho	A Phoenix secretary em	1960-09-08	Horror	8.5	15
25	Vertigo	A former police detecti	1958-05-28	Mystery	8.3	15
26	Edward Scissorhands	An artificial man, who	1990-12-07	Drama	7.9	16
27	Big Fish	A frustrated son tries tc	2003-12-25	Adventure	8.0	16
28	Million Dollar Baby	A determined woman v	2004-12-15	Drama	8.1	17
29	Gran Torino	Disgruntled Korean Wa	2008-12-12	Drama	8.1	17
30	The Shape of Water	At a top secret research	2017-12-01	Adventure	7.3	19
31	Pan's Labyrinth	In the Falangist Spain o	2006-10-11	Drama	8.2	19
32	Fight Club	An insomniac office wc	1999-10-15	Drama	8.8	20
33	Gone Girl	With his wife's disappe	2014-10-03	Drama	8.1	20
34	Brokeback Mountain	The story of a forbidde	2005-12-09	Drama	7.7	21
35	Life of Pi	A young man who surv	2012-11-21	Adventure	7.9	21
36	The Grand Budapest Hc	A writer encounters the	2014-03-28	Adventure	8.1	22
37	The Royal Tenenbaums	The eccentric members	2001-12-14	Comedy	7.6	22
38	Barbie	Barbie movie	2024-06-13	Drama	8.0	1
39	My Movie	.	2023-12-11	Action	3.2	1
40	my new movie	a movie	2024-12-23	Drama	5.2	3

Figure 7: Movie added



review_id	user_id	show_id	movie_id	rating	review_text	review_date
46	1	None	1	8.0	Amazing start to the se	2021-01-01
47	2	None	2	7.5	A solid follow-up with	2021-01-02
48	1	None	3	9.0	The best Harry Potter m	2021-01-03
49	3	None	4	8.5	Loved the Triwizard Tou	2021-01-04
50	2	None	5	7.0	A bit slow but still enjo	2021-01-05
51	1	1	None	9.5	One of the best TV sho	2021-01-06
52	2	2	None	9.0	Epic battles and great s	2021-01-07
53	3	3	None	8.5	Spooky and thrilling!	2021-01-08
54	4	4	None	8.0	Hilarious and relatable!	2021-01-09
55	5	5	None	9.0	Iconic show!	2021-01-10
56	6	None	6	7.5	Great visual effects!	2021-01-11
57	7	None	7	8.0	Interesting story.	2021-01-12
58	8	None	8	8.5	Amazing performances	2021-01-13
59	9	None	9	7.5	Thrilling!	2021-01-14
60	10	None	10	8.0	Great adventure!	2021-01-15
61	11	None	11	8.5	Iconic scenes!	2021-01-16
62	12	None	12	8.0	Epic battles!	2021-01-17
63	13	None	13	7.5	Classic movie!	2021-01-18
64	14	None	14	8.0	Great direction!	2021-01-19
65	15	None	15	8.5	Amazing plot!	2021-01-20
66	16	None	16	7.5	Good movie!	2021-01-21
67	17	None	17	8.0	Well acted!	2021-01-22
68	18	None	18	8.5	Interesting storyline!	2021-01-23
69	19	None	19	7.5	Thrilling plot!	2021-01-24
70	20	None	20	8.0	Great special effects!	2021-01-25

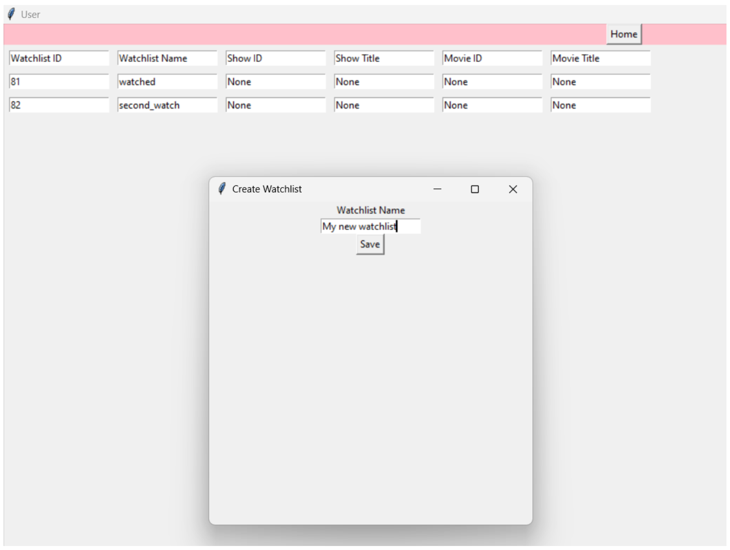
Figure 8: Admins can view all reviews



Role	Name
Amy Santiago	Melissa Fumero
Andy Dwyer	Chris Pratt
Ann Perkins	Rashida Jones
April Ludgate	Aubrey Plaza
Billy Butcher	Karl Urban
Cara Dune	Gina Carano
Chandler Bing	Matthew Perry
Daenerys Targaryen	Emilia Clarke
Dolores Abernathy	Evan Rachel Wood
Eleven	Millie Bobby Brown
Geralt of Rivia	Henry Cavill
Harry Potter	Daniel Radcliffe
Hermione Granger	Emma Watson
Hughie Campbell	Jack Quaid
Jake Peralta	Andy Samberg
Jesse Pinkman	Aaron Paul
Jim Halpert	John Krasinski
Jim Hopper	David Harbour
Joey Tribbiani	Matt LeBlanc
Jon Snow	Kit Harington
Joyce Byers	Winona Ryder
Leslie Knope	Amy Poehler
Maeve Millay	Thandie Newton
Michael Scott	Steve Carell
Monica Geller	Courteney Cox

Figure 9: Cast view by admin

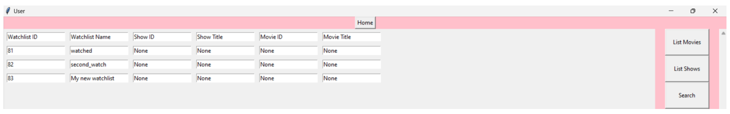
Users can list shows and movies, make a search for movie or show. They also can write reviews and update/delete them and create watch lists and update them and see all of the watch lists they created.



Watchlist ID	Watchlist Name	Show ID	Show Title	Movie ID	Movie Title
81	watched	None	None	None	None
82	second_watch	None	None	None	None

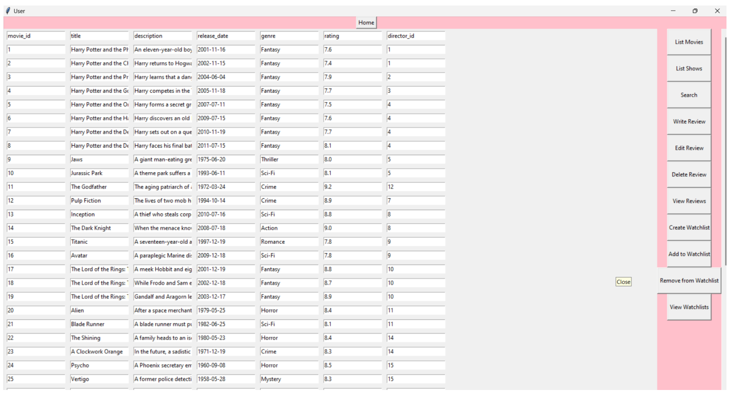
My new watchlist

Figure 10: User can view their watchlists and add new watchlists



Watchlist ID	Watchlist Name	Show ID	Show Title	Movie ID	Movie Title
81	watched	None	None	None	None
82	second_watch	None	None	None	None
83	My new watchlist	None	None	None	None

