XJTLU Entrepreneur College (Taicang) Cover Sheet

Module code and Title	DTS106TC: Introduction to Database	
School Title	School of AI and Advanced Computing	
Assignment Title	Assessment Task 001 (CW): Individual Coursework	
Submission Deadline	**May 16th, 2025 at 11:59 PM**	
Final Word Count	NA	
If you agree to let the university use your work anonymously for teaching and learning purposes, please type **"yes"** here.		**Yes**

I certify that I have read and understood the University's Policy for dealing with Plagiarism, Collusion and the Fabrication of Data (available on Learning Mall Online). With reference to this policy I certify that:

- My work does not contain any instances of plagiarism and/or collusion. My work does not contain any fabricated data.

By uploading my assignment onto Learning Mall Online, I formally declare that all of the above information is true to the best of my knowledge and belief.

Scoring – For Tutor Use	
Student ID	**2363570**

Stage of Marking	**Marker Code**	**Learning Outcomes Achieved (F/P/M/D) (please modify as appropriate)**				**Final Score**
		A	**B**	**C**	**D**	
1st Marker – red pen						
Moderation – green pen	**IM Initials**	The original mark has been accepted by the moderator (please circle as appropriate):				Y / N
		Data entry and score calculation have been checked by another tutor (please circle):				Y
2nd Marker if needed – green pen						

For Academic Office Use			**Possible Academic Infringement (please tick as appropriate)**	
Date Received	**Days late**	**Late Penalty**	☐ **Category A**	Total Academic Infringement Penalty (A,B, C, D, E, Please modify where necessary) ______________________
			☐ **Category B**	
			☐ **Category C**	
			☐ **Category D**	
			☐ **Category E**	

Q1:

The database of our website is used to store, manage and provide access to data of movies and user behaviors.

In Business Scenario,

1. Each user needs an account and relative information.
2. New movies require users to create entries and add/update information.
3. The interaction of users with movies should be stored, and the platform needs to build a relation to the movie/user.
4. Platforms need to provide the SEARCH function for users.
5. User actions and comments/ratings should be traceable.

To achieve this, our database should effectively store:

1. Each movie and its related information (meta information, review records, genre, etc.)
2. User account and personal information (email, password, review records, etc.)
3. Entry operation/comment log (to facilitate tracing and avoid legal risks)

The reasons why I choose database:

1. Database supports the construction of complex data relationships (e.x. user-review-movie),
2. Database ensures safe and feasible concurrent writing and maintains data consistency,
3. Database supports fast and reliable multi-condition retrieval.

Business rules and assumptions:

RULES:

1. Every user/movie/review has a unique user/movie/review ID.
2. Every movie entry contains basic information.
3. Every user must register before using interactive functions.
4. Users have the right to delete an account at any time.
5. Users can edit movie entry information.
6. Each movie must belong to a genre, and each genre can contain multiple movies.
7. A movie can have multiple actors, and an actor can participate in multiple movies.
8. A director can direct multiple movies, but each movie has only one main director.
9. Users can only keep one final recorded rating and review for each movie, which can be modified or deleted at any time.
10. Each comment must be associated with a unique, existing user and movie.
11. The platform must archive all ratings/comments/operations.
12. The platform needs to support filtering movies by multiple conditions.

ASSUMPTIONS:

1. The information of a movie may be updated. But the UPDATE function of movie information may be misused. So, the data should support updates and version tracking.
2. Some movie entry information may be incomplete at the beginning, and the system needs to allow NULL.
3. For multiple ratings/comments of the same movie by the same user, the system will only display the last submitted version.

Problems and possible solutions:

Law: If the platform does not provide account cancellation or comment deletion, it will violate the Personal Information Protection Law. My solution is to enable DELETE both for account and review.

Ethics: If this platform is designed with social functions, it may lead to cyber violence. My solution is to cut down the community and social functions of this platform.

Finance: If the database is not designed properly, the expansion of data volume and high concurrent access will make the platform invest more in hardware costs. My solution is to store pictures and media information externally when the database only saves references.

Requirements:

1. Allow users to create accounts, log in, and delete account.
2. Allow users to add, modify, and delete movie information.
3. Users can and can only keep one "currently effective" rating + comment for the same movie, which can be updated or deleted at any time.
4. Movies need to have multiple associated classes, such as genre, actor, director, etc.
5. Support users to filter movies by release years, rating ranges, etc.
6. Allow system to timestamp each record and change, the person and time of change.
7. Allow system to timestamp each review and score, and corresponding information.

Reflection:

When collecting requirements for our platform, I tried to consider the problem as comprehensively as possible. For example, I didn’t consider that users need to delete their accounts at first, but I found it in the analysis of the problems, and added it. Meanwhile, I asked the “why” behind each requirement. I found that users want the search function to efficiently find the movies they want to watch. Then, I thought. What kind of movies do users really want to watch? Through research on Douban’s search function and reflection on my own search habits, I introduced factors such as genre and release time to help users filter movies.

(word count: 699)

Q2:

Step 1:

According to the given scenario and my requirements analysis, I find the **User is a core entity**, who can register, log in, and manage their movie reviews. Then, **Movie is also an entity**, with basic information. **Genre, Actor, Director are entities associated with Movie. The actions of reviewing and rating mean that Review is an entity**, including attributes like score and comment. **To support tracing back , I introduce OperationLog and ReviewLog entities** to respectively record user actions and review snapshots. The whole list of entities and attributes is as follows:

User
  user_id
  name
  email
  password
  reg_ip
  is_deleted

Movie

```
 movie_id
 title
 release_year
 synopsis       #nullable
 media_url   #nullable
 genre_id     #FK to Genre
 director_id    #FK to Director
 actor_ids    #FK to Actor
 avg_score       #derived
 review_count  #derived
 is_deleted

Genre
 genre_id
 name

Director
 director_id
 name

Actor
 actor_id
 name

Review
 review_id
 user_id        #FK to User
 movie_id       #FK to Movie
 score
 content        #nullable
 created_at
 updated_at
 is_deleted

OperationLog
 olog_id
 user_id         #FK to User
 movie_id       #FK to Movie
 action
 op_time

ReviewLog
 rlog_id
 review_id    #FK to Review
```

user_id	#FK to User
movie_id	#FK to Movie
score	
content	
op_time	
op_ip	

Step 2:

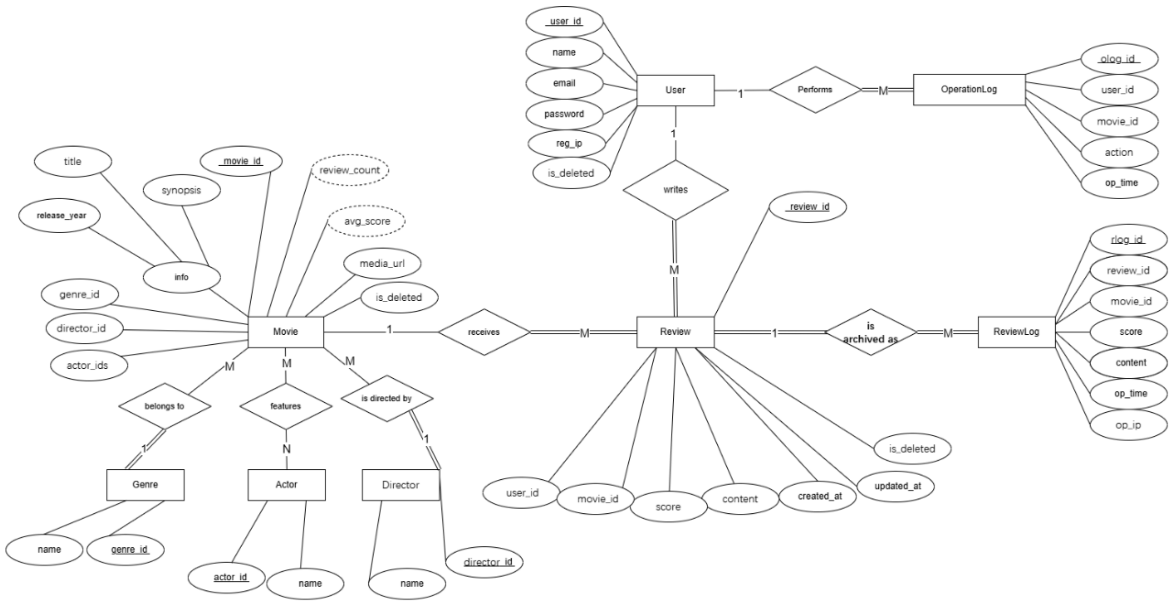
1. According to the rule that each comment must be associated with a unique, existing user and movie, the relationship between **User** and **Review** is a **writes** (1:M) relationship.
2. According to the rule that each comment must be associated with a unique, existing user and movie, the relationship between **Movie** and **Review** is **receives** (1:M).
3. According to the rule that a movie can feature multiple actors and an actor can act in multiple movies, the relationship between **Movie** and **Actor** is **features** (M:N).
4. According to the rule that a movie is directed by one director and a director can direct multiple movies, the relationship between **Movie** and **Director** is ***is directed by*** (M:1).
5. According to the rule that each movie belongs to one genre and one genre can include many movies, the relationship between **Movie** and **Genre** is **belongs to** (M:1).
6. According to the archive rule for operations, and one user can have several operations, the relationship between **User** and **OperationLog** is **performs** (1:M).
7. According to the archive rule for reviews, and one review can be updated several times, the relationship between **Review** and **ReviewLog** is ***is archived as*** (1:M).

