Movie Social Network Database

Student Name: To Thao Nhi Trinh
Course: 31061 Database Systems

Real-World Domain: Movie Social Network

This database models a Letterboxd-inspired platform where movie enthusiasts can manage their film journey and connect with others.

Core Features:

Personal Film Tracking:

- Log viewings with dates, locations & notes
- Rate films with half-star precision
- Track rewatch history and statistics

Content Curation:

- Create personalized lists
- Browse platform-curated collections
- Organize watchlists for future viewing

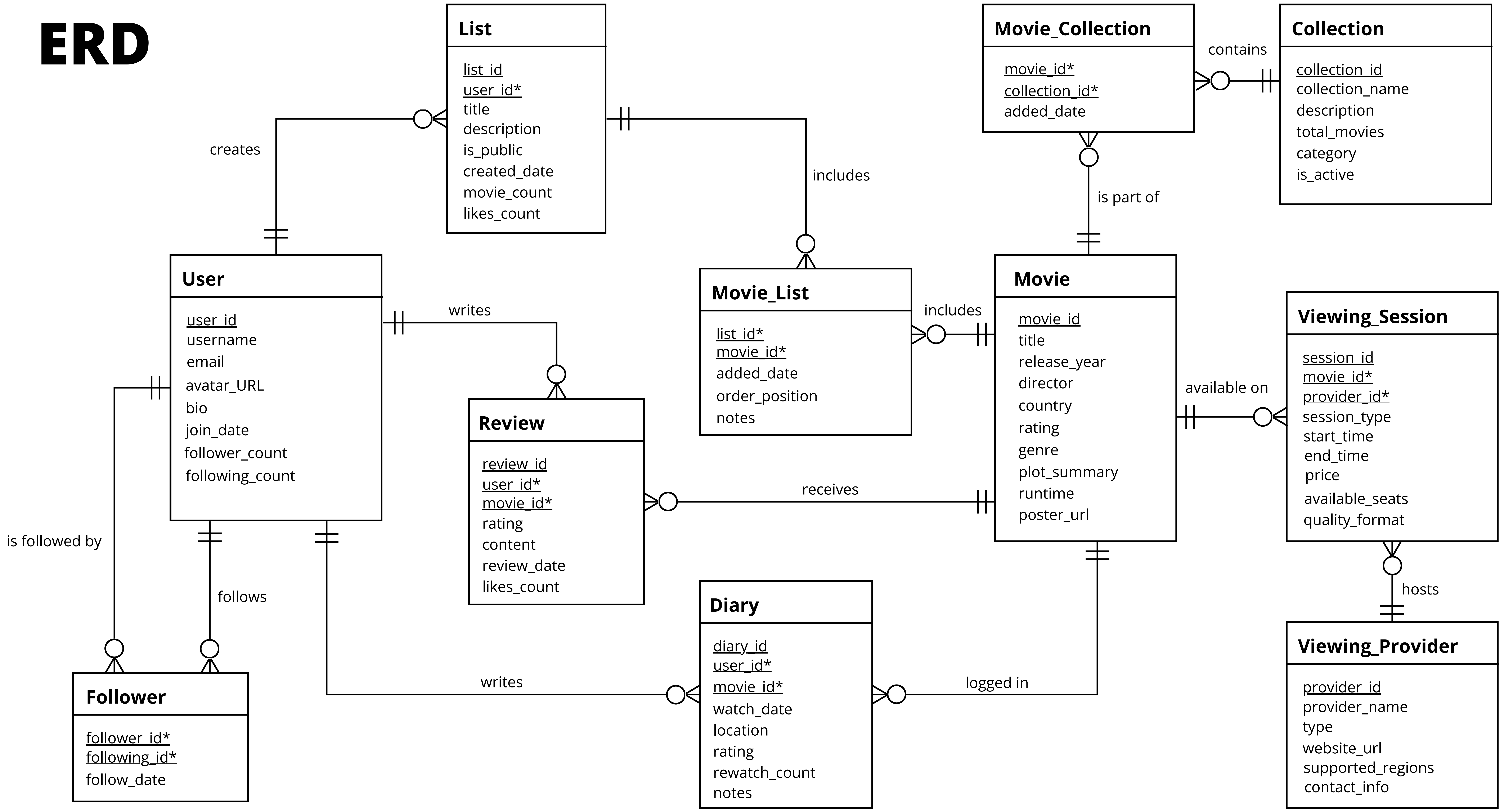
Social Interaction:

- Follow other users for film discovery
- Write and read detailed reviews
- Build social feed of friends' activity

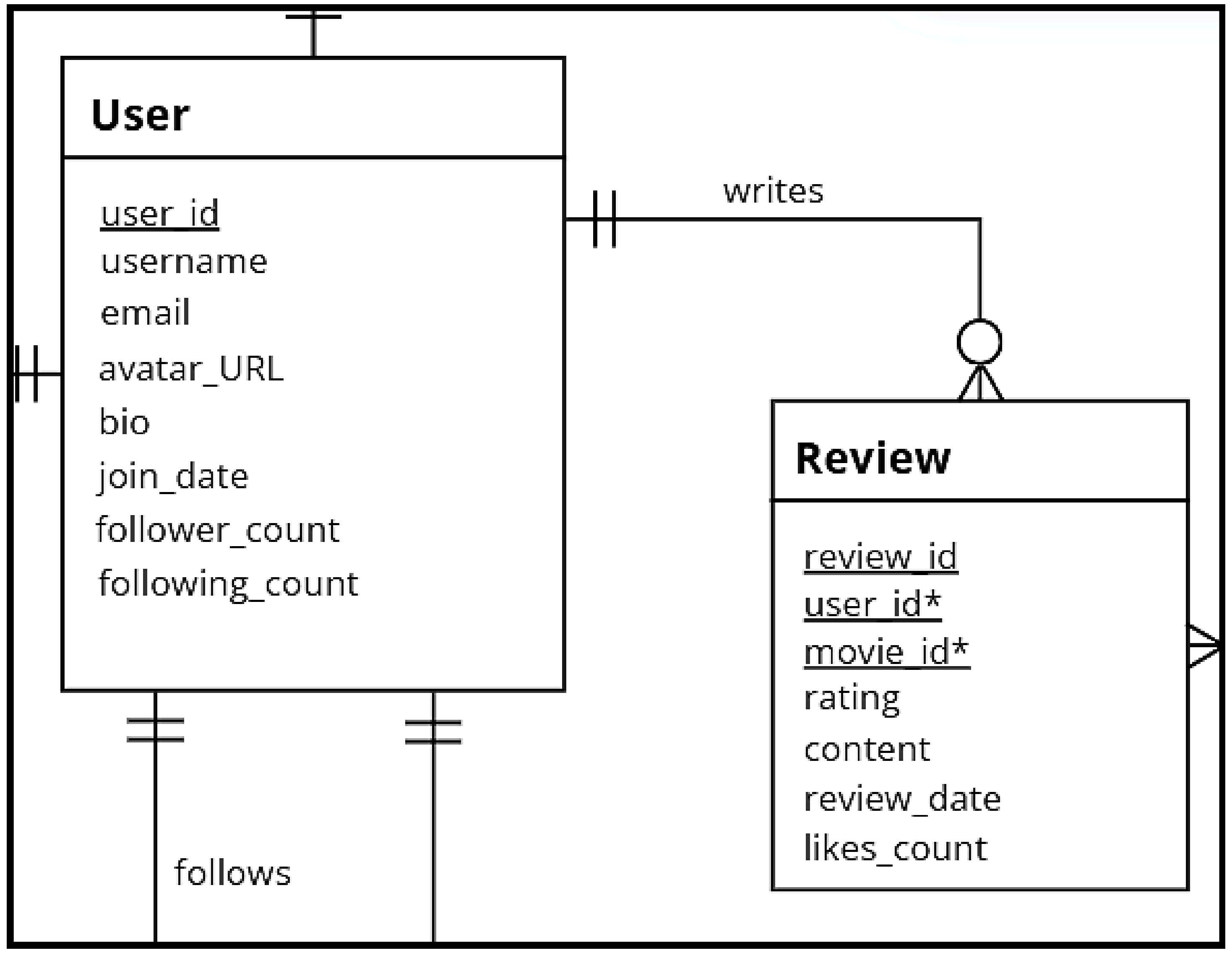
Discovery & Access:

- Find films across streaming services & theaters
- Explore by director, genre, era, or cultural impact