Figure 11: After adding the watchlist



movie_id	title	description	release_date	genre	rating	director_id
1	Harry Potter and the Ph	An eleven-year-old boy	2001-11-16	Fantasy	7.6	1
2	Harry Potter and the Cł	Harry returns to Hogwa	2002-11-15	Fantasy	7.4	1
3	Harry Potter and the Pr	Harry learns that a dan	2004-06-04	Fantasy	7.9	2
4	Harry Potter and the Gc	Harry competes in the	2005-11-18	Fantasy	7.7	3
5	Harry Potter and the Oı	Harry forms a secret gr	2007-07-11	Fantasy	7.5	4
6	Harry Potter and the Hi	Harry discovers an old	2009-07-15	Fantasy	7.6	4
7	Harry Potter and the De	Harry sets out on a que	2010-11-19	Fantasy	7.7	4
8	Harry Potter and the De	Harry faces his final bat	2011-07-15	Fantasy	8.1	4
9	Jaws	A giant man-eating gre	1975-06-20	Thriller	8.0	5
10	Jurassic Park	A theme park suffers a	1993-06-11	Sci-Fi	8.1	5
11	The Godfather	The aging patriarch of	1972-03-24	Crime	9.2	12
12	Pulp Fiction	The lives of two mob h	1994-10-14	Crime	8.9	7
13	Inception	A thief who steals corp	2010-07-16	Sci-Fi	8.8	8
14	The Dark Knight	When the menace knov	2008-07-18	Action	9.0	8
15	Titanic	A seventeen-year-old a	1997-12-19	Romance	7.8	9
16	Avatar	A paraplegic Marine di:	2009-12-18	Sci-Fi	7.8	9
17	The Lord of the Rings:	A meek Hobbit and eig	2001-12-19	Fantasy	8.8	10
18	The Lord of the Rings:	While Frodo and Sam e	2002-12-18	Fantasy	8.7	10
19	The Lord of the Rings:	Gandalf and Aragorn le	2003-12-17	Fantasy	8.9	10
20	Alien	After a space merchant	1979-05-25	Horror	8.4	11
21	Blade Runner	A blade runner must pu	1982-06-25	Sci-Fi	8.1	11
22	The Shining	A family heads to an is	1980-05-23	Horror	8.4	14
23	A Clockwork Orange	In the future, a sadistic	1971-12-19	Crime	8.3	14
24	Psycho	A Phoenix secretary en	1960-09-08	Horror	8.5	15
25	Vertigo	A former police detecti	1958-05-28	Mystery	8.3	15

Figure 12: Show movies for users

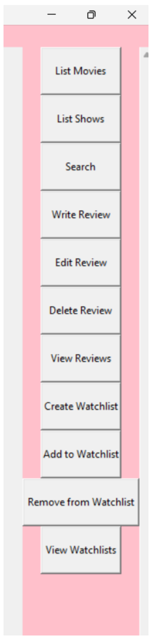
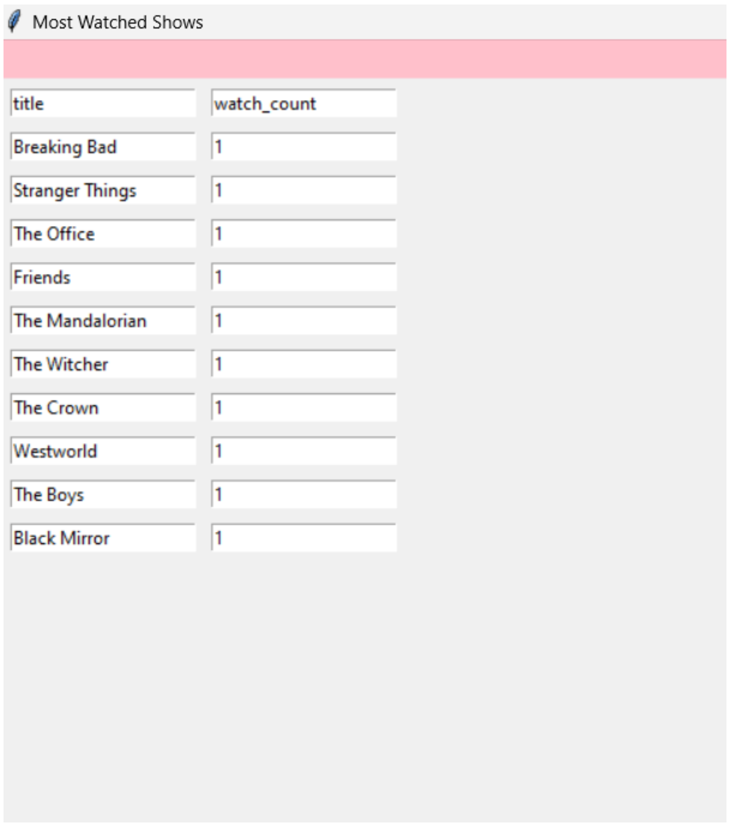


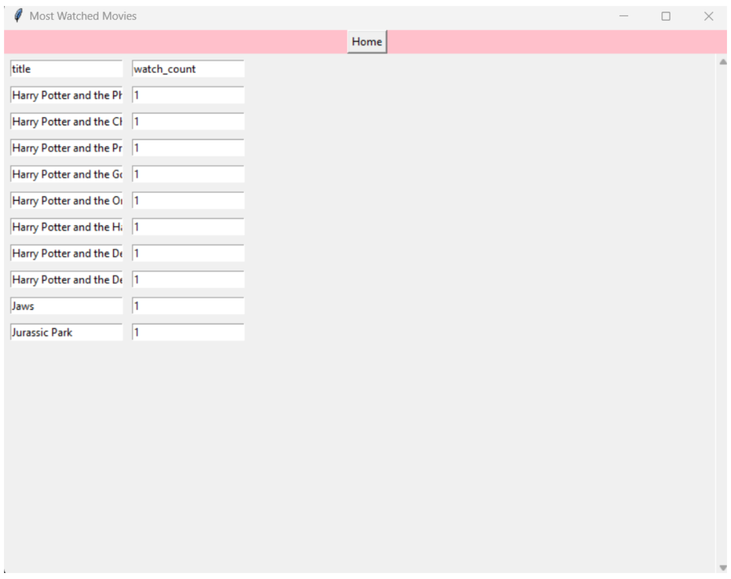
Figure 13: User Buttons

Most watched shows and movies are also listed through seperate buttons in the homepage.


Most Watched Shows

title	watch_count
Breaking Bad	1
Stranger Things	1
The Office	1
Friends	1
The Mandalorian	1
The Witcher	1
The Crown	1
Westworld	1
The Boys	1
Black Mirror	1

Figure 14: Most Watched Shows



title	watch_count
Harry Potter and the Ph	1
Harry Potter and the Ch	1
Harry Potter and the Pr	1
Harry Potter and the Go	1
Harry Potter and the Or	1
Harry Potter and the Ha	1
Harry Potter and the De	1
Harry Potter and the De	1
Jaws	1
Jurassic Park	1

Figure 15: Most Watched Movies

A critic can list movies and shows, write, delete and update reviews for tv shows and movies.They can also view their own reviews.