Step3:

1. **Primary Key (PK)** of **User** is **user_id**;
2. PK of **Movie** is **movie_id.** It contains three foreign keys, **genre_id**, which connects the movie to a genre, **actor_id** to associate the movie with actors, **director_id**, which links the movie to one director;
3. PK of **Actor** is **actor_id;**
4. PK of **Genre** is **genre_id**;
5. PK of **Review** is **review_id**, Review owns two **Foreign Keys (FK**). One is **user_id**, which ensures that every review is written by only one user. The other is **movie_id**, which ensures that each review belongs to one specific movie.
6. PK of **OperationLog** is **olog_id**. It contains two FKs, **user_id and movie_id**, which links each log entry to one user and one movie, ensuring that all system operations are traceable to the correct user.
7. PK of **ReviewLog** is **rlog_id**. It includes three foreign keys: **review_id**, which connects the log entry to the original review; **user_id**, which records the author of the review; and **movie_id**, which associates the review log with its corresponding movie.

Step4:

Figure1. ERD (Next Page)



Assumptions and constraints:

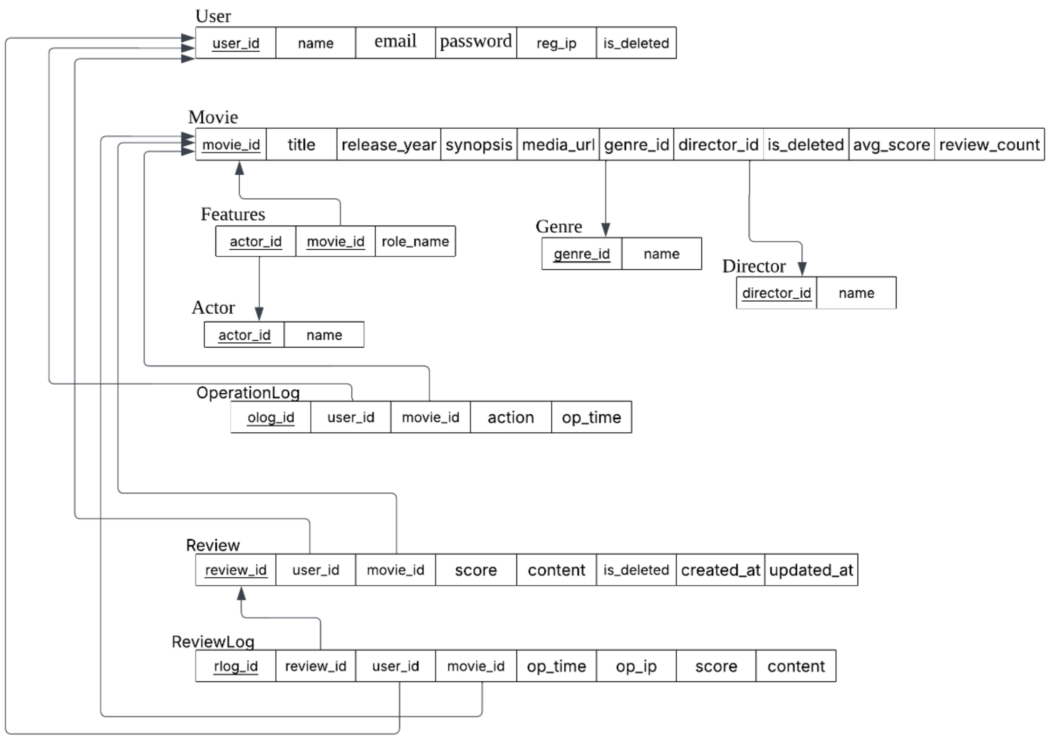
1. Movies can be created even if the cast or director has not been finalized, so actor_ids or director_id can be **temporarily null**.
2. All reviews are archived **at least once** (upon creation) and again whenever updated or deleted.
3. All operations are archived when conducted by a user.
4. Derived attributes like *avg_score* and *review_count* are **externally computed** and **will not be reflected in** the physical design of the movie class in the database.
5. The *Media_url* attribute in the Movie class points to **external storage** to prevent resource runs caused by data expansion.
6. Because a movie is strongly associated with a genre/director, so once a genre/director is defined, it cannot be deleted.
7. The many-to-many relationship between actors and movies requires a relational table in the coming physical model.
8. Although **a comment is mandatory to be bound to a valid user/movie**, **a user/movie does not necessarily need a comment.**

Reflection:

As I mentioned in the previous part of Q2, this ERD reflects the requirements and business rules/assumptions very well. During the whole process of answering Q2, I reviewed the business rules and assumptions I set before and f**ound some problems and made some changes.** For example, when I was drawing the ERD, I found that a review must be associated with one and only one valid movie and user. But I only mentioned that the review must be associated with the user. So, I adjusted this missing rule. Meanwhile, **to improve readability**, I set the *title, release_year* and *synopsis* under movie into the **parent attribute info**. I also studied the mutual mandatory relationships of each class and reflected them in the ERD.

Q3:

Figure 2. Logical Model



Analysis:

User: PK is user_id, which determines the other non-primary key. I assume that the registered email address is unique among users, but the username is not unique. So, email also determines the other non-primary keys as the candidate key. *is_deleted* defaults to FALSE, and *other attributes* are set to NOT NULL.

Movie: PK is movie_id, which determines the other non-primary key. I assume that *avg_score* and *review_count* are derived values, **logically determined by** *movie_id*, but **not stored in the physical table.** To maintain consistency with the ERD, I will **temporarily put it in the logical model**, but **it will not appear in the physical model and will not affect the 3NF of the model.** When a movie poster and other media files have not been uploaded, **media_url is set to NULL.** If it is confirmed that there is no such file, **it is set to "". The same is true for synopsis.** According to the rule that a movie must belong to one genre and one director, **genre_id and director_id are NOT NULL. But actor_ids can be NULL.**

Actor: PK is actor_id, which determines the other non-primary key. 'name' is NOT NULL.

Genre: PK is genre_id, which determines the other non-primary key. 'name' is NOT NULL.

Director: PK is director_id, which determines the other non-primary key. 'name' is NOT NULL.

Review": PK is review_id, which determines the other non-primary key. I assume that empty comments are allowed, so *content* can be set to "" or NULL, but empty ratings are not allowed, so *score* is set to NOT NULL. The timestamp *updated_at* is NULL if it is not modified and *is_deleted* defaults to FALSE.

ReviewLog: PK is rlog_id, which determines the other non-primary key. All other keys are set to NOT NULL to ensure complete audit logs. If the content of a review is empty, the content of this class can also be "" or NULL.

OperationLog: PK is olog_id, which determines the other non-primary key. All other keys are set to NOT NULL to ensure complete audit logs.
Features: As **a many-to-many association table between Movie and Actor**, the **composite primary key** (movie_id, actor_id) of Features also serves as a foreign key to reference the primary table, ensuring that one mapping record corresponds to only one movie and one actor. Its functional dependency is **(movie_id, actor_id) → role_name, which is completely dependent on the entire primary key** and complies with 3NF.

Further explanation:
As shown above, when converting ERD to logical model, I converted a many-to-many relationship between movie and actor into **intersection entity 'Features' and** used *'movie_id'* and *'actor_id'* as **both primary key and foreign key**, and the remaining attribute '*role_name'* in the table **is completely dependent on the entire primary key**. I ensure 3NF for it. Furthermore, for other entities, there are no repeating groups or arrays (1NF), and each entity has only one PK, eliminating partial dependencies (2NF). And as I mentioned before, all non-primary key attributes are directly and completely dependent on the table's primary key, without any transitive dependencies (3NF). Except for the intersection entity 'Features' relationship I have established, the remaining relationships in my model are **all 1:M** relationships **between different entities.** How I use foreign keys to **link primary keys** and implement relationships is shown in the figure.