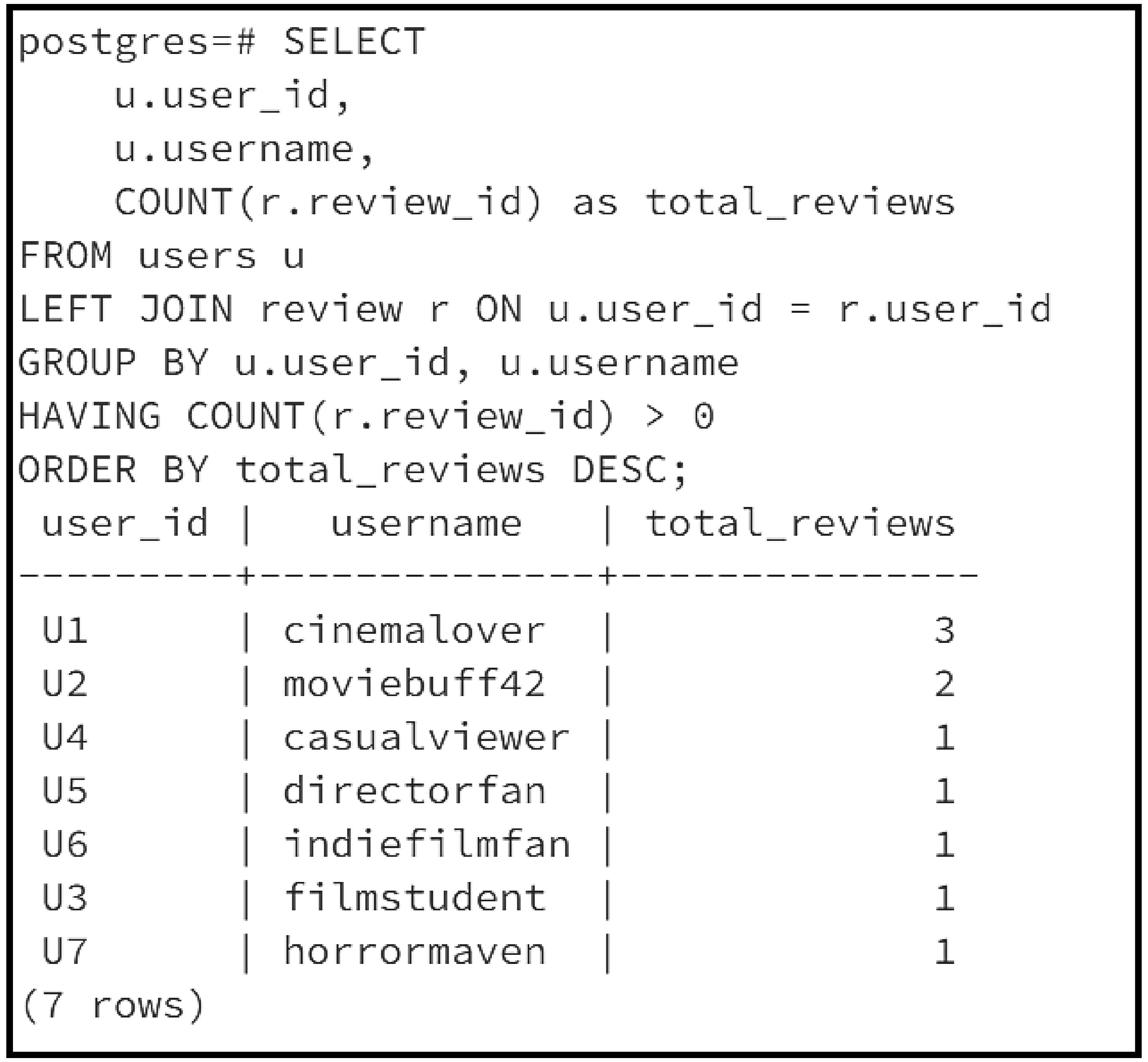
ERD


List
list_id
user_id*
title
description
is_public
created_date
movie_count
likes_count
creates
includes
Movie_Collection
movie_id*
collection_id*
added_date
contains
Collection
collection_id
collection_name
description
total_movies
category
is_active
is part of
User
user_id
username
email
avatar_URL
bio
join_date
follower_count
following_count
writes
Review
review_id
user_id*
movie_id*
rating
content
review_date
likes_count
Movie_List
list_id*
movie_id*
added_date
order_position
notes
includes
Movie
movie_id
title
release_year
director
country
rating
genre
plot_summary
runtime
poster_url
available on
Viewing_Session
session_id
movie_id*
provider_id*
session_type
start_time
end_time
price
available_seats
quality_format
receives
is followed by
follows
hosts
Follower
follower_id*
following_id*
follow_date
writes
Diary
diary_id
user_id*
movie_id*
watch_date
location
rating
rewatch_count
notes
logged in
Viewing_Provider
provider_id
provider_name
type
website_url
supported_regions
contact_info