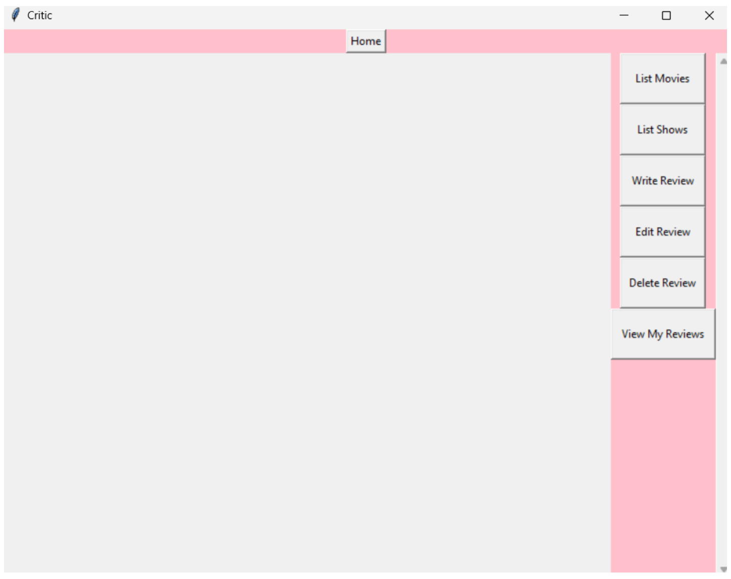
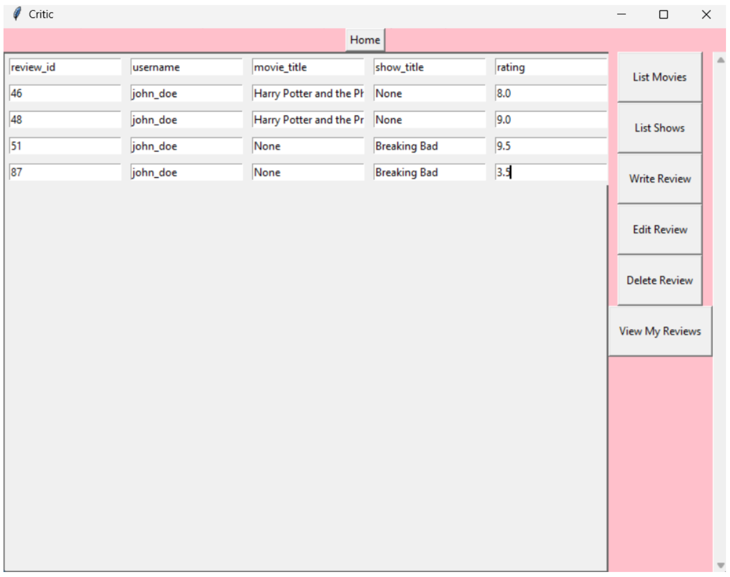
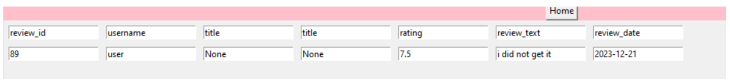


Figure 16: Critic UI



review_id	username	movie_title	show_title	rating
46	john_doe	Harry Potter and the Ph	None	8.0
48	john_doe	Harry Potter and the Pr	None	9.0
51	john_doe	None	Breaking Bad	9.5
87	john_doe	None	Breaking Bad	3.5

Figure 17: Critic Reviews



review_id	username	title	title	rating	review_text	review_date
89	user	None	None	7.5	i did not get it	2023-12-21

Figure 18: New review added

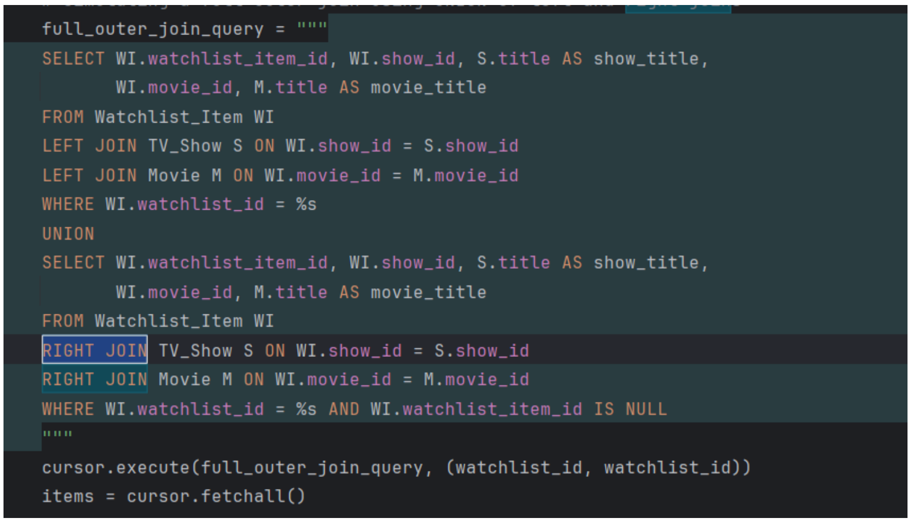
8 Query Development

- Query details...

9 Joins

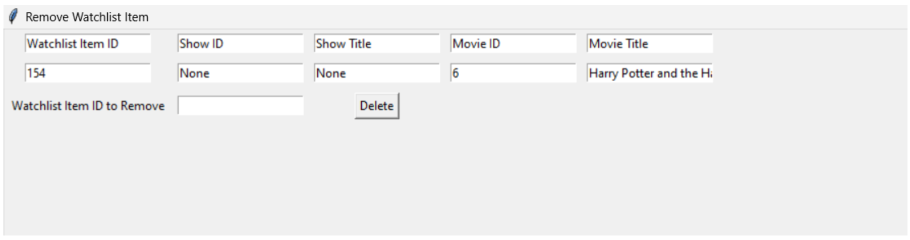
9.1 Right Join

My sql does not support full join.



```
full_outer_join_query = """
SELECT WI.watchlist_item_id, WI.show_id, S.title AS show_title,
       WI.movie_id, M.title AS movie_title
FROM Watchlist_Item WI
LEFT JOIN TV_Show S ON WI.show_id = S.show_id
LEFT JOIN Movie M ON WI.movie_id = M.movie_id
WHERE WI.watchlist_id = %s
UNION
SELECT WI.watchlist_item_id, WI.show_id, S.title AS show_title,
       WI.movie_id, M.title AS movie_title
FROM Watchlist_Item WI
RIGHT JOIN TV_Show S ON WI.show_id = S.show_id
RIGHT JOIN Movie M ON WI.movie_id = M.movie_id
WHERE WI.watchlist_id = %s AND WI.watchlist_item_id IS NULL
"""
cursor.execute(full_outer_join_query, (watchlist_id, watchlist_id))
items = cursor.fetchall()
```

Figure 19: Join Full

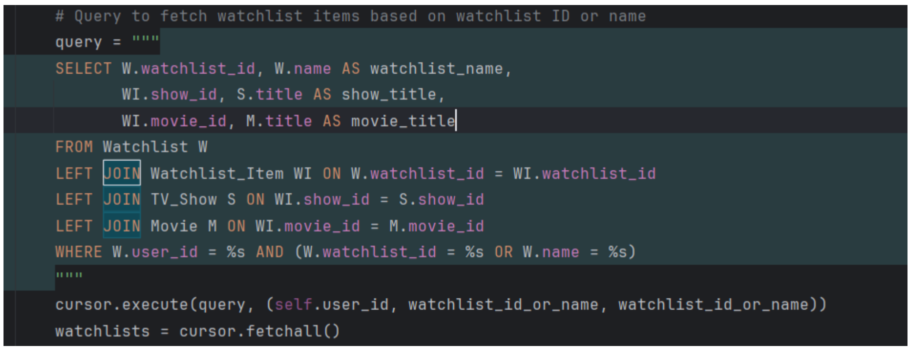


Watchlist Item ID	Show ID	Show Title	Movie ID	Movie Title
154	None	None	6	Harry Potter and the H