Q4:

TABLES and Explanations (DATA type and SAMPLE):
For the primary key of all entities, **I use SERIAL PRIMARY KEY because it can be easily auto increased.**

Table1: USERS:

Column Name	USER_ID	EMAIL	*PASSWORD*	REG_IP	IS_DELETED
Key Type	PK				
Not Null = NN Unique = U	NN, U	NN, U	NN	NN	
DATA Type	**SERIAL**	*VARCHAR*	*VARCHAR*	*VARCHAR*	BOOLEAN
Length	/	200	255	45	/

User:
email VARCHAR(200) NOT NULL UNIQUE ensures that each user registers with **a unique** email address;
password VARCHAR(255) NOT NULL to **accept strong encryption passwords** provided by some companies (such as Apple);
reg_ip VARCHAR(45) NOT NULL supports the **maximum length of IPv6**;
is_deleted BOOLEAN DEFAULT FALSE is used for **soft deletion**.
I inserted **normal and soft deleted users** into the sample data to verify the system's query and filtering of different account statuses.

Movie:

Table2: MOVIES:

Column Name	MOVIE_ID	TITLE	*RELEASE_YEAR*	SYNOPSIS	MEDIA_URL	GENRE_ID	DIRECTOR_ID	IS_DELETED
Key Type	PK					FK	FK	
Not Null = NN Unique = U	NN, U	NN, U	NN			NN	NN	
DATA Type	**SERIAL**	*VARCHAR*	*SMALLINT*	*TEXT*	*VARCHAR*	*INTEGER*	*INTEGER*	BOOLEAN
Length	/	200	/	/	500	/	/	/

title VARCHAR(200) NOT NULL to **avoid long titles** that cannot be stored;
release_year SMALLINT NOT NULL CHECK (release_year BETWEEN 1888 AND EXTRACT(YEAR FROM CURRENT_DATE)+1) to **ensure the legality of the release year**;
synopsis TEXT supports **variable length storage**,
media_url VARCHAR(500) limits the maximum length to **avoid malicious links**;
genre_id INTEGER NOT NULL REFERENCES genre(genre_id) and *director_id INTEGER NOT NULL REFERENCES director(director_id)* force each movie to be **associated with a genre and a main director**;
is_deleted BOOLEAN DEFAULT FALSE to **achieve soft deletion**;
avg_score and *review_count* are derived attributes that are calculated later and are **not physically stored in the main table**.

I introduced **several movies of the same genre/director** into the table to help simulate conditional filtering. I also included both **fully filled entries** (such as "SciFi Galaxy") and **partially missing entries** (such as "13th" and "Speed Chase" with media_url to NULL) to simulate the incomplete media resources in the early stage of the launch. I also designed a **soft deletion** for a movie.

Genre:

Table3: GENRES:

Column Name	GENRE_ID	NAME
Key Type	PK	
Not Null = NN Unique = U	NN, U	NN, U
DATA Type	**SERIAL**	*VARCHAR*
Length	/	100

name VARCHAR(100) NOT NULL UNIQUE realizes **name uniqueness.**
The sample data covers 15 common types.

Director:

Table4: DIRECTORS:

Column Name	DIRECTOR_ID	NAME
Key Type	PK	
Not Null = NN Unique = U	NN, U	NN, U
DATA Type	**SERIAL**	*VARCHAR*
Length	/	150

name VARCHAR(150) NOT NULL meets the **name required.**

The sample data covers 15 famous director.

Actor:

Table5: ACTORS:

Column Name	ACTOR_ID	NAME
Key Type	PK	
Not Null = NN Unique = U	NN, U	NN, U
DATA Type	**SERIAL**	*VARCHAR*
Length	/	150

name VARCHAR(150) NOT NULL meets the **name required.**

The sample data covers 15 famous actor.

Review:

Table6: REVIEWS:

Column Name	REVIEW_ID	USER_ID	*MOVIE_ID*	SCORE	CONTENT	*CREATED_AT*	*UPDATED_AT*	IS_DELETED
Key Type	PK	FK	FK					
Not Null = NN Unique = U	NN, U	NN	NN	NN		NN		
DATA Type	**SERIAL**	*INTEGER*	*INTEGER*	*SMALLINT*	*VARCHAR*	*TIMESTAMP*	*TIMESTAMP*	BOOLEAN
Length	/	/	/	/	500	/	/	/

INTEGER NOT NULL REFERENCES "user"(user_id) and *INTEGER NOT NULL REFERENCES movie(movie_id)* to ensure that movie reviews must be associated with **valid users and movies**;
score SMALLINT NOT NULL CHECK (score BETWEEN 0 AND 10) enforces the **score range**;
content VARCHAR(500) allows **empty strings or NULL to simulate 'no comments'**;
created_at TIMESTAMP NOT NULL and *updated_at TIMESTAMP* capture the f**irst and edit times**;
is_deleted BOOLEAN DEFAULT FALSE handles **soft deletion;**

The review table contains both records that **have not been modified** since the initial creation and **updated entries** with updated_at timestamps, and the soft deletion is simulated through the is_deleted flag.

ReviewLog:

Table7: REVIEWLOGS:

Column Name	RLOG_ID	REVIEW_ID	USER_ID	*MOVIE_ID*	SCORE	CONTENT	*OP_TIME*	*OP_IP*
Key Type	PK	FK	FK	FK				
Not Null = NN Unique = U	NN, U	NN	NN	NN	NN		NN	NN
DATA Type	**SERIAL**	*INTEGER*	*INTEGER*	*INTEGER*	*SMALLINT*	*VARCHAR*	*TIMESTAMP*	*VARCHAR*
Length	/	/	/	/	/	500	/	45

review_id INTEGER NOT NULL REFERENCES review(review_id) etc. ensure that each log can accurately **map the review, user and movie;**

score SMALLINT NOT NULL CHECK (score BETWEEN 0 AND 10) and *content VARCHAR(500)* are consistent with the review table;

op_time TIMESTAMP NOT NULL and *op_ip VARCHAR(45) NOT NULL* record the time and IP;

For each review with an updated record, I introduced the **original rating and review** before it was updated (at creation time) in ReviewLog, and **kept the user and movie unchanged** to simulate the real update situation.

OperationLog:

Table8: OPERATIONLOGS:

Column Name	OLOG_ID	USER_ID	*MOVIE_ID*	ACTION	*OP_TIME*
Key Type	PK	FK	FK		
Not Null = NN Unique = U	NN, U	NN	NN		NN
DATA Type	**SERIAL**	*INTEGER*	*INTEGER*	*VARCHAR*	*TIMESTAMP*
Length	/	/	/	50	/

user_id INTEGER NOT NULL REFERENCES "user"(user_id) etc. ensure that each log associates the initiator of the operation with the affected movie;

action VARCHAR(50) NOT NULL describes the operation type;

op_time TIMESTAMP NOT NULL records the operation time;

I simulated multiple **CREATE and UPDATE** operations in OperationLog, and I also set a user **DELETE** operation in the log **for a movie marked as soft deleted in the movie class** to simulate reality.

Features:

Table9: FEATURES:

Column Name	MOVIE _ID	ACTOR _ID	ROLE_ NAME
Key Type	PK, FK	PK, FK	
Not Null = NN Unique = U	NN	NN	
DATA Type	*INTEGER*	*INTEGER*	*VARCHAR*
Length	/	/	100

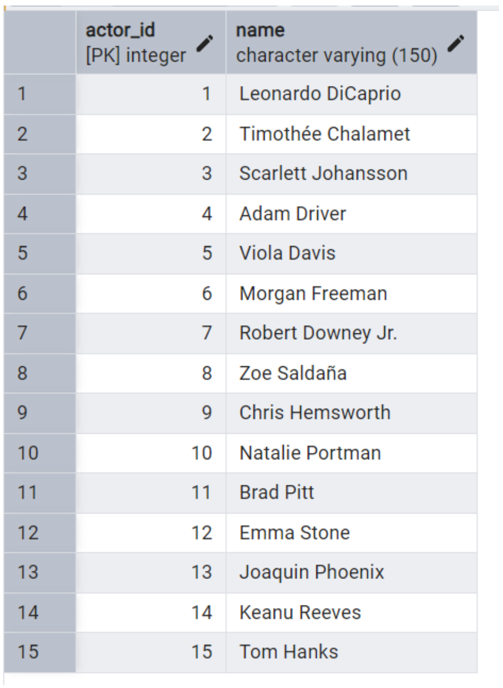
As a many-to-many association between Movie and Actor, the Features table uses a composite primary key (movie_id, actor_id) to achieve a unique combination; both are defined as I*NTEGER NOT NULL REFERENCES...* to **ensure the integrity of the relationship**; *role_name VARCHAR(100)* allows NULL.