An Example One-to-many Relationship

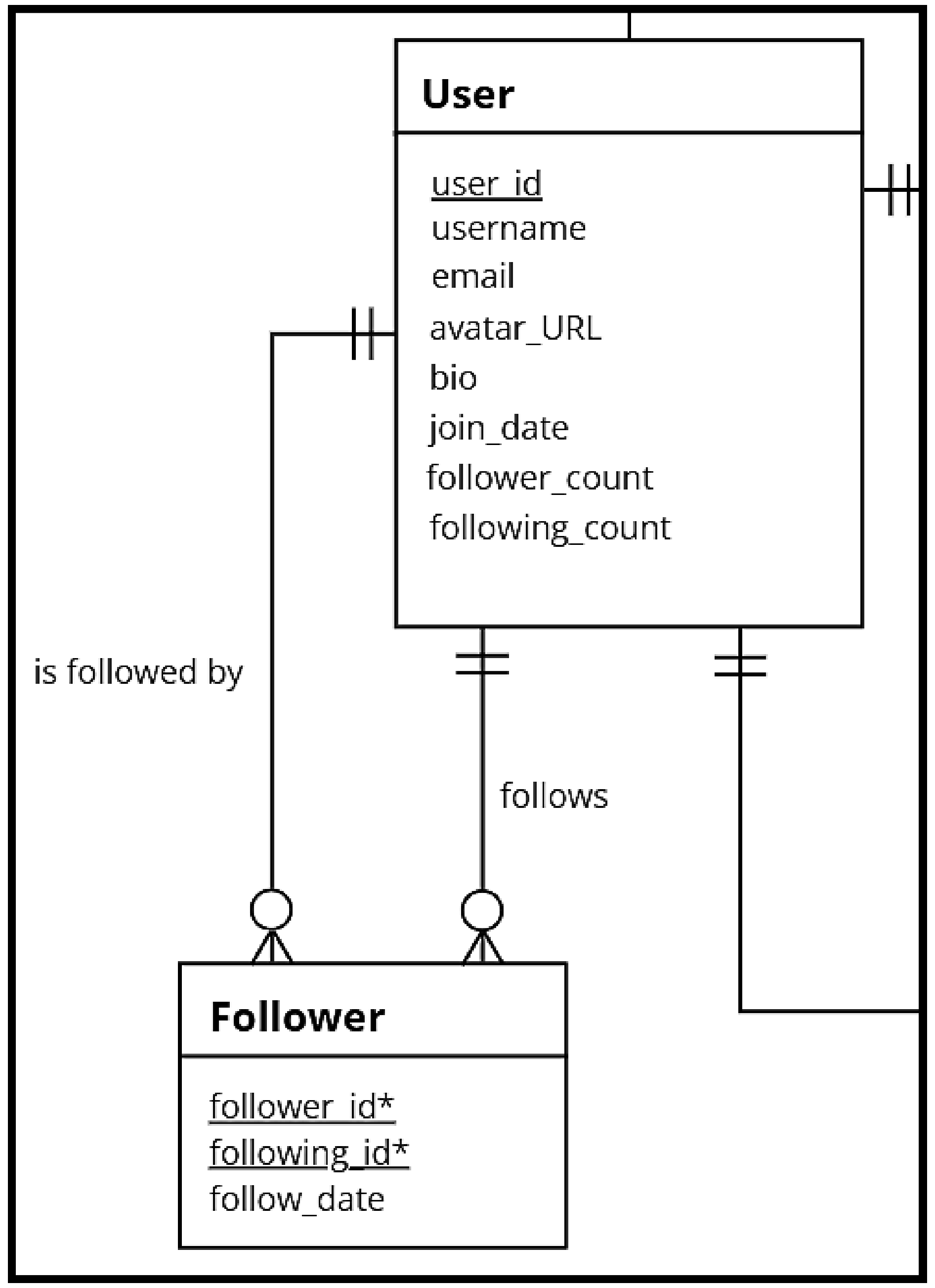


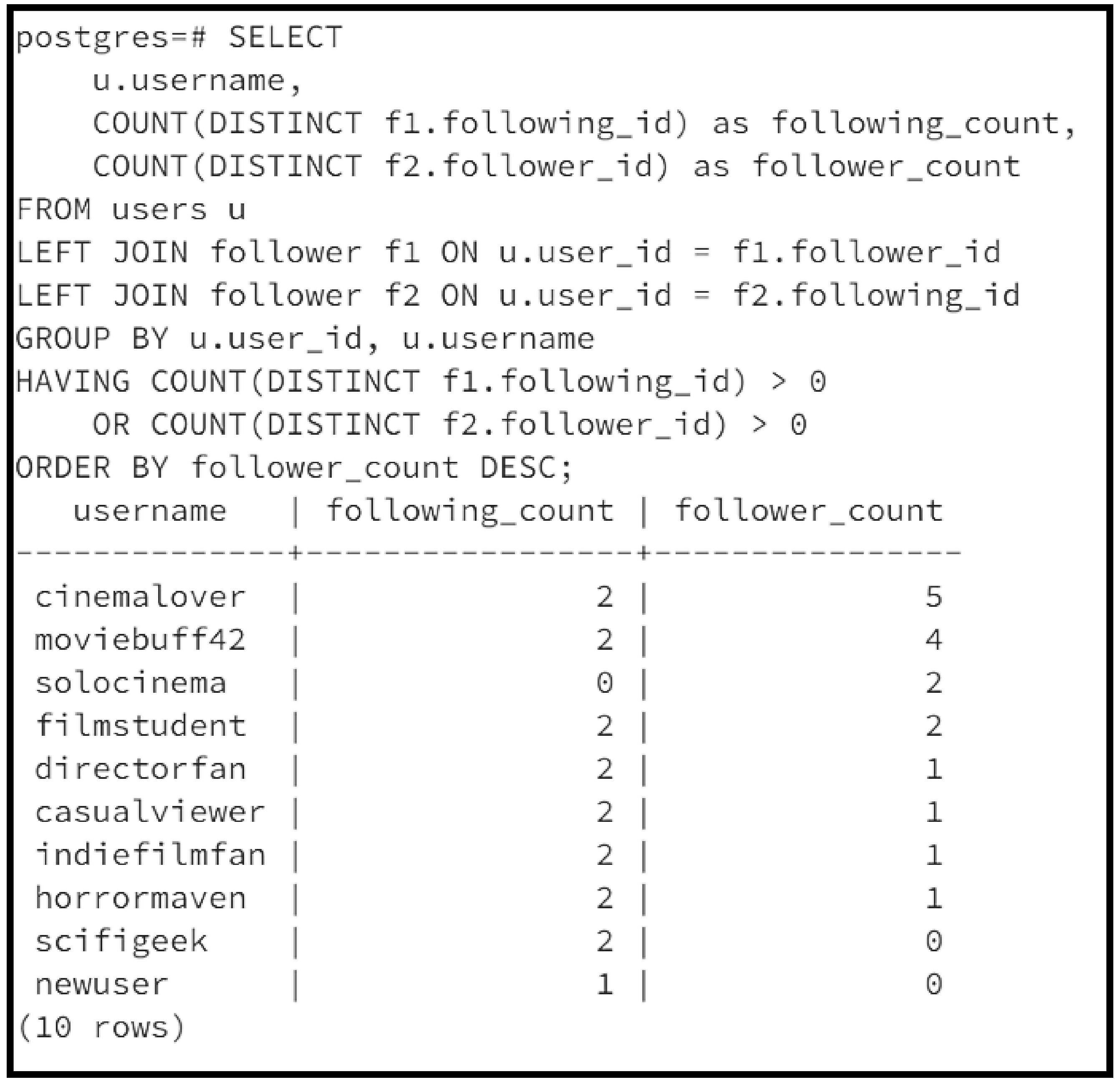
- Each user can write **multiple reviews**
- Each review is written by **exactly one user**



```
postgres=# SELECT
    u.user_id,
    u.username,
    COUNT(r.review_id) as total_reviews
FROM users u
LEFT JOIN review r ON u.user_id = r.user_id
GROUP BY u.user_id, u.username
HAVING COUNT(r.review_id) > 0
ORDER BY total_reviews DESC;
 user_id |   username    | total_reviews
---------+--------------+---------------
 U1      | cinemalover  |             3
 U2      | moviebuff42  |             2
 U4      | casualviewer |             1
 U5      | directorfan  |             1
 U6      | indiefilmfan |             1
 U3      | filmstudent  |             1
 U7      | horrormaven  |             1
(7 rows)
```

An Example Many-to-many Relationship



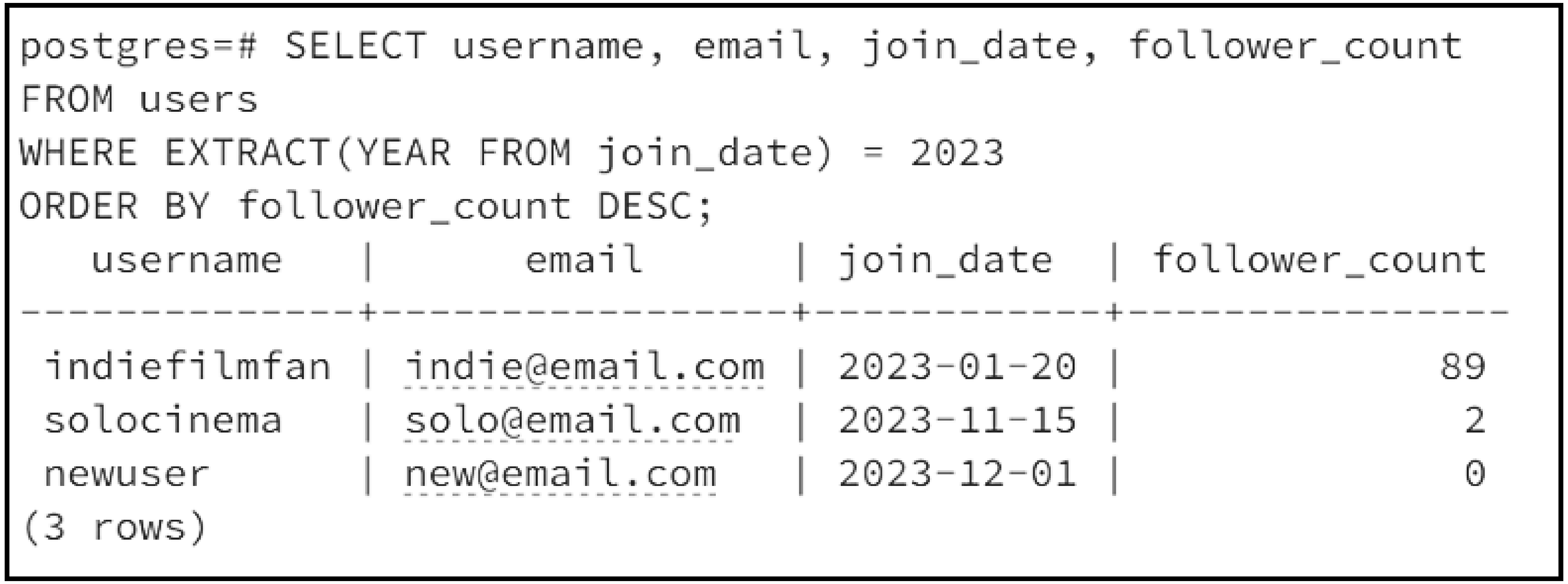


```
postgres=# SELECT
    u.username,
    COUNT(DISTINCT f1.following_id) as following_count,
    COUNT(DISTINCT f2.follower_id) as follower_count
FROM users u
LEFT JOIN follower f1 ON u.user_id = f1.follower_id
LEFT JOIN follower f2 ON u.user_id = f2.following_id
GROUP BY u.user_id, u.username
HAVING COUNT(DISTINCT f1.following_id) > 0
    OR COUNT(DISTINCT f2.follower_id) > 0
ORDER BY follower_count DESC;
   username    | following_count | follower_count
---------------+-----------------+----------------
 cinemalover   |               2 |              5
 moviebuff42   |               2 |              4
 solocinema    |               0 |              2
 filmstudent   |               2 |              2
 directorfan   |               2 |              1
 casualviewer  |               2 |              1
 indiefilmfan  |               2 |              1
 horrormaven   |               2 |              1
 scifigeek     |               2 |              0
 newuser       |               1 |              0
(10 rows)
```

- Each user is followed by multiple users
- Each user follows multiple users
- Associative entity (Follower) for M:M relationship
- Composite PK ensures uniqueness

Query 1 - Simple Single Table Query

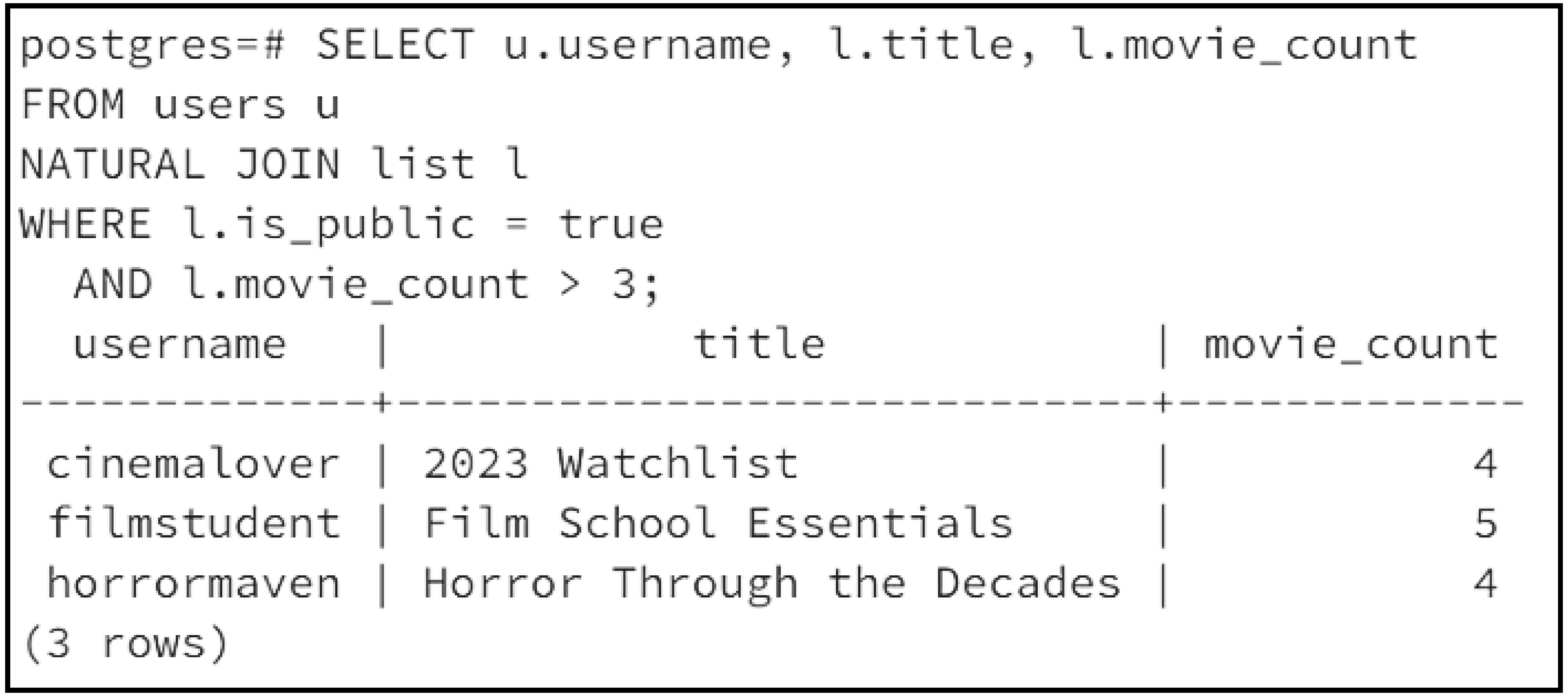
Find all active users who joined in 2023, showing their username, join date, and follower count, ordered by most popular first.



```
postgres=# SELECT username, email, join_date, follower_count
FROM users
WHERE EXTRACT(YEAR FROM join_date) = 2023
ORDER BY follower_count DESC;
   username   |      email       | join_date  | follower_count
--------------+------------------+------------+----------------
 indiefilmfan | indie@email.com  | 2023-01-20 |             89
 solocinema   | solo@email.com   | 2023-11-15 |              2
 newuser      | new@email.com    | 2023-12-01 |              0
(3 rows)
```