Watchlist Item ID to Remove

Figure 20: The Equivalent Table

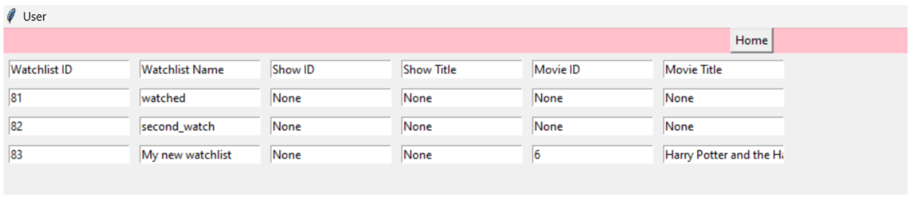
9.2 Left Join



```
# Query to fetch watchlist items based on watchlist ID or name
query = """
SELECT W.watchlist_id, W.name AS watchlist_name,
       WI.show_id, S.title AS show_title,
       WI.movie_id, M.title AS movie_title
FROM Watchlist W
LEFT JOIN Watchlist_Item WI ON W.watchlist_id = WI.watchlist_id
LEFT JOIN TV_Show S ON WI.show_id = S.show_id
LEFT JOIN Movie M ON WI.movie_id = M.movie_id
WHERE W.user_id = %s AND (W.watchlist_id = %s OR W.name = %s)
"""
cursor.execute(query, (self.user_id, watchlist_id_or_name, watchlist_id_or_name))
watchlists = cursor.fetchall()
```

Figure 21: Join Left

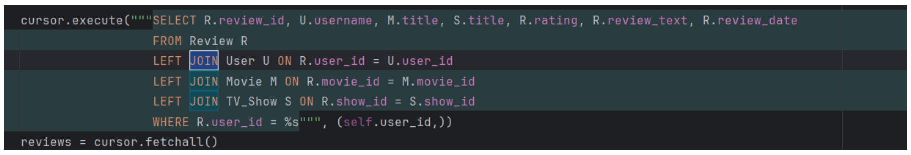
9.3 Left Join



Watchlist ID	Watchlist Name	Show ID	Show Title	Movie ID	Movie Title
81	watched	None	None	None	None
82	second_watch	None	None	None	None
83	My new watchlist	None	None	6	Harry Potter and the H

Figure 22: Join Left Example Equivalent Table

9.4 Join



```
cursor.execute("""SELECT R.review_id, U.username, M.title, S.title, R.rating, R.review_text, R.review_date
                  FROM Review R
                  LEFT JOIN User U ON R.user_id = U.user_id
                  LEFT JOIN Movie M ON R.movie_id = M.movie_id
                  LEFT JOIN TV_Show S ON R.show_id = S.show_id
                  WHERE R.user_id = %s""", (self.user_id,))
reviews = cursor.fetchall()
```

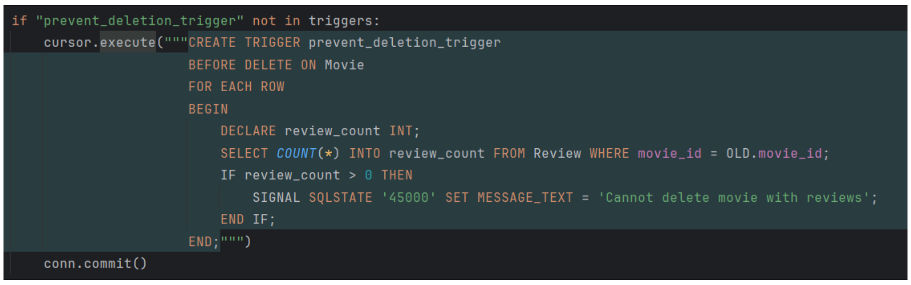
Figure 23: Join

10 Triggers

This section details the triggers implemented in the database, including their purpose and test cases.

10.1 Prevent Deletion Trigger

This trigger prevents the deletion of a movie if there are reviews associated with it. This ensures data integrity by preserving reviews for movies that users have reviewed.



```
if "prevent_deletion_trigger" not in triggers:
    cursor.execute("""CREATE TRIGGER prevent_deletion_trigger
                      BEFORE DELETE ON Movie
                      FOR EACH ROW
                      BEGIN
                          DECLARE review_count INT;
                          SELECT COUNT(*) INTO review_count FROM Review WHERE movie_id = OLD.movie_id;
                          IF review_count > 0 THEN
                              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete movie with reviews';
                          END IF;
                      END;""")
    conn.commit()
```

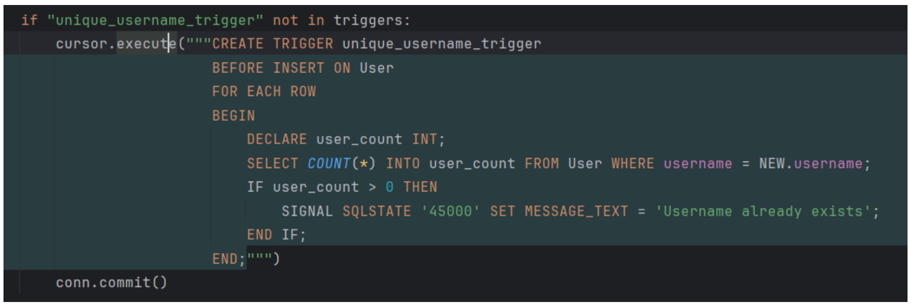
Figure 24: Prevent Deletion Trigger

Test Case:

- Attempt to delete a movie with reviews: The trigger should prevent this action and return an error message.
- Attempt to delete a movie without reviews: The movie should be successfully deleted.

10.2 Unique Username Trigger

This trigger ensures that each username is unique in the User table. If a username already exists, the trigger prevents the insertion of a new user with the same username.



```
if "unique_username_trigger" not in triggers:
    cursor.execute("""CREATE TRIGGER unique_username_trigger
                      BEFORE INSERT ON User
                      FOR EACH ROW
                      BEGIN
                          DECLARE user_count INT;
                          SELECT COUNT(*) INTO user_count FROM User WHERE username = NEW.username;
                          IF user_count > 0 THEN
                              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Username already exists';
                          END IF;
                      END;""")
    conn.commit()
```

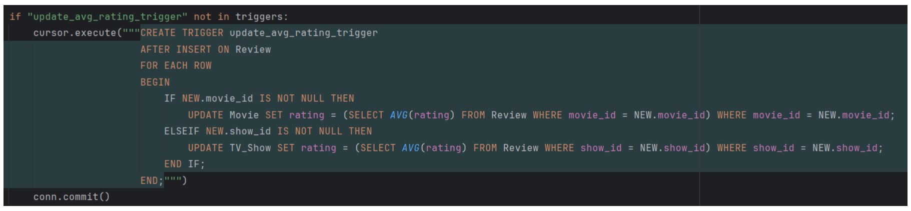
Figure 25: Unique Username Trigger

Test Case:

- Insert a user with a unique username: The insertion should succeed.
- Insert a user with a duplicate username: The trigger should prevent this action and return an error message.

10.3 Update Average Rating Trigger

This trigger updates the average rating of a movie or TV show whenever a new review is inserted. This keeps the average ratings up-to-date automatically.



```
if "update_avg_rating_trigger" not in triggers:
    cursor.execute("""CREATE TRIGGER update_avg_rating_trigger
                      AFTER INSERT ON Review
                      FOR EACH ROW
                      BEGIN
                          IF NEW.movie_id IS NOT NULL THEN
                              UPDATE Movie SET rating = (SELECT AVG(rating) FROM Review WHERE movie_id = NEW.movie_id) WHERE movie_id = NEW.movie_id;
                          ELSEIF NEW.show_id IS NOT NULL THEN
                              UPDATE TV_Show SET rating = (SELECT AVG(rating) FROM Review WHERE show_id = NEW.show_id) WHERE show_id = NEW.show_id;
                          END IF;
                      END;""")
    conn.commit()
```

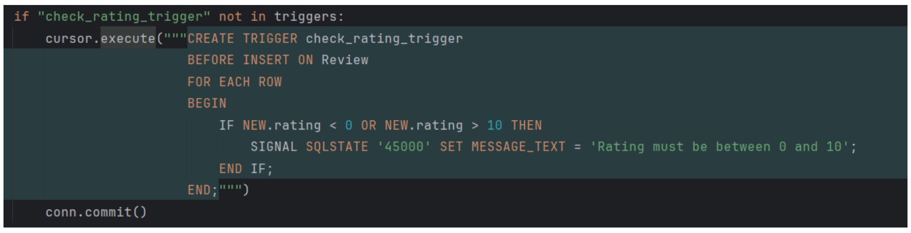
Figure 26: Update Average Rating Trigger

Test Case:

- Insert a new review for a movie: Check that the movie's average rating is updated correctly.
- Insert a new review for a TV show: Check that the TV show's average rating is updated correctly.