For the optional association between movies and actors, I set **several role_names to NULL** to simulate the lack of role information.

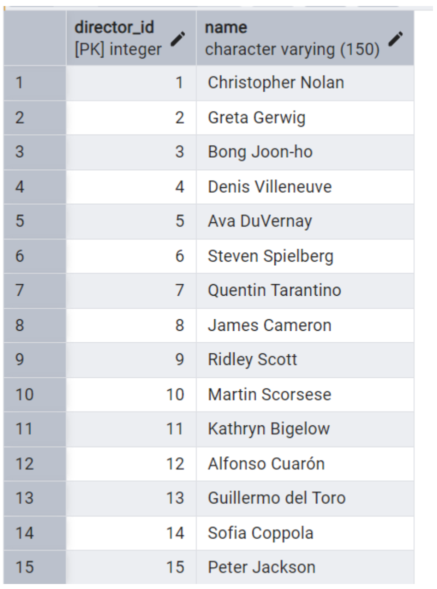
Reflection:

1. To support the search function, I reserved avg_score and review_count attributes during the initial ERD design, but I couldn't determine their physical storage form, and I was always worried that it would damage 3NF of movie. For this reason, I first searched the official PostgreSQL document and realized that these are derived data and could be processed later. Later, after learning about SQL queries in class, I understood how to calculate these values in the database.
2. I knew nothing when choosing proper length. Therefore, I searched the official PostgreSQL document and analyzed the sample data. For example, I fetched the title lengths of the top 100 popular movies on IMDb and found that their lengths ranged from 3 to 89 characters, with the 95th percentile being about 60 characters. But when I further searched, I found that some movie titles in certain languages have very large amounts of characters. For example, I found a movie with a title of 161 characters. So I finally chose the movie title as VARCHAR(200).

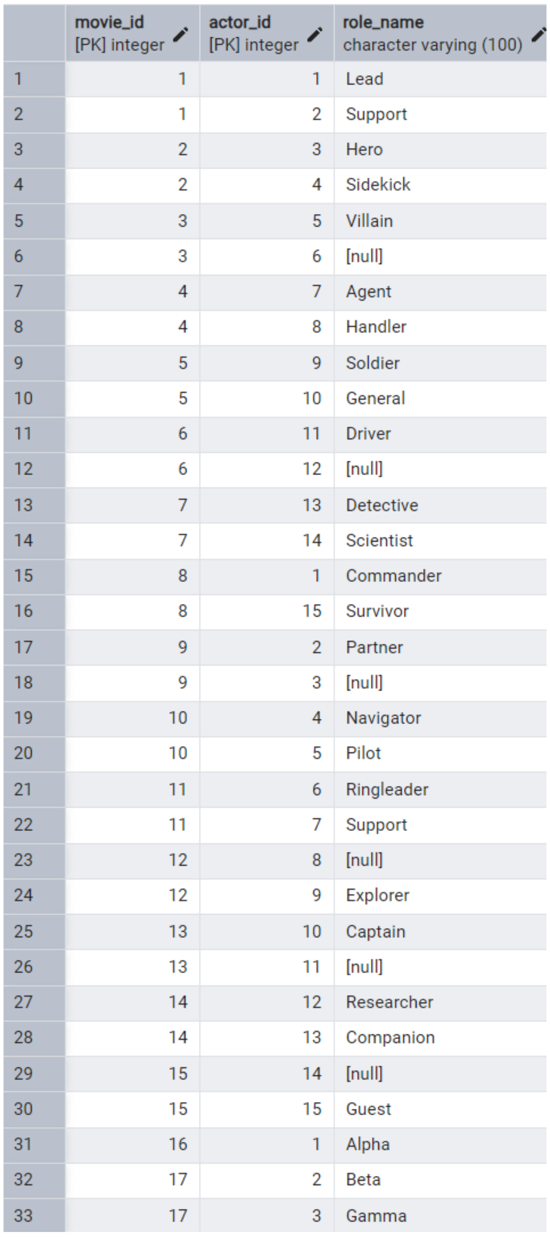
Tables created in the database:



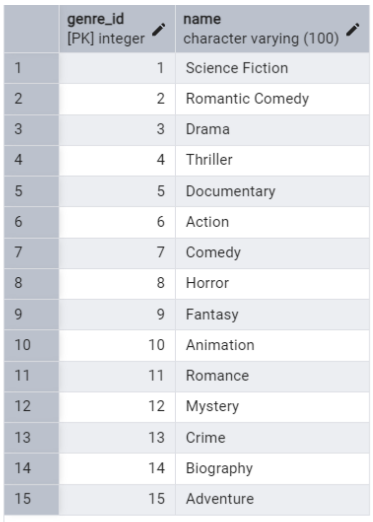
	actor_id [PK] integer	name character varying (150)
1	1	Leonardo DiCaprio
2	2	Timothée Chalamet
3	3	Scarlett Johansson
4	4	Adam Driver
5	5	Viola Davis
6	6	Morgan Freeman
7	7	Robert Downey Jr.
8	8	Zoe Saldaña
9	9	Chris Hemsworth
10	10	Natalie Portman
11	11	Brad Pitt
12	12	Emma Stone
13	13	Joaquin Phoenix
14	14	Keanu Reeves
15	15	Tom Hanks



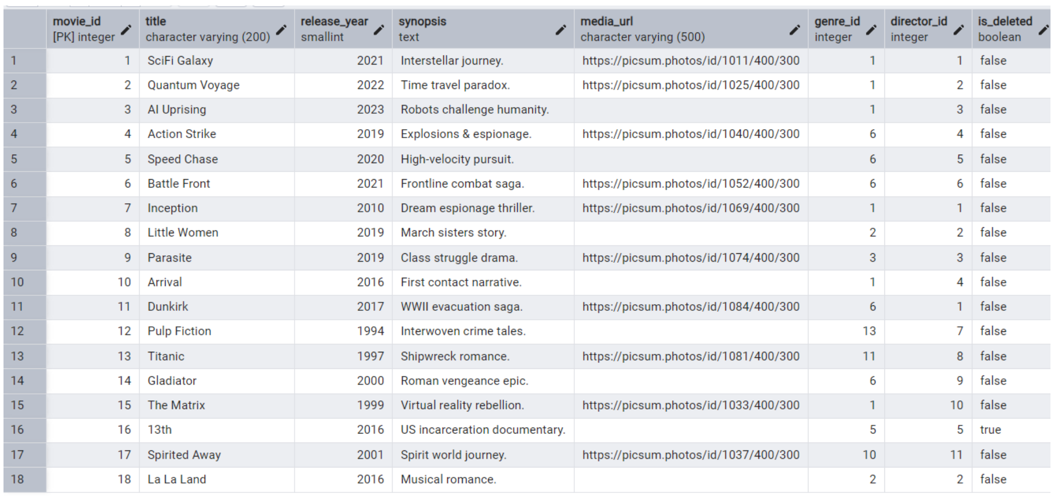
	director_id [PK] integer	name character varying (150)
1	1	Christopher Nolan
2	2	Greta Gerwig
3	3	Bong Joon-ho
4	4	Denis Villeneuve
5	5	Ava DuVernay
6	6	Steven Spielberg
7	7	Quentin Tarantino
8	8	James Cameron
9	9	Ridley Scott
10	10	Martin Scorsese
11	11	Kathryn Bigelow
12	12	Alfonso Cuarón
13	13	Guillermo del Toro
14	14	Sofia Coppola
15	15	Peter Jackson