Query 2 - Natural Join Query

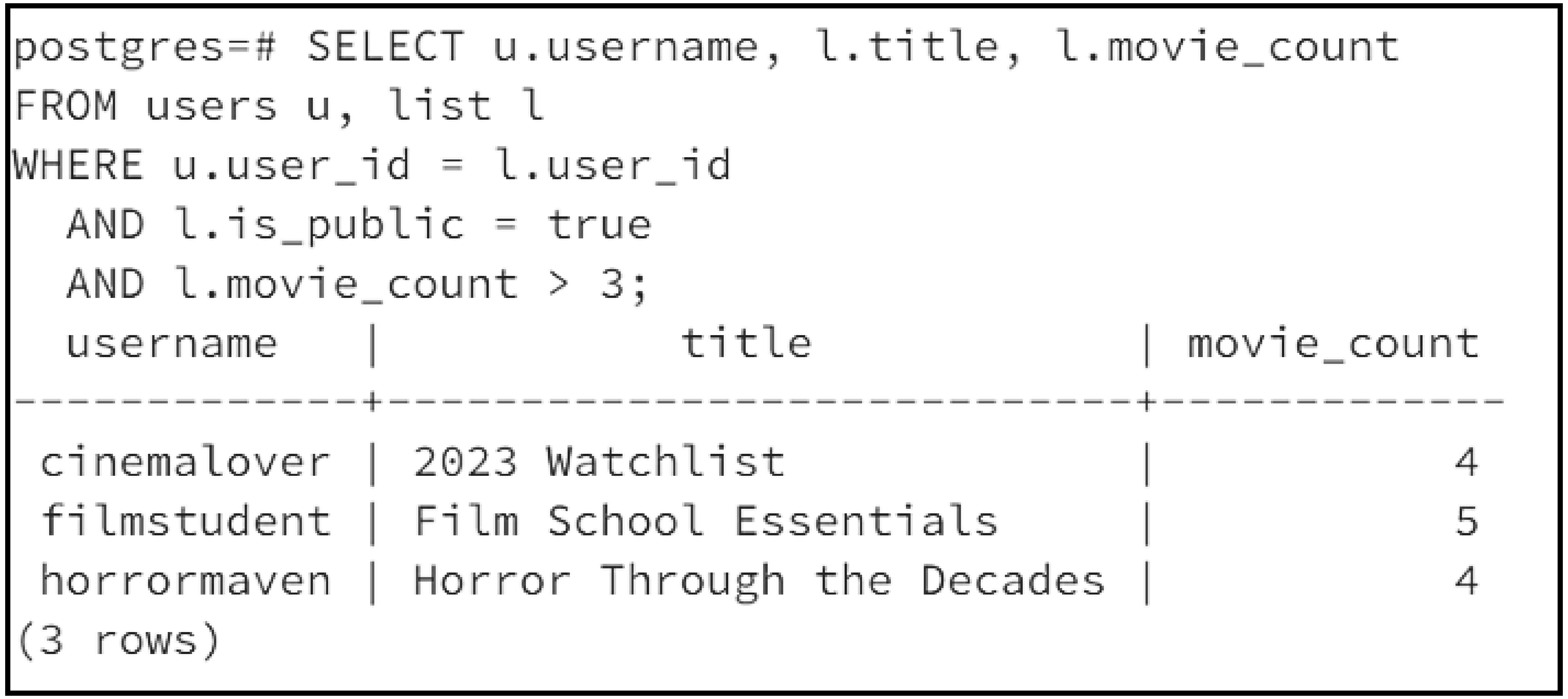
Find public lists created by users, showing username and list title for lists with more than 3 movies.



```
postgres=# SELECT u.username, l.title, l.movie_count
FROM users u
NATURAL JOIN list l
WHERE l.is_public = true
  AND l.movie_count > 3;
  username   |            title            | movie_count
-------------+-----------------------------+-------------
 cinemalover | 2023 Watchlist              |           4
 filmstudent | Film School Essentials      |           5
 horrormaven | Horror Through the Decades  |           4
(3 rows)
```

Query 3 - Cross Product Equivalent

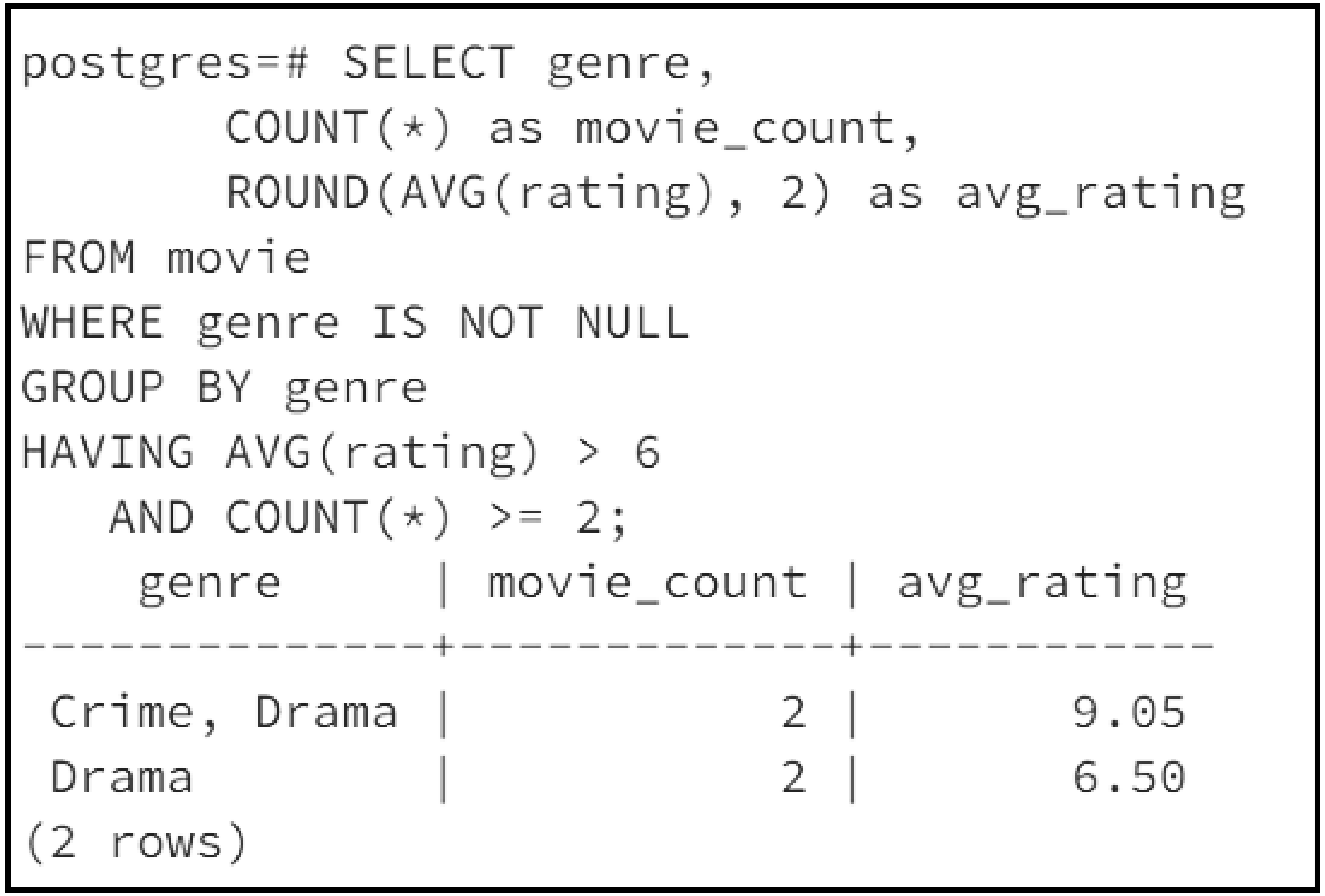
Same as query b but using cross product: public lists with more than 3 movies and their creators.



```
postgres=# SELECT u.username, l.title, l.movie_count
FROM users u, list l
WHERE u.user_id = l.user_id
  AND l.is_public = true
  AND l.movie_count > 3;
  username   |           title            | movie_count
-------------+----------------------------+-------------
 cinemalover | 2023 Watchlist             |           4
 filmstudent | Film School Essentials     |           5
 horrormaven | Horror Through the Decades |           4
(3 rows)
```