10.4 Check Rating Trigger

This trigger ensures that the rating provided in a review is between 0 and 10. If the rating is outside this range, the trigger prevents the insertion.

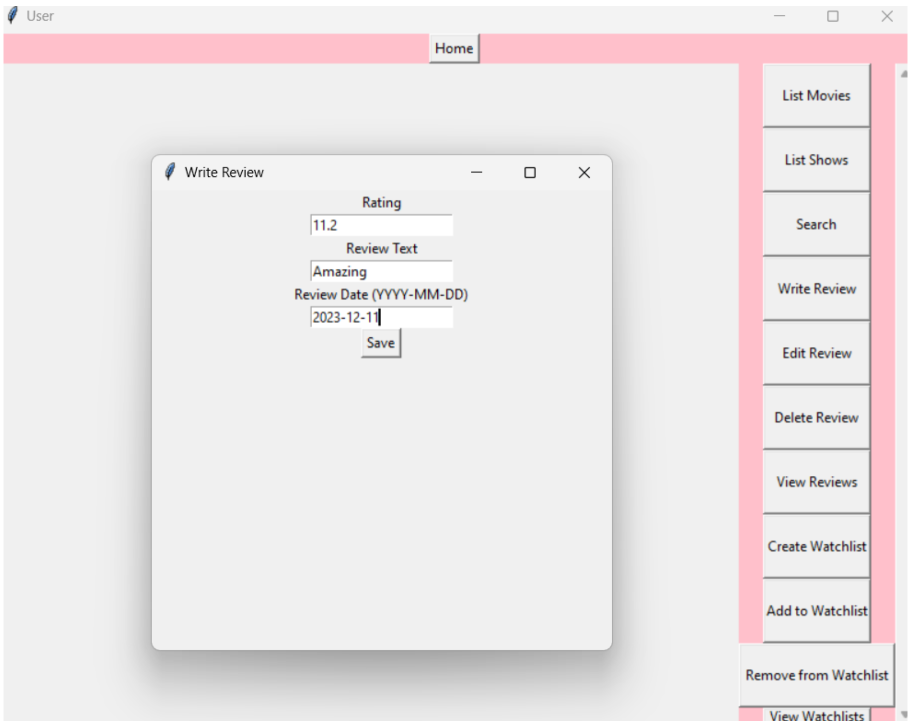


```
if "check_rating_trigger" not in triggers:
    cursor.execute("""CREATE TRIGGER check_rating_trigger
                      BEFORE INSERT ON Review
                      FOR EACH ROW
                      BEGIN
                          IF NEW.rating < 0 OR NEW.rating > 10 THEN
                              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Rating must be between 0 and 10';
                          END IF;
                      END;""")
    conn.commit()
```

Figure 27: Check Rating Trigger

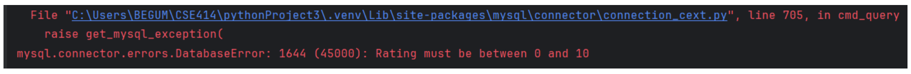
Test Case:

- Insert a review with a rating between 0 and 10: The insertion should succeed.
- Insert a review with a rating less than 0 or greater than 10: The trigger should prevent this action and return an error message.



Rating
11.2
Review Text
Amazing
Review Date (YYYY-MM-DD)
2023-12-11

Figure 28: Test Case for the Trigger

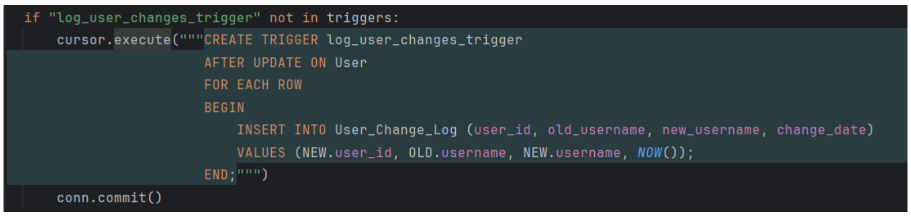


```
  File "C:\Users\BEGUM\CSE414\pythonProject3\.venv\Lib\site-packages\mysql\connector\connection_cext.py", line 705, in cmd_query
    raise get_mysql_exception(
mysql.connector.errors.DatabaseError: 1644 (45000): Rating must be between 0 and 10
```

Figure 29: Rating 0-10 Error

10.5 Log User Changes Trigger

This trigger logs changes to a user's username. It helps track modifications and maintain an audit trail.



```
if "log_user_changes_trigger" not in triggers:
    cursor.execute("""CREATE TRIGGER log_user_changes_trigger
                      AFTER UPDATE ON User
                      FOR EACH ROW
                      BEGIN
                          INSERT INTO User_Change_Log (user_id, old_username, new_username, change_date)
                          VALUES (NEW.user_id, OLD.username, NEW.username, NOW());
                      END;""")
    conn.commit()
```

Figure 30: Log User Changes Trigger

Test Case:

- Update a user's username: Check the User_Change_Log table to ensure the change is logged correctly.

11 Views

This section details the views created in the database to facilitate various queries and reporting.

11.1 UserMovieReviews View

This view combines user information, movie details, and their corresponding reviews.

```
CREATE VIEW UserMovieReviews AS
SELECT User.username, Movie.title, Review.rating, Review.review_text, Review.review_date
FROM User
JOIN Review ON User.user_id = Review.user_id
JOIN Movie ON Review.movie_id = Movie.movie_id;
```

11.2 UserShowReviews View

This view combines user information, TV show details, and their corresponding reviews.

```
CREATE VIEW UserShowReviews AS
SELECT User.username, TV_Show.title, Review.rating, Review.review_text, Review.review_date
FROM User
JOIN Review ON User.user_id = Review.user_id
JOIN TV_Show ON Review.show_id = TV_Show.show_id;
```

11.3 Movie$_AvgRatingsView$

This view calculates the average rating for each movie.

```
CREATE VIEW Movie_Avg_Ratings AS
SELECT Movie.title, AVG(Review.rating) AS avg_rating
FROM Movie
JOIN Review ON Movie.movie_id = Review.movie_id
GROUP BY Movie.title;
```

11.4 Show$_AvgRatingsView$

This view calculates the average rating for each TV show.

```
CREATE VIEW Show_Avg_Ratings AS
SELECT TV_Show.title, AVG(Review.rating) AS avg_rating
FROM TV_Show
JOIN Review ON TV_Show.show_id = Review.show_id
GROUP BY TV_Show.title;
```

11.5 User$_{Review_{Counts}}View$

This view counts the number of reviews written by each user.

```
CREATE VIEW User_Review_Counts AS
SELECT User.username, COUNT(Review.review_id) AS review_count
FROM User
JOIN Review ON User.user_id = Review.user_id
GROUP BY User.username;
```