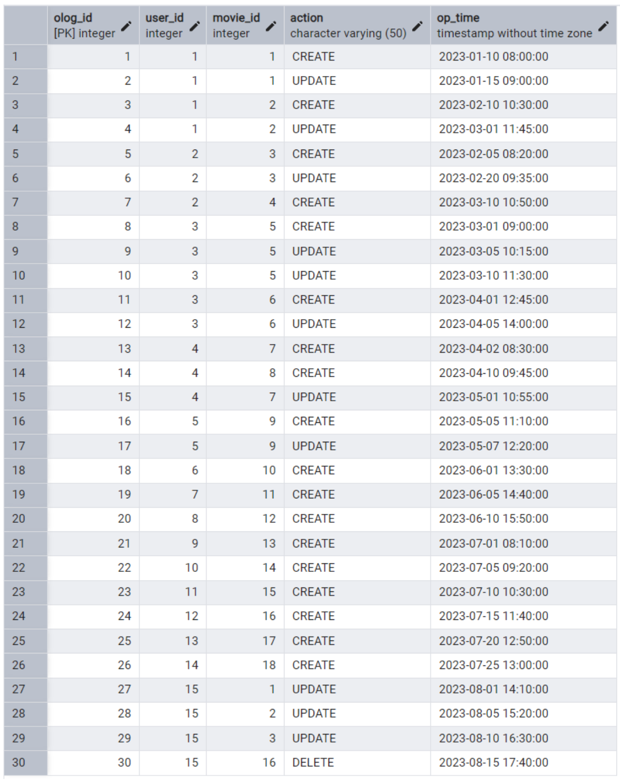
	movie_id [PK] integer	actor_id [PK] integer	role_name character varying (100)
1	1	1	Lead
2	1	2	Support
3	2	3	Hero
4	2	4	Sidekick
5	3	5	Villain
6	3	6	[null]
7	4	7	Agent
8	4	8	Handler
9	5	9	Soldier
10	5	10	General
11	6	11	Driver
12	6	12	[null]
13	7	13	Detective
14	7	14	Scientist
15	8	1	Commander
16	8	15	Survivor
17	9	2	Partner
18	9	3	[null]
19	10	4	Navigator
20	10	5	Pilot
21	11	6	Ringleader
22	11	7	Support
23	12	8	[null]
24	12	9	Explorer
25	13	10	Captain
26	13	11	[null]
27	14	12	Researcher
28	14	13	Companion
29	15	14	[null]
30	15	15	Guest
31	16	1	Alpha
32	17	2	Beta
33	17	3	Gamma



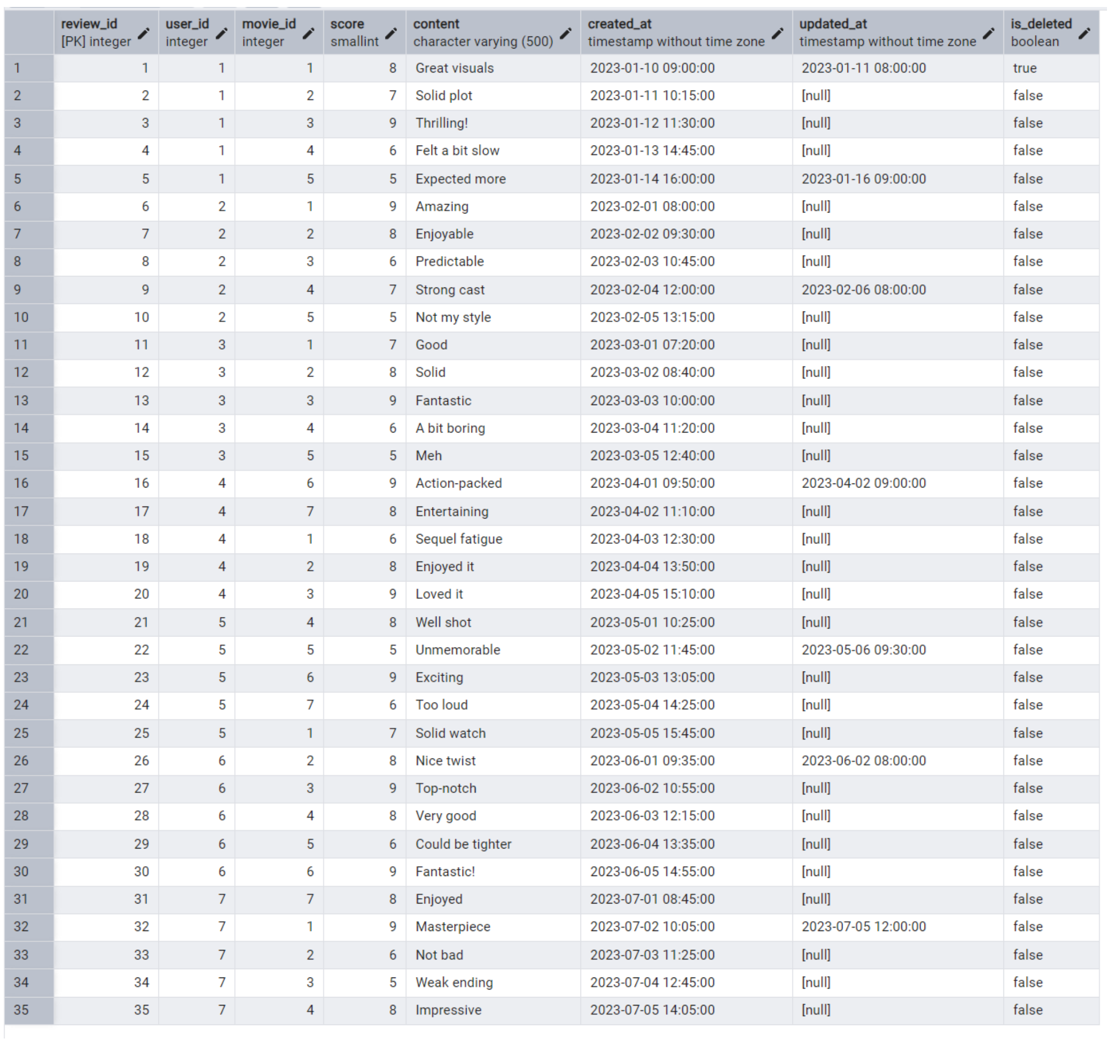
	genre_id [PK] integer	name character varying (100)
1	1	Science Fiction
2	2	Romantic Comedy
3	3	Drama
4	4	Thriller
5	5	Documentary
6	6	Action
7	7	Comedy
8	8	Horror
9	9	Fantasy
10	10	Animation
11	11	Romance
12	12	Mystery
13	13	Crime
14	14	Biography
15	15	Adventure



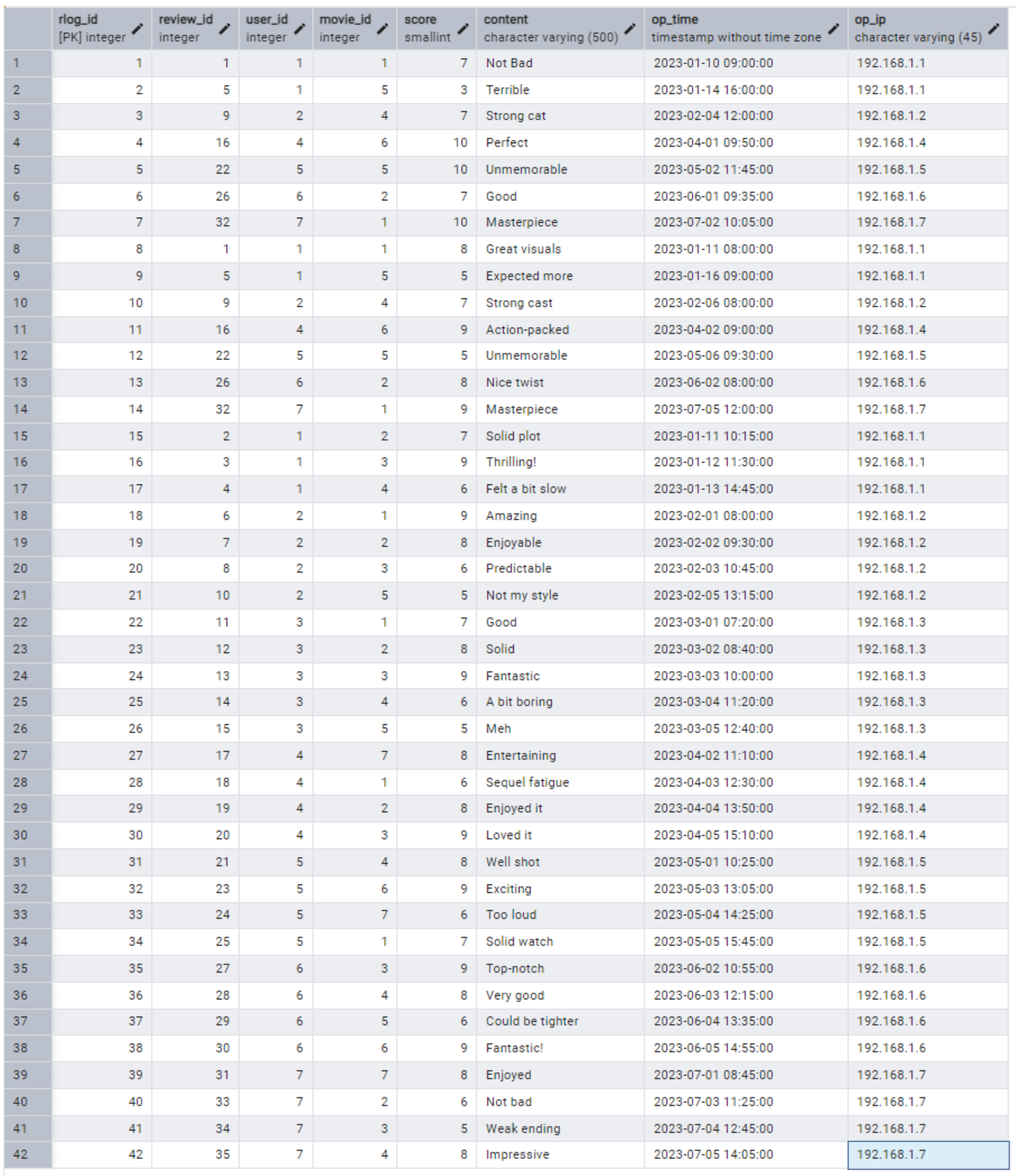
	movie_id [PK] integer	title character varying (200)	release_year smallint	synopsis text	media_url character varying (500)	genre_id integer	director_id integer	is_deleted boolean
1	1	SciFi Galaxy	2021	Interstellar journey.	https://picsum.photos/id/1011/400/300	1	1	false
2	2	Quantum Voyage	2022	Time travel paradox.	https://picsum.photos/id/1025/400/300	1	2	false
3	3	AI Uprising	2023	Robots challenge humanity.		1	3	false
4	4	Action Strike	2019	Explosions & espionage.	https://picsum.photos/id/1040/400/300	6	4	false
5	5	Speed Chase	2020	High-velocity pursuit.		6	5	false
6	6	Battle Front	2021	Frontline combat saga.	https://picsum.photos/id/1052/400/300	6	6	false
7	7	Inception	2010	Dream espionage thriller.	https://picsum.photos/id/1069/400/300	1	1	false
8	8	Little Women	2019	March sisters story.		2	2	false
9	9	Parasite	2019	Class struggle drama.	https://picsum.photos/id/1074/400/300	3	3	false
10	10	Arrival	2016	First contact narrative.		1	4	false
11	11	Dunkirk	2017	WWII evacuation saga.	https://picsum.photos/id/1084/400/300	6	1	false
12	12	Pulp Fiction	1994	Interwoven crime tales.		13	7	false
13	13	Titanic	1997	Shipwreck romance.	https://picsum.photos/id/1081/400/300	11	8	false
14	14	Gladiator	2000	Roman vengeance epic.		6	9	false
15	15	The Matrix	1999	Virtual reality rebellion.	https://picsum.photos/id/1033/400/300	1	10	false
16	16	13th	2016	US incarceration documentary.		5	5	true
17	17	Spirited Away	2001	Spirit world journey.	https://picsum.photos/id/1037/400/300	10	11	false
18	18	La La Land	2016	Musical romance.		2	2	false