Query 4 - GROUP BY with HAVING

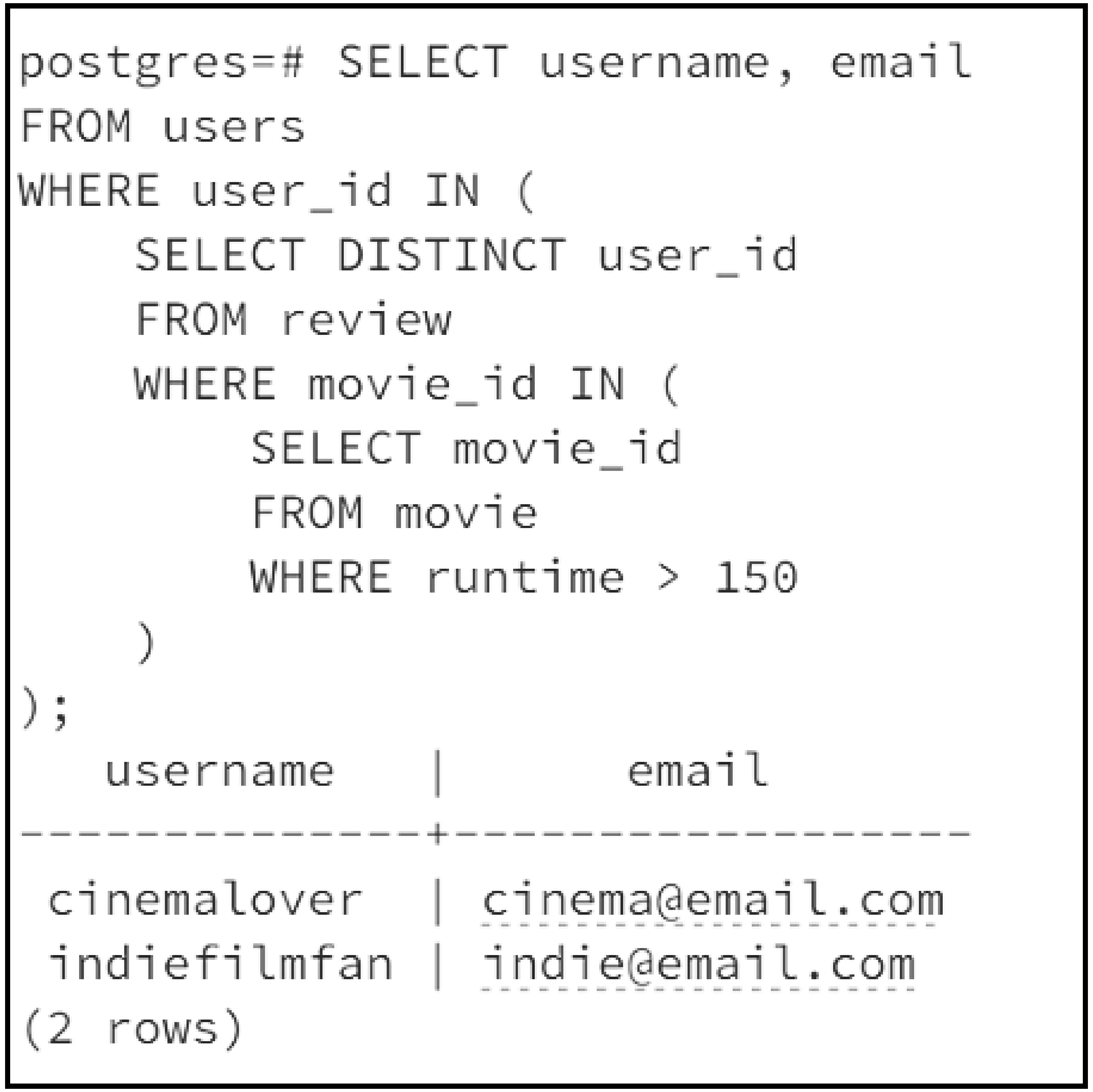
Find movie genres that have an average rating above 6 and contain at least 2 movies.



```
postgres=# SELECT genre,
       COUNT(*) as movie_count,
       ROUND(AVG(rating), 2) as avg_rating
FROM movie
WHERE genre IS NOT NULL
GROUP BY genre
HAVING AVG(rating) > 6
   AND COUNT(*) >= 2;
    genre     | movie_count | avg_rating
--------------+-------------+------------
 Crime, Drama |           2 |       9.05
 Drama        |           2 |       6.50
(2 rows)
```

Query 5 - Subquery

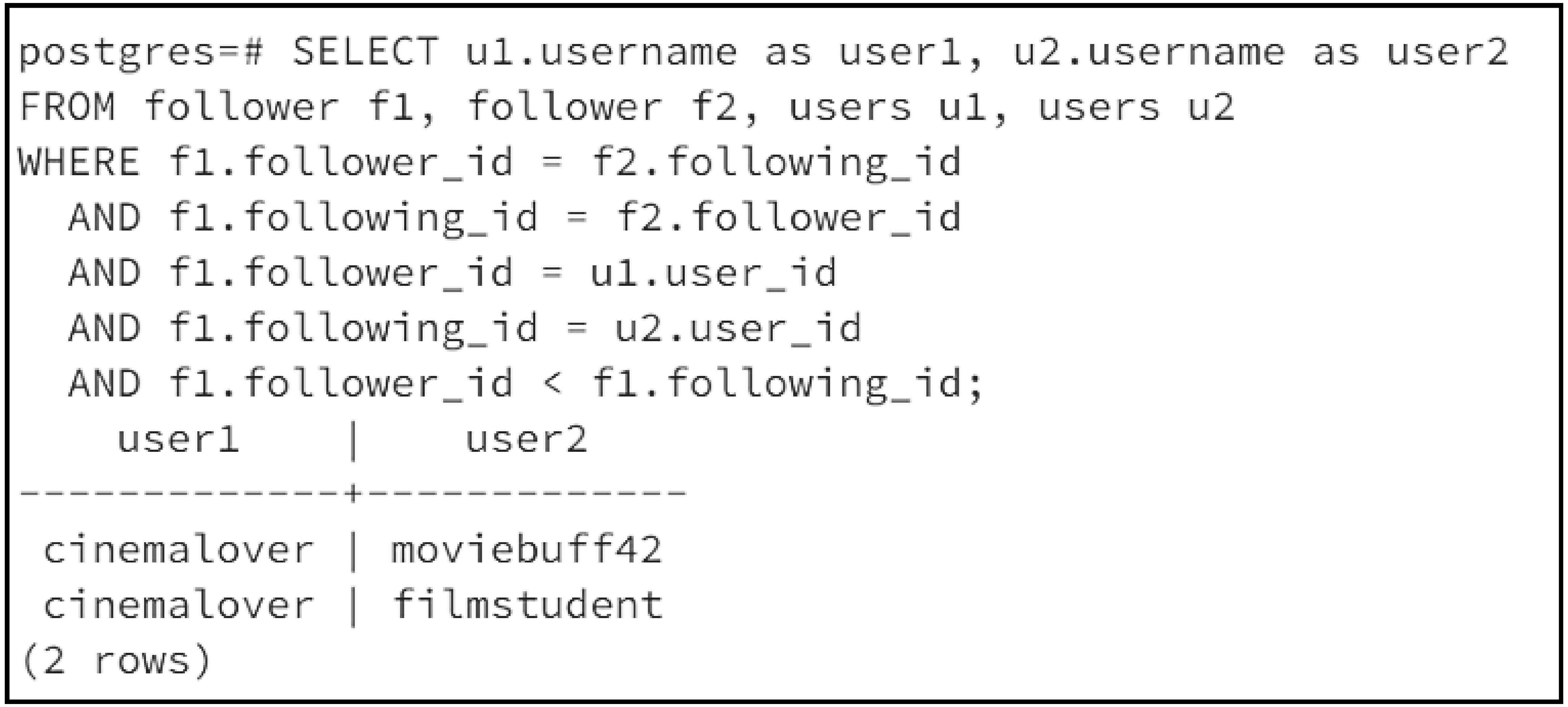
Find users who have reviewed movies that are longer than 150 minutes.



```
postgres=# SELECT username, email
FROM users
WHERE user_id IN (
    SELECT DISTINCT user_id
    FROM review
    WHERE movie_id IN (
        SELECT movie_id
        FROM movie
        WHERE runtime > 150
    )
);
   username   |      email
--------------+------------------
 cinemalover  | cinema@email.com
 indiefilmfan | indie@email.com
(2 rows)
```