11.6 Movies$_{Most_{Reviews}}View$

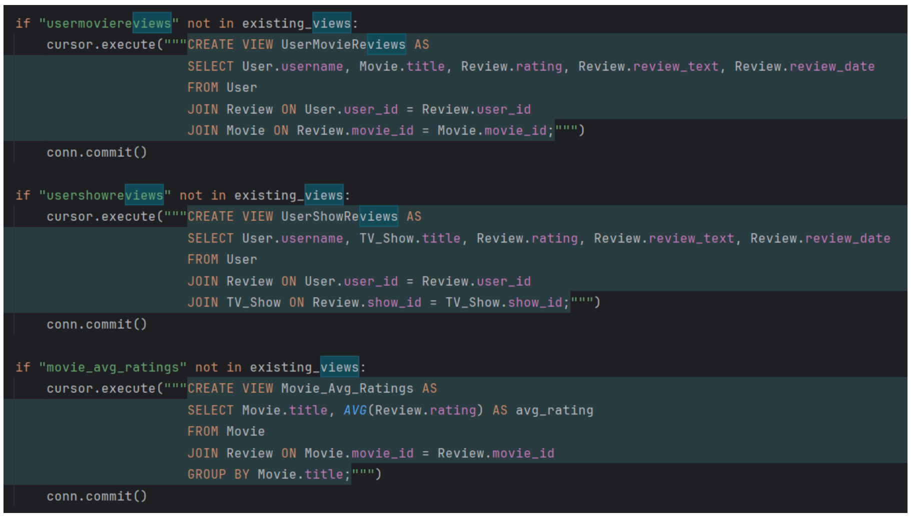
This view lists the top 10 movies with the most reviews.

```
CREATE VIEW Movies_Most_Reviews AS
SELECT Movie.title, COUNT(Review.review_id) AS review_count
FROM Movie
JOIN Review ON Movie.movie_id = Review.movie_id
GROUP BY Movie.title
ORDER BY review_count DESC
LIMIT 10;
```

11.7 Shows$_{Most_{Reviews}}View$

This view lists the top 10 TV shows with the most reviews.

```
CREATE VIEW Shows_Most_Reviews AS
SELECT TV_Show.title, COUNT(Review.review_id) AS review_count
FROM TV_Show
JOIN Review ON TV_Show.show_id = Review.show_id
GROUP BY TV_Show.title
ORDER BY review_count DESC
LIMIT 10;
```

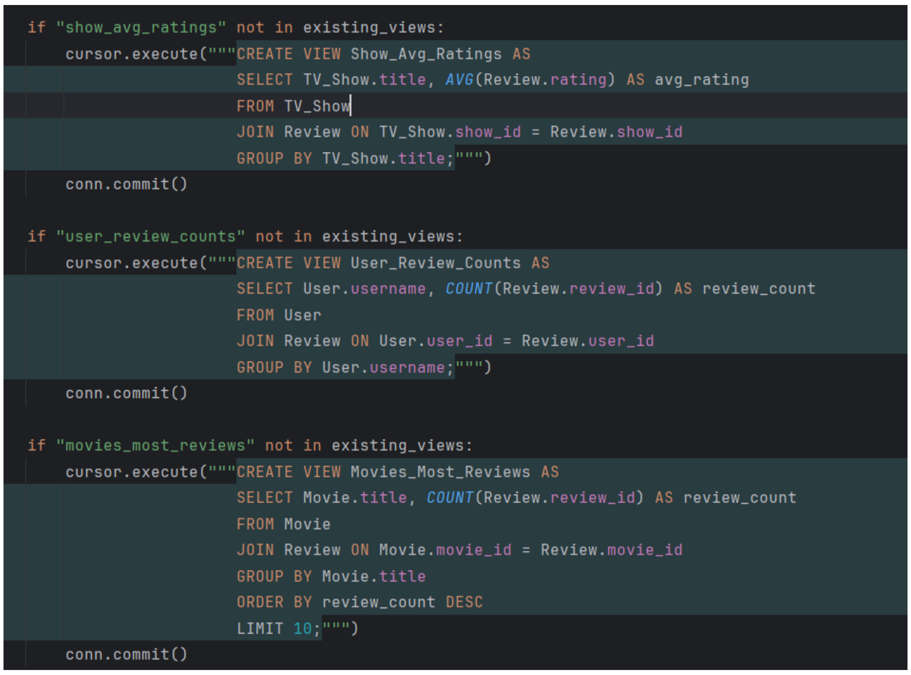


```
if "usermoviereviews" not in existing_views:
    cursor.execute("""CREATE VIEW UserMovieReviews AS
                   SELECT User.username, Movie.title, Review.rating, Review.review_text, Review.review_date
                   FROM User
                   JOIN Review ON User.user_id = Review.user_id
                   JOIN Movie ON Review.movie_id = Movie.movie_id;""")
    conn.commit()

if "usershowreviews" not in existing_views:
    cursor.execute("""CREATE VIEW UserShowReviews AS
                   SELECT User.username, TV_Show.title, Review.rating, Review.review_text, Review.review_date
                   FROM User
                   JOIN Review ON User.user_id = Review.user_id
                   JOIN TV_Show ON Review.show_id = TV_Show.show_id;""")
    conn.commit()

if "movie_avg_ratings" not in existing_views:
    cursor.execute("""CREATE VIEW Movie_Avg_Ratings AS
                   SELECT Movie.title, AVG(Review.rating) AS avg_rating
                   FROM Movie
                   JOIN Review ON Movie.movie_id = Review.movie_id
                   GROUP BY Movie.title;""")
    conn.commit()
```

Figure 31: View 1

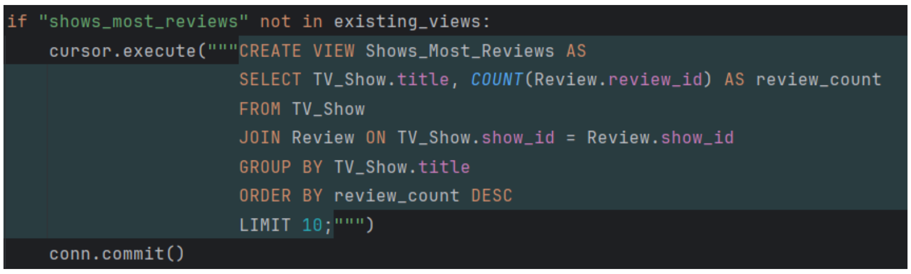


```
if "show_avg_ratings" not in existing_views:
    cursor.execute("""CREATE VIEW Show_Avg_Ratings AS
                      SELECT TV_Show.title, AVG(Review.rating) AS avg_rating
                      FROM TV_Show
                      JOIN Review ON TV_Show.show_id = Review.show_id
                      GROUP BY TV_Show.title;""")
    conn.commit()

if "user_review_counts" not in existing_views:
    cursor.execute("""CREATE VIEW User_Review_Counts AS
                      SELECT User.username, COUNT(Review.review_id) AS review_count
                      FROM User
                      JOIN Review ON User.user_id = Review.user_id
                      GROUP BY User.username;""")
    conn.commit()

if "movies_most_reviews" not in existing_views:
    cursor.execute("""CREATE VIEW Movies_Most_Reviews AS
                      SELECT Movie.title, COUNT(Review.review_id) AS review_count
                      FROM Movie
                      JOIN Review ON Movie.movie_id = Review.movie_id
                      GROUP BY Movie.title
                      ORDER BY review_count DESC
                      LIMIT 10;""")
    conn.commit()
```

Figure 32: View 2



```
if "shows_most_reviews" not in existing_views:
    cursor.execute("""CREATE VIEW Shows_Most_Reviews AS
                      SELECT TV_Show.title, COUNT(Review.review_id) AS review_count
                      FROM TV_Show
                      JOIN Review ON TV_Show.show_id = Review.show_id
                      GROUP BY TV_Show.title
                      ORDER BY review_count DESC
                      LIMIT 10;""")
    conn.commit()
```

Figure 33: View 3

12 Stored Procedures and Transactions

This section details the stored procedures and transactions implemented in the database.