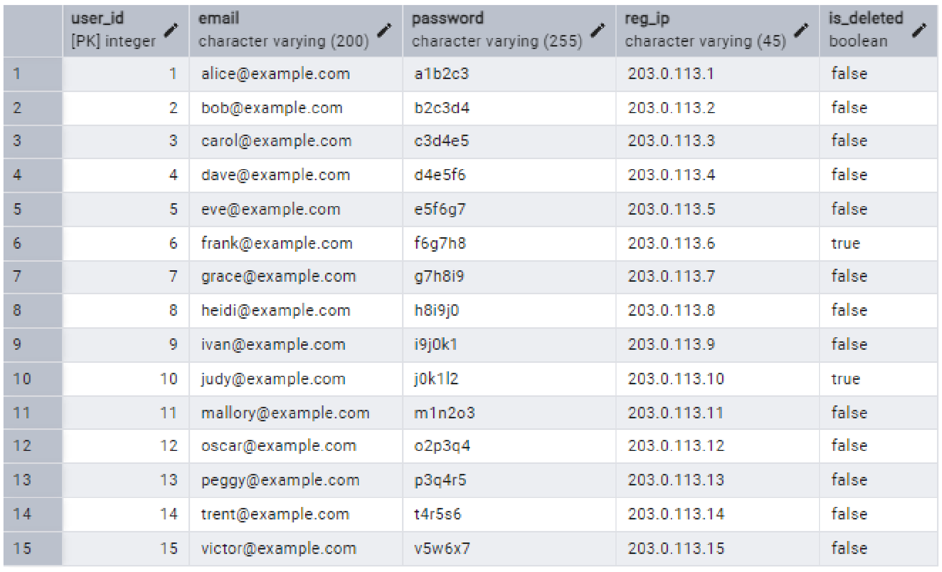
	olog_id [PK] integer	user_id integer	movie_id integer	action character varying (50)	op_time timestamp without time zone
1	1	1	1	CREATE	2023-01-10 08:00:00
2	2	1	1	UPDATE	2023-01-15 09:00:00
3	3	1	2	CREATE	2023-02-10 10:30:00
4	4	1	2	UPDATE	2023-03-01 11:45:00
5	5	2	3	CREATE	2023-02-05 08:20:00
6	6	2	3	UPDATE	2023-02-20 09:35:00
7	7	2	4	CREATE	2023-03-10 10:50:00
8	8	3	5	CREATE	2023-03-01 09:00:00
9	9	3	5	UPDATE	2023-03-05 10:15:00
10	10	3	5	UPDATE	2023-03-10 11:30:00
11	11	3	6	CREATE	2023-04-01 12:45:00
12	12	3	6	UPDATE	2023-04-05 14:00:00
13	13	4	7	CREATE	2023-04-02 08:30:00
14	14	4	8	CREATE	2023-04-10 09:45:00
15	15	4	7	UPDATE	2023-05-01 10:55:00
16	16	5	9	CREATE	2023-05-05 11:10:00
17	17	5	9	UPDATE	2023-05-07 12:20:00
18	18	6	10	CREATE	2023-06-01 13:30:00
19	19	7	11	CREATE	2023-06-05 14:40:00
20	20	8	12	CREATE	2023-06-10 15:50:00
21	21	9	13	CREATE	2023-07-01 08:10:00
22	22	10	14	CREATE	2023-07-05 09:20:00
23	23	11	15	CREATE	2023-07-10 10:30:00
24	24	12	16	CREATE	2023-07-15 11:40:00
25	25	13	17	CREATE	2023-07-20 12:50:00
26	26	14	18	CREATE	2023-07-25 13:00:00
27	27	15	1	UPDATE	2023-08-01 14:10:00
28	28	15	2	UPDATE	2023-08-05 15:20:00
29	29	15	3	UPDATE	2023-08-10 16:30:00
30	30	15	16	DELETE	2023-08-15 17:40:00



	review_id [PK] integer	user_id integer	movie_id integer	score smallint	content character varying (500)	created_at timestamp without time zone	updated_at timestamp without time zone	is_deleted boolean
1	1	1	1	8	Great visuals	2023-01-10 09:00:00	2023-01-11 08:00:00	true
2	2	1	2	7	Solid plot	2023-01-11 10:15:00	[null]	false
3	3	1	3	9	Thrilling!	2023-01-12 11:30:00	[null]	false
4	4	1	4	6	Felt a bit slow	2023-01-13 14:45:00	[null]	false
5	5	1	5	5	Expected more	2023-01-14 16:00:00	2023-01-16 09:00:00	false
6	6	2	1	9	Amazing	2023-02-01 08:00:00	[null]	false
7	7	2	2	8	Enjoyable	2023-02-02 09:30:00	[null]	false
8	8	2	3	6	Predictable	2023-02-03 10:45:00	[null]	false
9	9	2	4	7	Strong cast	2023-02-04 12:00:00	2023-02-06 08:00:00	false
10	10	2	5	5	Not my style	2023-02-05 13:15:00	[null]	false
11	11	3	1	7	Good	2023-03-01 07:20:00	[null]	false
12	12	3	2	8	Solid	2023-03-02 08:40:00	[null]	false
13	13	3	3	9	Fantastic	2023-03-03 10:00:00	[null]	false
14	14	3	4	6	A bit boring	2023-03-04 11:20:00	[null]	false
15	15	3	5	5	Meh	2023-03-05 12:40:00	[null]	false
16	16	4	6	9	Action-packed	2023-04-01 09:50:00	2023-04-02 09:00:00	false
17	17	4	7	8	Entertaining	2023-04-02 11:10:00	[null]	false
18	18	4	1	6	Sequel fatigue	2023-04-03 12:30:00	[null]	false
19	19	4	2	8	Enjoyed it	2023-04-04 13:50:00	[null]	false
20	20	4	3	9	Loved it	2023-04-05 15:10:00	[null]	false
21	21	5	4	8	Well shot	2023-05-01 10:25:00	[null]	false
22	22	5	5	5	Unmemorable	2023-05-02 11:45:00	2023-05-06 09:30:00	false
23	23	5	6	9	Exciting	2023-05-03 13:05:00	[null]	false
24	24	5	7	6	Too loud	2023-05-04 14:25:00	[null]	false
25	25	5	1	7	Solid watch	2023-05-05 15:45:00	[null]	false
26	26	6	2	8	Nice twist	2023-06-01 09:35:00	2023-06-02 08:00:00	false
27	27	6	3	9	Top-notch	2023-06-02 10:55:00	[null]	false
28	28	6	4	8	Very good	2023-06-03 12:15:00	[null]	false
29	29	6	5	6	Could be tighter	2023-06-04 13:35:00	[null]	false
30	30	6	6	9	Fantastic!	2023-06-05 14:55:00	[null]	false
31	31	7	7	8	Enjoyed	2023-07-01 08:45:00	[null]	false
32	32	7	1	9	Masterpiece	2023-07-02 10:05:00	2023-07-05 12:00:00	false
33	33	7	2	6	Not bad	2023-07-03 11:25:00	[null]	false
34	34	7	3	5	Weak ending	2023-07-04 12:45:00	[null]	false
35	35	7	4	8	Impressive	2023-07-05 14:05:00	[null]	false