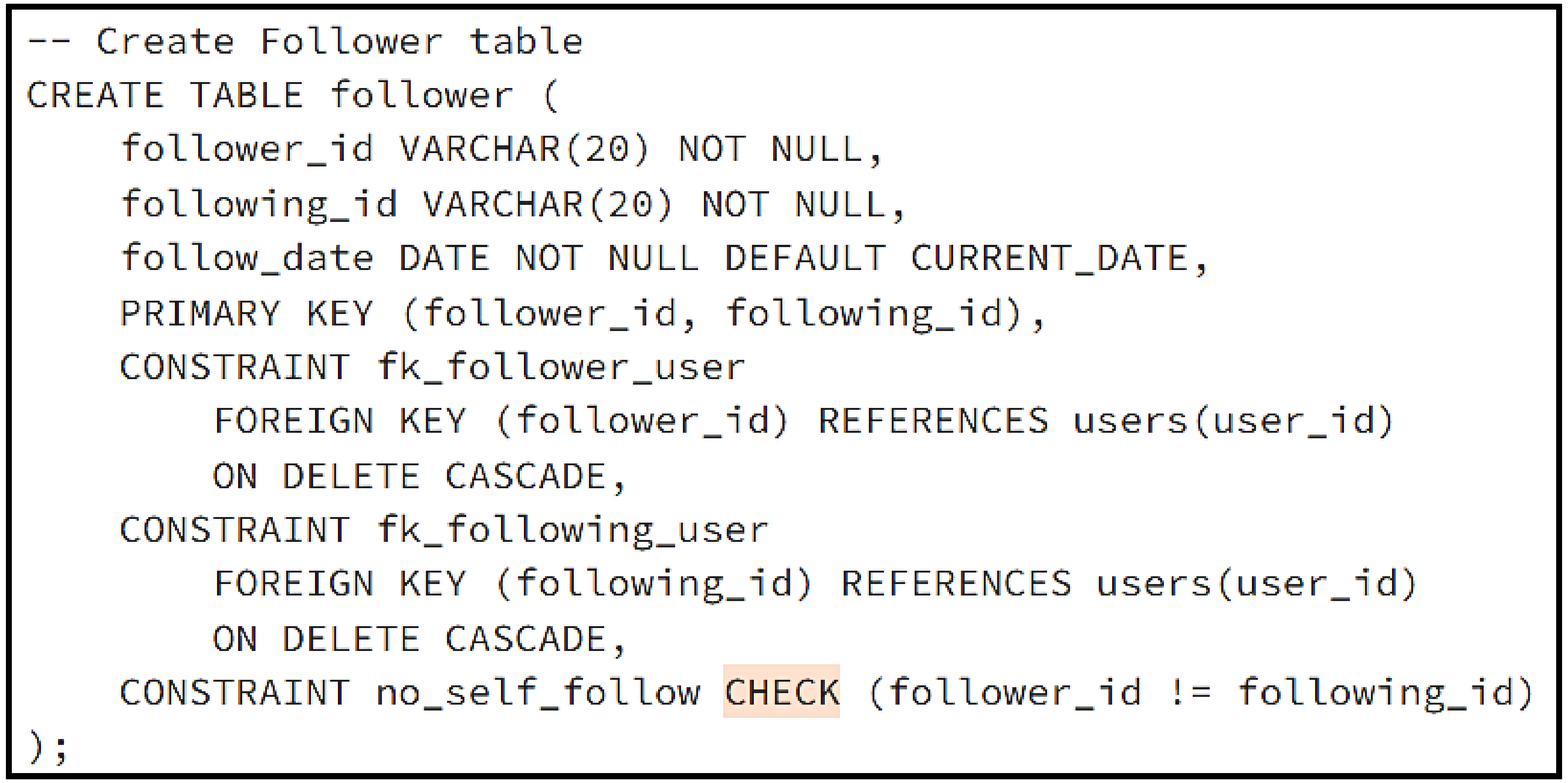
Query 6 - Self Join (No NATURAL JOIN)

Find pairs of different users where both follow each other (mutual follows).

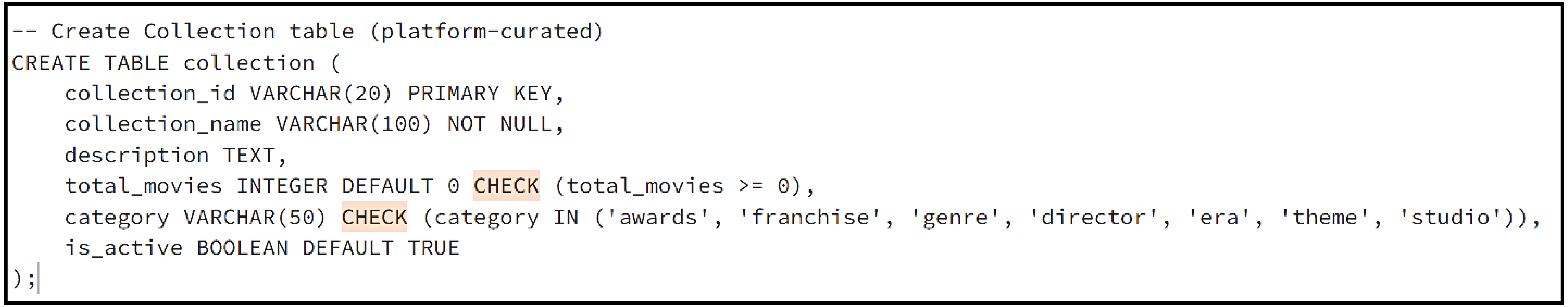


```
postgres=# SELECT u1.username as user1, u2.username as user2
FROM follower f1, follower f2, users u1, users u2
WHERE f1.follower_id = f2.following_id
  AND f1.following_id = f2.follower_id
  AND f1.follower_id = u1.user_id
  AND f1.following_id = u2.user_id
  AND f1.follower_id < f1.following_id;
    user1    |    user2
-------------+-------------
 cinemalover | moviebuff42
 cinemalover | filmstudent
(2 rows)
```

CHECK Statements



```sql
-- Create Follower table
CREATE TABLE follower (
    follower_id VARCHAR(20) NOT NULL,
    following_id VARCHAR(20) NOT NULL,
    follow_date DATE NOT NULL DEFAULT CURRENT_DATE,
    PRIMARY KEY (follower_id, following_id),
    CONSTRAINT fk_follower_user
        FOREIGN KEY (follower_id) REFERENCES users(user_id)
        ON DELETE CASCADE,
    CONSTRAINT fk_following_user
        FOREIGN KEY (following_id) REFERENCES users(user_id)
        ON DELETE CASCADE,
    CONSTRAINT no_self_follow CHECK (follower_id != following_id)
);
```



```sql
-- Create Collection table (platform-curated)
CREATE TABLE collection (
    collection_id VARCHAR(20) PRIMARY KEY,
    collection_name VARCHAR(100) NOT NULL,
    description TEXT,
    total_movies INTEGER DEFAULT 0 CHECK (total_movies >= 0),
    category VARCHAR(50) CHECK (category IN ('awards', 'franchise', 'genre', 'director', 'era', 'theme', 'studio')),
    is_active BOOLEAN DEFAULT TRUE
);
```

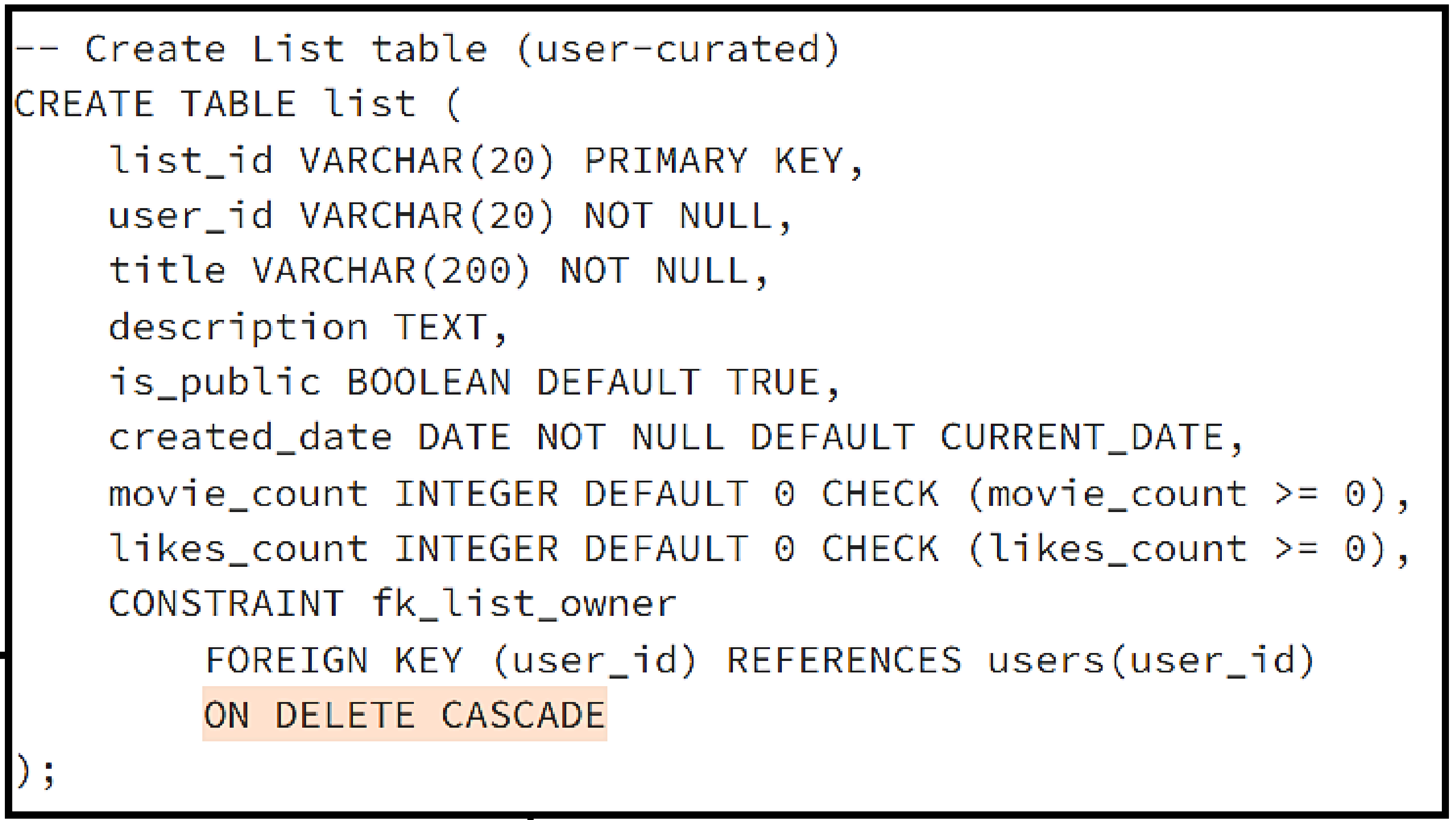
CHECK Statements

```sql
-- Create Users table
CREATE TABLE users (
    user_id VARCHAR(20) PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    bio TEXT,
    avatar_URL VARCHAR(255),
    join_date DATE NOT NULL DEFAULT CURRENT_DATE,
    follower_count INTEGER DEFAULT 0 CHECK (follower_count >= 0),
    following_count INTEGER DEFAULT 0 CHECK (following_count >= 0)
);
```