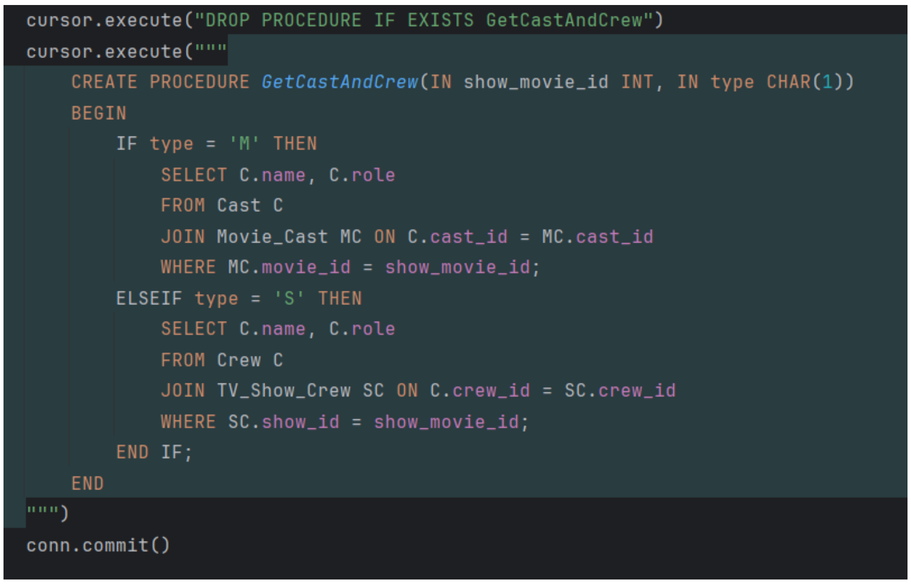
12.1 Stored Procedures

Stored procedures are used to encapsulate logic in the database to perform complex operations.

12.1.1 GetCastAndCrew Procedure

This procedure retrieves the cast and crew for a given show or movie.

```
CREATE PROCEDURE GetCastAndCrew(IN show_movie_id INT, IN type CHAR(1))
BEGIN
    IF type = 'M' THEN
        SELECT C.name, C.role
        FROM Cast C
        JOIN Movie_Cast MC ON C.cast_id = MC.cast_id
        WHERE MC.movie_id = show_movie_id;
    ELSEIF type = 'S' THEN
        SELECT C.name, C.role
        FROM Crew C
        JOIN TV_Show_Crew SC ON C.crew_id = SC.crew_id
        WHERE SC.show_id = show_movie_id;
    END IF;
END
```



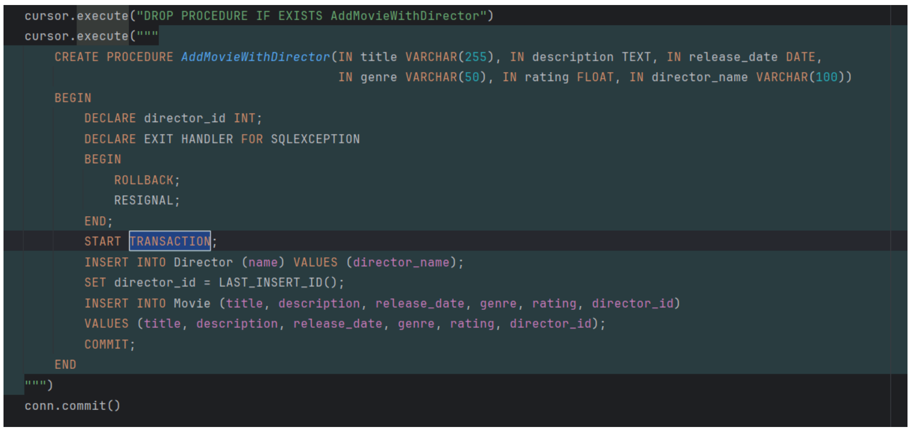
```
cursor.execute("DROP PROCEDURE IF EXISTS GetCastAndCrew")
cursor.execute("""
    CREATE PROCEDURE GetCastAndCrew(IN show_movie_id INT, IN type CHAR(1))
    BEGIN
        IF type = 'M' THEN
            SELECT C.name, C.role
            FROM Cast C
            JOIN Movie_Cast MC ON C.cast_id = MC.cast_id
            WHERE MC.movie_id = show_movie_id;
        ELSEIF type = 'S' THEN
            SELECT C.name, C.role
            FROM Crew C
            JOIN TV_Show_Crew SC ON C.crew_id = SC.crew_id
            WHERE SC.show_id = show_movie_id;
        END IF;
    END
""")
conn.commit()
```

Figure 34: Get Cast and Crew Procedure

12.1.2 AddMovieWithDirector Procedure

This procedure adds a new movie along with its director, ensuring both are added within a transaction.

```
CREATE PROCEDURE AddMovieWithDirector(IN title VARCHAR(255), IN description TEXT, IN release_date DA
                                      IN genre VARCHAR(50), IN rating FLOAT, IN director_name VARCHA
BEGIN
    DECLARE director_id INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;
    INSERT INTO Director (name) VALUES (director_name);
    SET director_id = LAST_INSERT_ID();
    INSERT INTO Movie (title, description, release_date, genre, rating, director_id)
    VALUES (title, description, release_date, genre, rating, director_id);
    COMMIT;
END
```



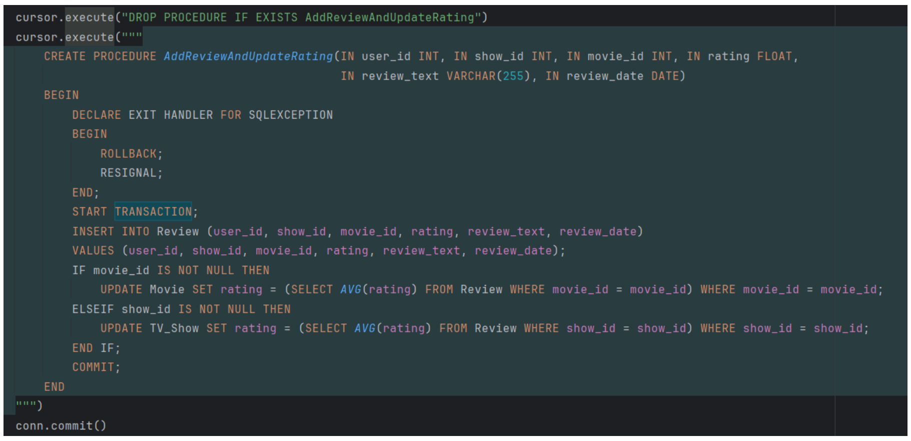
```
cursor.execute("DROP PROCEDURE IF EXISTS AddMovieWithDirector")
cursor.execute("""
    CREATE PROCEDURE AddMovieWithDirector(IN title VARCHAR(255), IN description TEXT, IN release_date DATE,
                                          IN genre VARCHAR(50), IN rating FLOAT, IN director_name VARCHAR(100))
    BEGIN
        DECLARE director_id INT;
        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
        START TRANSACTION;
        INSERT INTO Director (name) VALUES (director_name);
        SET director_id = LAST_INSERT_ID();
        INSERT INTO Movie (title, description, release_date, genre, rating, director_id)
        VALUES (title, description, release_date, genre, rating, director_id);
        COMMIT;
    END
""")
conn.commit()
```