	rlog_id [PK] integer	review_id integer	user_id integer	movie_id integer	score smallint	content character varying (500)	op_time timestamp without time zone	op_ip character varying (45)
1	1	1	1	1	7	Not Bad	2023-01-10 09:00:00	192.168.1.1
2	2	5	1	5	3	Terrible	2023-01-14 16:00:00	192.168.1.1
3	3	9	2	4	7	Strong cat	2023-02-04 12:00:00	192.168.1.2
4	4	16	4	6	10	Perfect	2023-04-01 09:50:00	192.168.1.4
5	5	22	5	5	10	Unmemorable	2023-05-02 11:45:00	192.168.1.5
6	6	26	6	2	7	Good	2023-06-01 09:35:00	192.168.1.6
7	7	32	7	1	10	Masterpiece	2023-07-02 10:05:00	192.168.1.7
8	8	1	1	1	8	Great visuals	2023-01-11 08:00:00	192.168.1.1
9	9	5	1	5	5	Expected more	2023-01-16 09:00:00	192.168.1.1
10	10	9	2	4	7	Strong cast	2023-02-06 08:00:00	192.168.1.2
11	11	16	4	6	9	Action-packed	2023-04-02 09:00:00	192.168.1.4
12	12	22	5	5	5	Unmemorable	2023-05-06 09:30:00	192.168.1.5
13	13	26	6	2	8	Nice twist	2023-06-02 08:00:00	192.168.1.6
14	14	32	7	1	9	Masterpiece	2023-07-05 12:00:00	192.168.1.7
15	15	2	1	2	7	Solid plot	2023-01-11 10:15:00	192.168.1.1
16	16	3	1	3	9	Thrilling!	2023-01-12 11:30:00	192.168.1.1
17	17	4	1	4	6	Felt a bit slow	2023-01-13 14:45:00	192.168.1.1
18	18	6	2	1	9	Amazing	2023-02-01 08:00:00	192.168.1.2
19	19	7	2	2	8	Enjoyable	2023-02-02 09:30:00	192.168.1.2
20	20	8	2	3	6	Predictable	2023-02-03 10:45:00	192.168.1.2
21	21	10	2	5	5	Not my style	2023-02-05 13:15:00	192.168.1.2
22	22	11	3	1	7	Good	2023-03-01 07:20:00	192.168.1.3
23	23	12	3	2	8	Solid	2023-03-02 08:40:00	192.168.1.3
24	24	13	3	3	9	Fantastic	2023-03-03 10:00:00	192.168.1.3
25	25	14	3	4	6	A bit boring	2023-03-04 11:20:00	192.168.1.3
26	26	15	3	5	5	Meh	2023-03-05 12:40:00	192.168.1.3
27	27	17	4	7	8	Entertaining	2023-04-02 11:10:00	192.168.1.4
28	28	18	4	1	6	Sequel fatigue	2023-04-03 12:30:00	192.168.1.4
29	29	19	4	2	8	Enjoyed it	2023-04-04 13:50:00	192.168.1.4
30	30	20	4	3	9	Loved it	2023-04-05 15:10:00	192.168.1.4
31	31	21	5	4	8	Well shot	2023-05-01 10:25:00	192.168.1.5
32	32	23	5	6	9	Exciting	2023-05-03 13:05:00	192.168.1.5
33	33	24	5	7	6	Too loud	2023-05-04 14:25:00	192.168.1.5
34	34	25	5	1	7	Solid watch	2023-05-05 15:45:00	192.168.1.5
35	35	27	6	3	9	Top-notch	2023-06-02 10:55:00	192.168.1.6
36	36	28	6	4	8	Very good	2023-06-03 12:15:00	192.168.1.6
37	37	29	6	5	6	Could be tighter	2023-06-04 13:35:00	192.168.1.6
38	38	30	6	6	9	Fantastic!	2023-06-05 14:55:00	192.168.1.6
39	39	31	7	7	8	Enjoyed	2023-07-01 08:45:00	192.168.1.7
40	40	33	7	2	6	Not bad	2023-07-03 11:25:00	192.168.1.7
41	41	34	7	3	5	Weak ending	2023-07-04 12:45:00	192.168.1.7
42	42	35	7	4	8	Impressive	2023-07-05 14:05:00	192.168.1.7



	user_id [PK] integer	email character varying (200)	password character varying (255)	reg_ip character varying (45)	is_deleted boolean
1	1	alice@example.com	a1b2c3	203.0.113.1	false
2	2	bob@example.com	b2c3d4	203.0.113.2	false
3	3	carol@example.com	c3d4e5	203.0.113.3	false
4	4	dave@example.com	d4e5f6	203.0.113.4	false
5	5	eve@example.com	e5f6g7	203.0.113.5	false
6	6	frank@example.com	f6g7h8	203.0.113.6	true
7	7	grace@example.com	g7h8i9	203.0.113.7	false
8	8	heidi@example.com	h8i9j0	203.0.113.8	false
9	9	ivan@example.com	i9j0k1	203.0.113.9	false
10	10	judy@example.com	j0k1l2	203.0.113.10	true
11	11	mallory@example.com	m1n2o3	203.0.113.11	false
12	12	oscar@example.com	o2p3q4	203.0.113.12	false
13	13	peggy@example.com	p3q4r5	203.0.113.13	false
14	14	trent@example.com	t4r5s6	203.0.113.14	false
15	15	victor@example.com	v5w6x7	203.0.113.15	false

Q5:

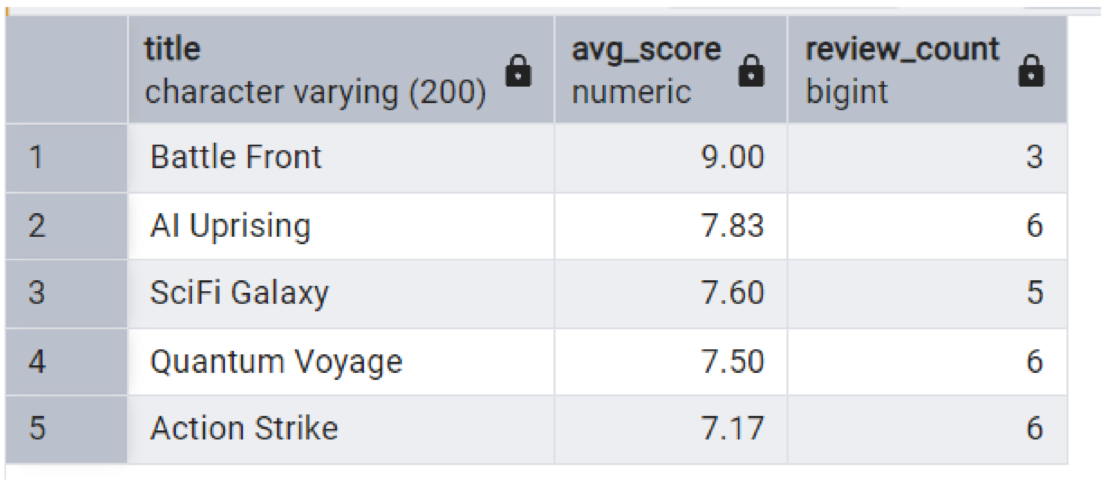
Query 1: Films Released After 2018 with Strong Audience Reception

This query identifies movies **released after 2018** whose **average review score is larger than 7** and which have received **more than one active review**.

This combines the movie and review tables using a JOIN; filters out logically deleted records and restricts to the desired release date via the WHERE clause; groups reviews per film using GROUP BY; and applies a HAVING to filter the average-score and review-count condition.