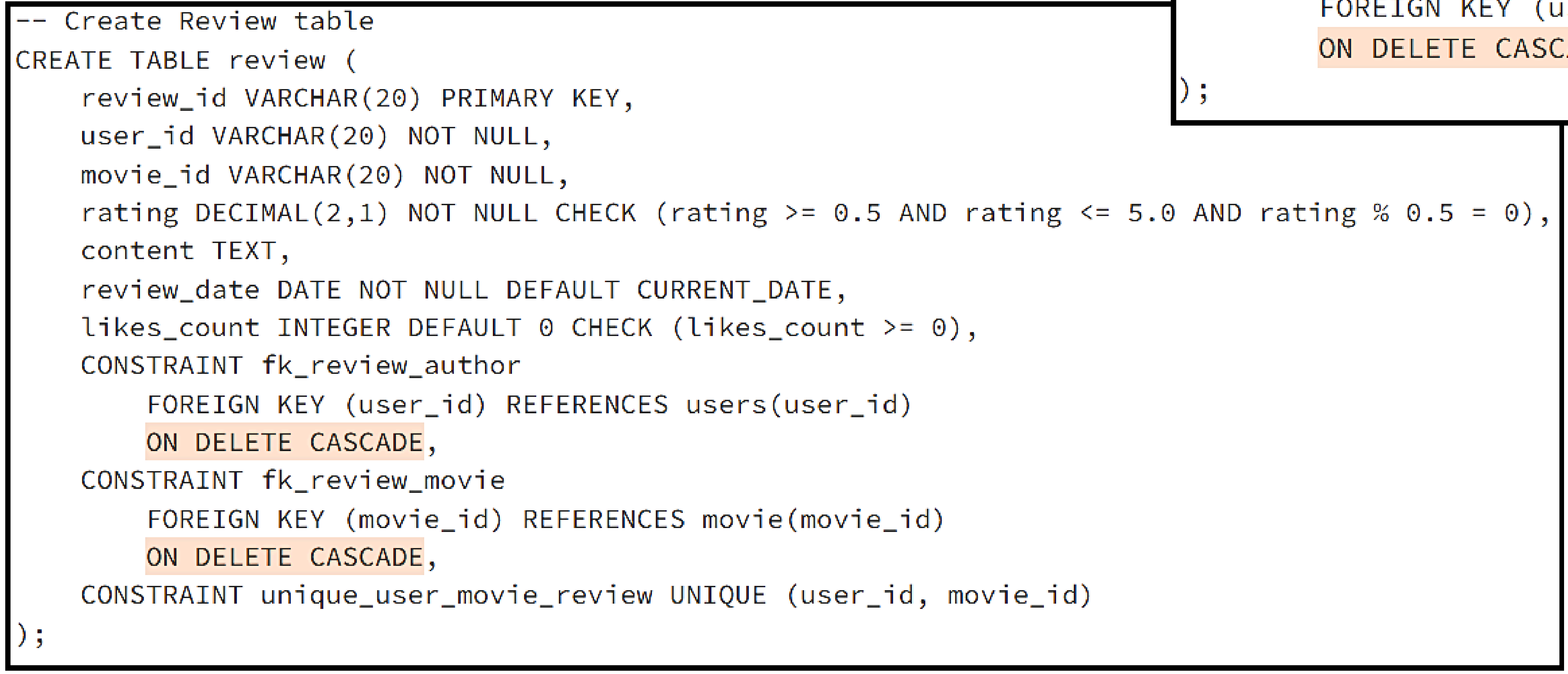
```sql
-- Create Diary table
CREATE TABLE diary (
    diary_id VARCHAR(20) PRIMARY KEY,
    user_id VARCHAR(20) NOT NULL,
    movie_id VARCHAR(20) NOT NULL,
    watch_date DATE NOT NULL,
    location VARCHAR(100),
    rating DECIMAL(2,1) CHECK (rating >= 0.5 AND rating <= 5.0 AND rating % 0.5 = 0),
    rewatch_count INTEGER DEFAULT 0 CHECK (rewatch_count >= 0),
    notes TEXT,
    CONSTRAINT fk_diary_user
        FOREIGN KEY (user_id) REFERENCES users(user_id)
        ON DELETE CASCADE,
    CONSTRAINT fk_diary_movie
        FOREIGN KEY (movie_id) REFERENCES movie(movie_id)
        ON DELETE CASCADE
);
```

Action Statements

When a user is deleted, all their content is automatically removed.

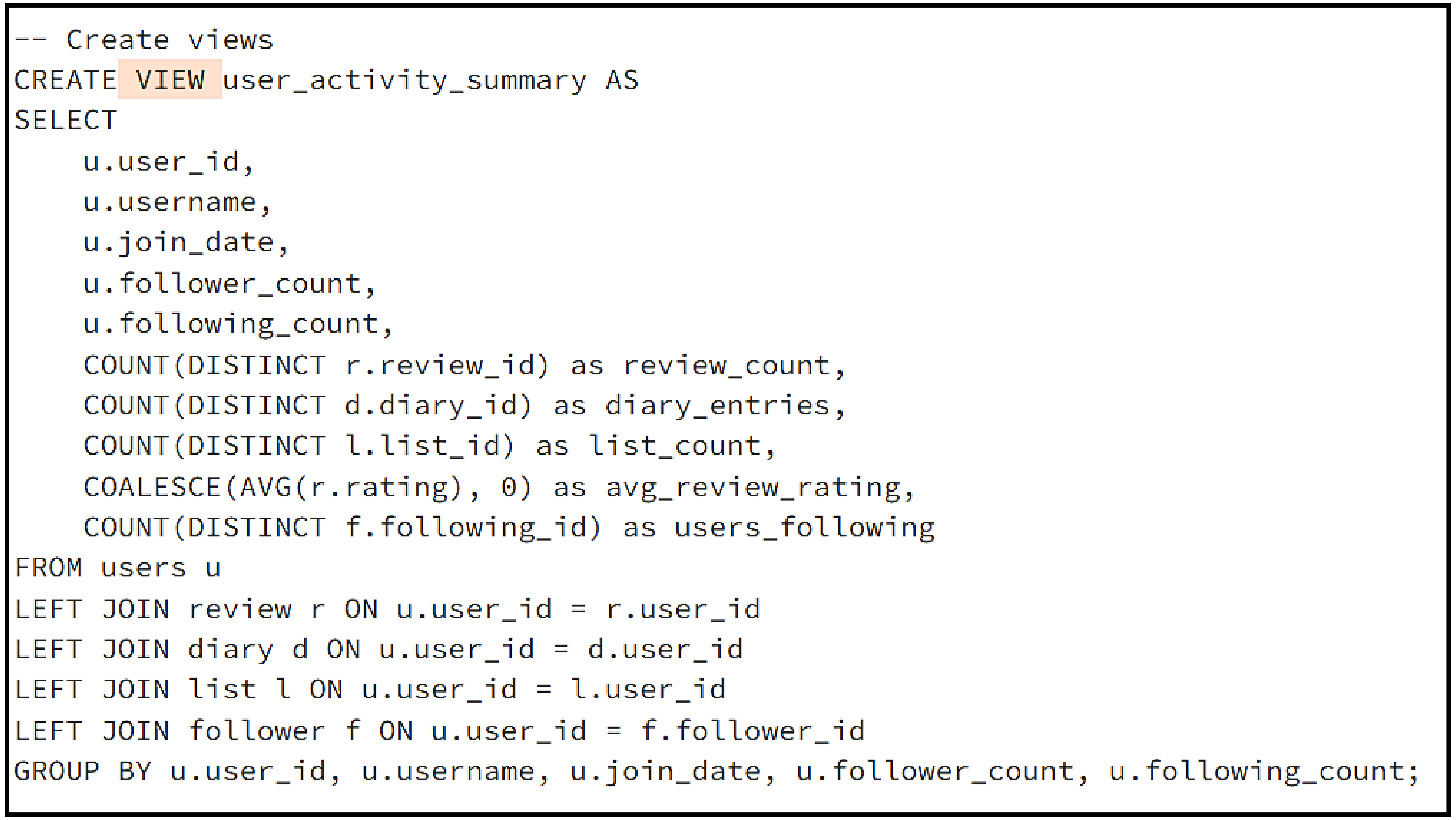


```
-- Create List table (user-curated)
CREATE TABLE list (
    list_id VARCHAR(20) PRIMARY KEY,
    user_id VARCHAR(20) NOT NULL,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    is_public BOOLEAN DEFAULT TRUE,
    created_date DATE NOT NULL DEFAULT CURRENT_DATE,
    movie_count INTEGER DEFAULT 0 CHECK (movie_count >= 0),
    likes_count INTEGER DEFAULT 0 CHECK (likes_count >= 0),
    CONSTRAINT fk_list_owner
        FOREIGN KEY (user_id) REFERENCES users(user_id)
        ON DELETE CASCADE
);
```



```
-- Create Review table
CREATE TABLE review (
    review_id VARCHAR(20) PRIMARY KEY,
    user_id VARCHAR(20) NOT NULL,
    movie_id VARCHAR(20) NOT NULL,
    rating DECIMAL(2,1) NOT NULL CHECK (rating >= 0.5 AND rating <= 5.0 AND rating % 0.5 = 0),
    content TEXT,
    review_date DATE NOT NULL DEFAULT CURRENT_DATE,
    likes_count INTEGER DEFAULT 0 CHECK (likes_count >= 0),
    CONSTRAINT fk_review_author
        FOREIGN KEY (user_id) REFERENCES users(user_id)
        ON DELETE CASCADE,
    CONSTRAINT fk_review_movie
        FOREIGN KEY (movie_id) REFERENCES movie(movie_id)
        ON DELETE CASCADE,
    CONSTRAINT unique_user_movie_review UNIQUE (user_id, movie_id)
);
```