Figure 35: Add Movie with Director

12.1.3 AddReviewAndUpdateRating Procedure

This procedure adds a new review and updates the average rating for the corresponding movie or TV show.

```
CREATE PROCEDURE AddReviewAndUpdateRating(IN user_id INT, IN show_id INT, IN movie_id INT, IN rating
                                          IN review_text VARCHAR(255), IN review_date DATE)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;
    INSERT INTO Review (user_id, show_id, movie_id, rating, review_text, review_date)
    VALUES (user_id, show_id, movie_id, rating, review_text, review_date);
    IF movie_id IS NOT NULL THEN
        UPDATE Movie SET rating = (SELECT AVG(rating) FROM Review WHERE movie_id = movie_id) WHERE m
    ELSEIF show_id IS NOT NULL THEN
        UPDATE TV_Show SET rating = (SELECT AVG(rating) FROM Review WHERE show_id = show_id) WHERE s
    END IF;
    COMMIT;
END
```



```
cursor.execute("DROP PROCEDURE IF EXISTS AddReviewAndUpdateRating")
cursor.execute("""
    CREATE PROCEDURE AddReviewAndUpdateRating(IN user_id INT, IN show_id INT, IN movie_id INT, IN rating FLOAT,
                                              IN review_text VARCHAR(255), IN review_date DATE)
    BEGIN
        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
        START TRANSACTION;
        INSERT INTO Review (user_id, show_id, movie_id, rating, review_text, review_date)
        VALUES (user_id, show_id, movie_id, rating, review_text, review_date);
        IF movie_id IS NOT NULL THEN
            UPDATE Movie SET rating = (SELECT AVG(rating) FROM Review WHERE movie_id = movie_id) WHERE movie_id = movie_id;
        ELSEIF show_id IS NOT NULL THEN
            UPDATE TV_Show SET rating = (SELECT AVG(rating) FROM Review WHERE show_id = show_id) WHERE show_id = show_id;
        END IF;
        COMMIT;
    END
""")
conn.commit()
```

Figure 36: Add Review and Update Rating Transaction

12.2 Transactions

Transactions ensure that multiple SQL operations are executed in a safe, consistent manner.

- **Adding a new user and logging the action:**

```
START TRANSACTION;
INSERT INTO User (username, password, email, role) VALUES ('new_user', 'password', 'email',
INSERT INTO User_Log (user_id, action, action_date) VALUES (LAST_INSERT_ID(), 'User Created
COMMIT;
```

- **Updating a review and recalculating average ratings:**

```
START TRANSACTION;
UPDATE Review SET rating = 9 WHERE review_id = 1;
UPDATE Movie SET rating = (SELECT AVG(rating) FROM Review WHERE movie_id = 1) WHERE movie_i
COMMIT;
```

13 Concurrency Control

Concurrency control is essential to ensure data integrity when multiple transactions occur simultaneously.

- **Implementing Locks:**

```
-- Example of locking a review row for update
START TRANSACTION;
SELECT * FROM Review WHERE review_id = 1 FOR UPDATE;
-- Perform update operations here
COMMIT;
```

- **Using Transaction Isolation Levels:**

```
-- Setting the isolation level to SERIALIZABLE
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
-- Perform operations here
COMMIT;
```

  Using appropriate isolation levels helps manage concurrent access to data, preventing issues like dirty reads, non-repeatable reads, and phantom reads.

14 Inheritance

Inheritance is implemented to distinguish between regular users and critics, with critics having additional attributes and privileges.

- **User Table:**

```
CREATE TABLE User (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    role ENUM('user', 'critic') NOT NULL
);
```

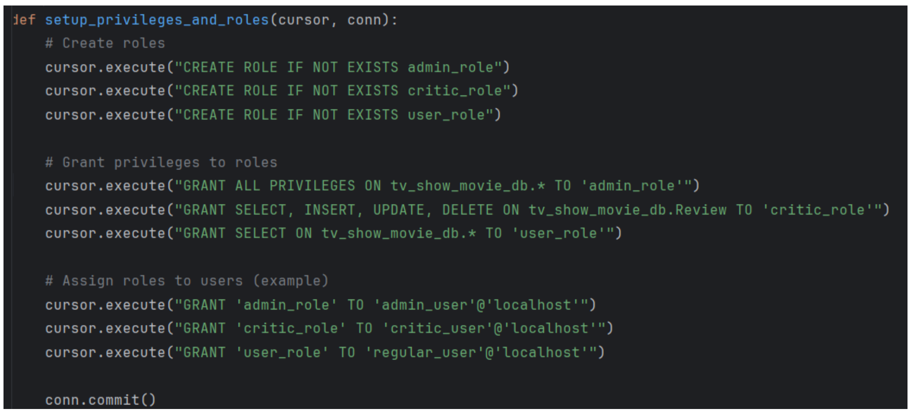
- **Critic Table:**

```
CREATE TABLE Critic (
    critic_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    publication VARCHAR(255),
    review_count INT DEFAULT 0,
    average_rating FLOAT DEFAULT 0,
    FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

The `Critic` table inherits from the `User` table by including a foreign key reference to `user_id`, allowing critics to have additional attributes like `publication`, `review_count`, and `average_rating`.

15 Privileges and Roles

Roles and privileges are created to manage access control in the database. Each role is granted specific privileges, and users are assigned roles based on their access requirements.



```
def setup_privileges_and_roles(cursor, conn):
    # Create roles
    cursor.execute("CREATE ROLE IF NOT EXISTS admin_role")
    cursor.execute("CREATE ROLE IF NOT EXISTS critic_role")
    cursor.execute("CREATE ROLE IF NOT EXISTS user_role")

    # Grant privileges to roles
    cursor.execute("GRANT ALL PRIVILEGES ON tv_show_movie_db.* TO 'admin_role'")
    cursor.execute("GRANT SELECT, INSERT, UPDATE, DELETE ON tv_show_movie_db.Review TO 'critic_role'")
    cursor.execute("GRANT SELECT ON tv_show_movie_db.* TO 'user_role'")

    # Assign roles to users (example)
    cursor.execute("GRANT 'admin_role' TO 'admin_user'@'localhost'")
    cursor.execute("GRANT 'critic_role' TO 'critic_user'@'localhost'")
    cursor.execute("GRANT 'user_role' TO 'regular_user'@'localhost'")

    conn.commit()
```

Figure 37: Grant Privileges

Implementation:

```
def setup_privileges_and_roles(cursor, conn):
    # Create roles
    cursor.execute("CREATE ROLE IF NOT EXISTS admin_role")
    cursor.execute("CREATE ROLE IF NOT EXISTS critic_role")
    cursor.execute("CREATE ROLE IF NOT EXISTS user_role")

    # Grant privileges to roles
    cursor.execute("GRANT ALL PRIVILEGES ON tv_show_movie_db.* TO ’admin_role’")
    cursor.execute("GRANT SELECT, INSERT, UPDATE, DELETE ON tv_show_movie_db.Review TO ’critic_role’
    cursor.execute("GRANT SELECT ON tv_show_movie_db.* TO ’user_role’")

    # Assign roles to users (example)
    cursor.execute("GRANT ’admin_role’ TO ’admin_user’@’localhost’")
    cursor.execute("GRANT ’critic_role’ TO ’critic_user’@’localhost’")
    cursor.execute("GRANT ’user_role’ TO ’regular_user’@’localhost’")

    conn.commit()
```

Explanation:

- **Admin Role:** Has all privileges on the database, allowing full control over all tables and operations.

- **Critic Role:** Has privileges to select, insert, update, and delete reviews, enabling critics to manage reviews.
- **User Role:** Has read-only access, allowing users to view data without making any modifications.
- **Assigning Roles:** Users are assigned roles based on their access needs. For example, `admin_user` is assigned the `admin_role`, `critic_user` is assigned the `critic_role`, and `regular_user` is assigned the `user_role`.

16 Additional Details

- **Database Design:** Ensuring efficient database design with proper indexing and normalization to optimize performance.
- **Optimization:** Regularly analyzing and optimizing queries to maintain fast response times.
- **Security Measures:** Implementing security measures such as encryption for sensitive data, regular backups, and access control to protect the database from unauthorized access.