This simulates the user's searching movies based on conditions (year to release, score, etc.)

Data Output:



	title character varying (200)	avg_score numeric	review_count bigint
1	Battle Front	9.00	3
2	AI Uprising	7.83	6
3	SciFi Galaxy	7.60	5
4	Quantum Voyage	7.50	6
5	Action Strike	7.17	6

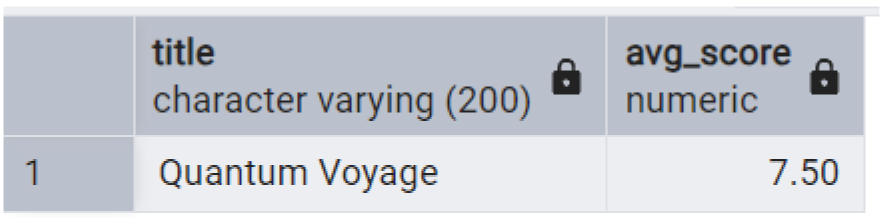
Query 2: Science Fiction Films by Greta Gerwig with High Average Score

This query identifies all **Science Fiction** movies directed by **Greta Gerwig** whose **average review score is larger than 7**.

This combines the movie and review tables using a JOIN; filters out logically deleted records and restricts to the **Sci-Fi** genre and specific director via **WHERE**; groups the scores per film using **GROUP BY**; and applies a **HAVING** to filter the average-score.

This simulates the user's searching movies based on conditions (genre, director, etc.)

Data Output:



	title character varying (200)	avg_score numeric
1	Quantum Voyage	7.50

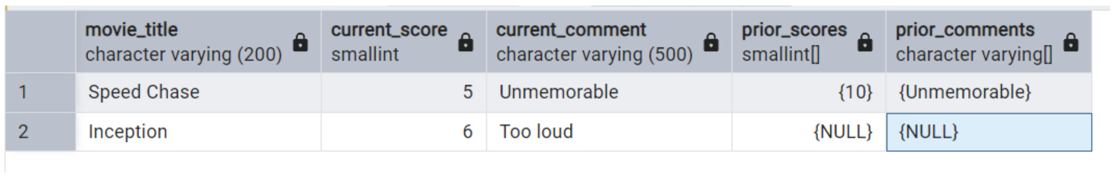
Query 3: May 2023 Low Ratings by "eve@example.com" with Historical Versions

This query identifies all reviews (rating **below 7**) submitted by user using email **eve@example.com** during **May 2023** on existing movies, and the review's historical version.

This combines the **review**, **user**, **movie**, and **reviewlog** tables using JOINs; filters out logically deleted records, restricts to the specified email, date range, and score via WHERE; uses a LEFT JOIN (with rl.op_time < r.updated_at) to **fetch historical reviews**; groups scores and comments using GROUP BY.

This simulates the user seeing both their current and historical feedback based on some conditions.

Data Output:



	movie_title character varying (200)	current_score smallint	current_comment character varying (500)	prior_scores smallint[]	prior_comments character varying[]
1	Speed Chase	5	Unmemorable	{10}	{Unmemorable}
2	Inception	6	Too loud	{NULL}	{NULL}

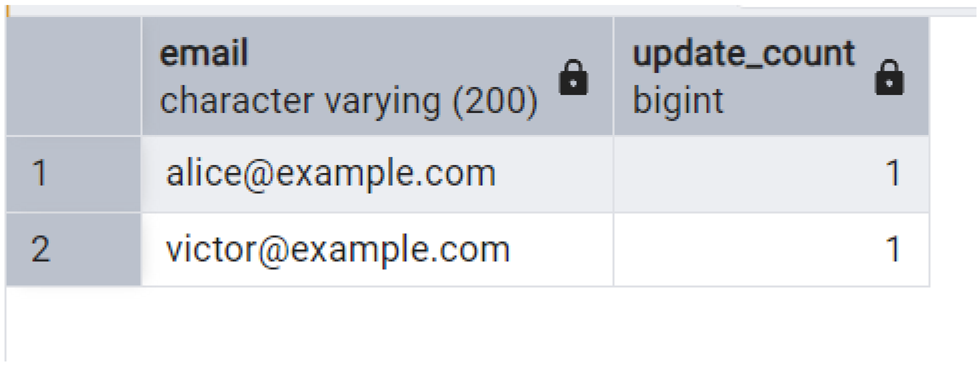
Query 4: Active Users Who Updated "SciFi Galaxy" in 2023

This query returns all users who conducted **UPDATE** operations on the movie **"SciFi Galaxy"** between **2023-01-01 00:00** and **2023-12-31 23:59**, along with the count of their updates.

This combines the movie, user and operationlog tables using a JOIN; restricts to the **'SciFi Galaxy'** movie, '**UPDATE**' operation, **BETWEEN 2023-01-01 00:00 AND 2023-12-31 23:59'** time range via **WHERE**; groups entries per user email using **GROUP BY**.

This simulates the tracing of user behavior (even if deleted of the interactive page)

Data Output:



	email character varying (200)	update_count bigint
1	alice@example.com	1
2	victor@example.com	1

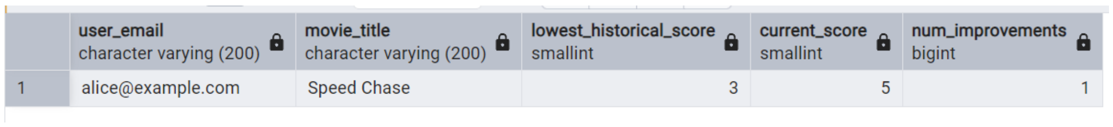
Query 5: Identify Users with Historical Ratings Below Their Current Score

This query identifies all **active** users **posting at least one review** for an **existing** movie and whose **historical** review score (from the audit log) was **lower** than their current score for the same film, then **groups** by user and film to count how many times they improved their rating.

This combines the reviewlog, review and user tables, as well as reviews and movie tables using JOINs; filters out logically deleted records and restricts to only those log entries where the old score is less than the current one via **WHERE**; groups each user–film combination using **GROUP BY**.

This simulates the platform's statistics on users' change between movie ratings.

Data Output:



	user_email character varying (200)	movie_title character varying (200)	lowest_historical_score smallint	current_score smallint	num_improvements bigint
1	alice@example.com	Speed Chase	3	5	1

Reflection:

During this part, I found that statistics require a large amount of samples, and the samples I created manually were very insufficient. I also discovered a new business need in the process: by counting the trend of changes in users' movie ratings, the platform can determine whether a movie may have a "second viewing" evaluation update.

Q6:

User Data:

When people register on our platform, we need to collect their **email addresses** to serve as a unique login credential. We also collect **usernames, passwords** as login credentials **and IP addresses** to locate account.

When they upload a review, the system will collect their **review content and scores** to support the platform and archive in the reviewlog**.** We will also record their **ip address** and **make a timestamp** to help trace back.

When they edit the entry information, the system will record their **operation type** and **make a timestamp** to help trace back.

Data storage and Privacy problems:

1. The physical location of the data center should be a climate-controlled, safe environment. We either carefully **select the location before setting up the server** or **choose a reliable cloud provider**.
2. We **monitor the IP addresses** of user operations to avoid attacks on the database caused by high-risk addresses.
3. We use **appropriate encryption algorithms** to encrypt and protect the database and back it up in time.
4. We should **monitor the input content part** (**especially the media_url in the movie entry**) in real time to avoid malicious input.
5. We regularly **conduct security audits** on reviewlog and operationlog to identify potential dangerous information.
6. The platform may have both Chinese users and international users. According to *China's Personal Information Protection Law*, Chinese user data must be stored domestically. My solution is to **separate the data** of Chinese users from that of overseas users based on the IP address at the time of registration. **Prioritize storing data in domestic data centers** and in future, **set up data center in the international user's area.**
7. According to the *China's Cybersecurity Law*, we will **archive user's ratings/comments/operations** and **retain relevant logs for no less than six months** to record the platform's status.
8. According to the *China's Personal Information Protection Law*, personal information processors should actively delete relevant personal information when the processing purpose has been achieved, or the legal retaining date has expired. Therefore, once the above requirements are met, **we will delete or irreversibly anonymize them.**
9. If the platform collects too much information beyond the necessary need, it will violate the minimum necessary principle of the *China's Personal Information Protection Law*. My solution is to **only collect information necessary for the service** (email, password and IP address).
10. If we do not anonymize users when making analyses based on their information, we may violate privacy regulations. My solution is to **replace private identifiers with pseudonym** and **remove personally identifiable information**.

(word count: 422)