SQL View

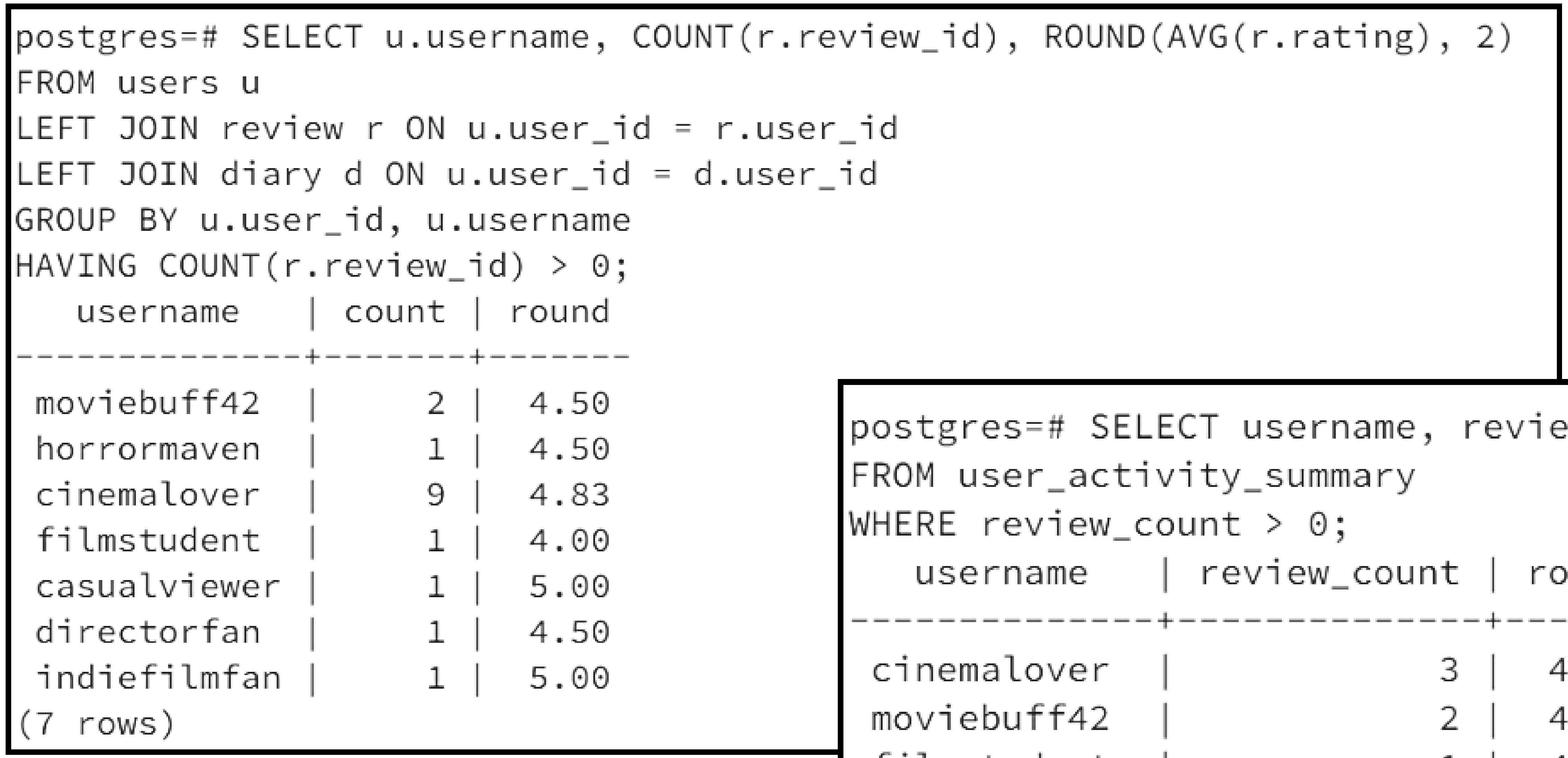


```
-- Create views
CREATE VIEW user_activity_summary AS
SELECT
    u.user_id,
    u.username,
    u.join_date,
    u.follower_count,
    u.following_count,
    COUNT(DISTINCT r.review_id) as review_count,
    COUNT(DISTINCT d.diary_id) as diary_entries,
    COUNT(DISTINCT l.list_id) as list_count,
    COALESCE(AVG(r.rating), 0) as avg_review_rating,
    COUNT(DISTINCT f.following_id) as users_following
FROM users u
LEFT JOIN review r ON u.user_id = r.user_id
LEFT JOIN diary d ON u.user_id = d.user_id
LEFT JOIN list l ON u.user_id = l.user_id
LEFT JOIN follower f ON u.user_id = f.follower_id
GROUP BY u.user_id, u.username, u.join_date, u.follower_count, u.following_count;
```

VIEW provides a simplified interface to complex data. It updates automatically when underlying data changes

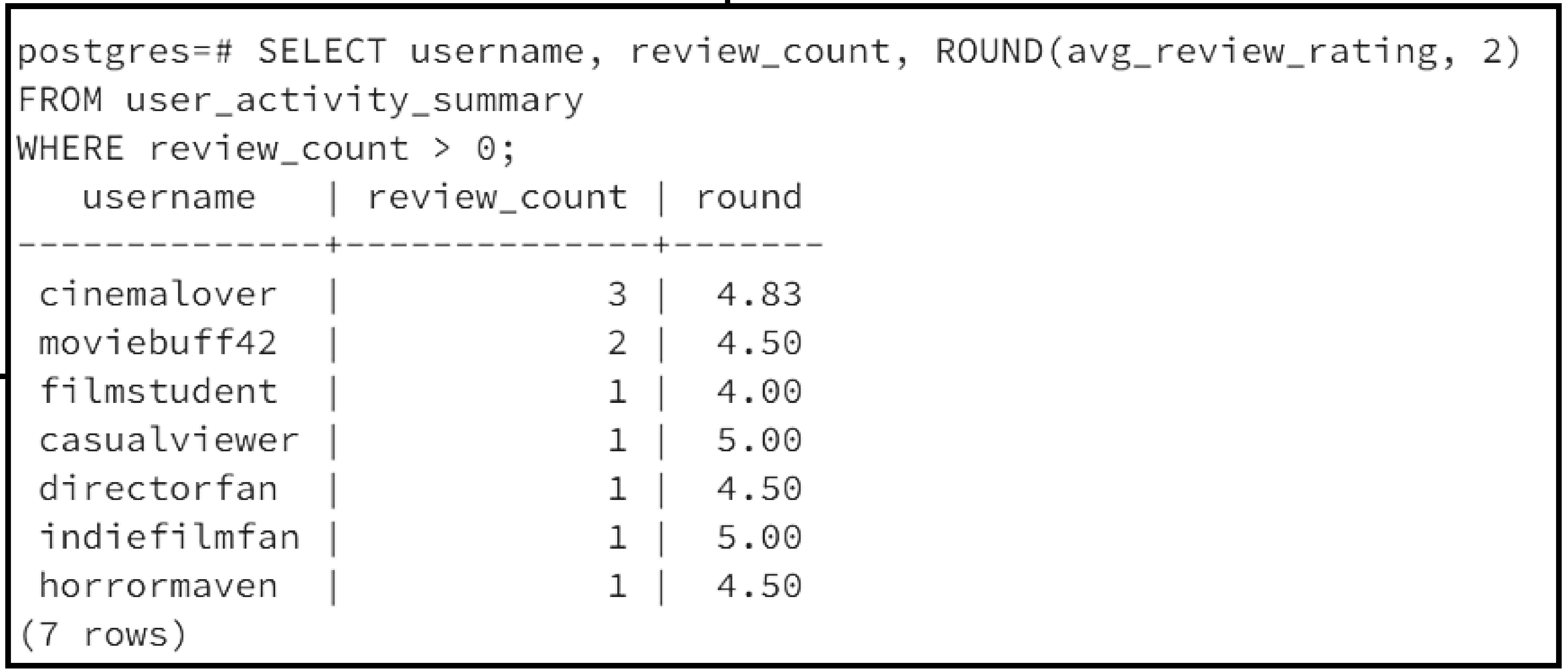
SQL View

Before View:



```
postgres=# SELECT u.username, COUNT(r.review_id), ROUND(AVG(r.rating), 2)
FROM users u
LEFT JOIN review r ON u.user_id = r.user_id
LEFT JOIN diary d ON u.user_id = d.user_id
GROUP BY u.user_id, u.username
HAVING COUNT(r.review_id) > 0;
   username    | count | round
---------------+-------+-------
 moviebuff42   |     2 |  4.50
 horrormaven   |     1 |  4.50
 cinemalover   |     9 |  4.83
 filmstudent   |     1 |  4.00
 casualviewer  |     1 |  5.00
 directorfan   |     1 |  4.50
 indiefilmfan  |     1 |  5.00
(7 rows)
```

After View:



```
postgres=# SELECT username, review_count, ROUND(avg_review_rating, 2)
FROM user_activity_summary
WHERE review_count > 0;
   username    | review_count | round
---------------+--------------+-------
 cinemalover   |            3 |  4.83
 moviebuff42   |            2 |  4.50
 filmstudent   |            1 |  4.00
 casualviewer  |            1 |  5.00
 directorfan   |            1 |  4.50
 indiefilmfan  |            1 |  5.00
 horrormaven   |            1 |  4.50
(7 rows)
```