

# Movies SQL Project Log

~ Anushka Sharma

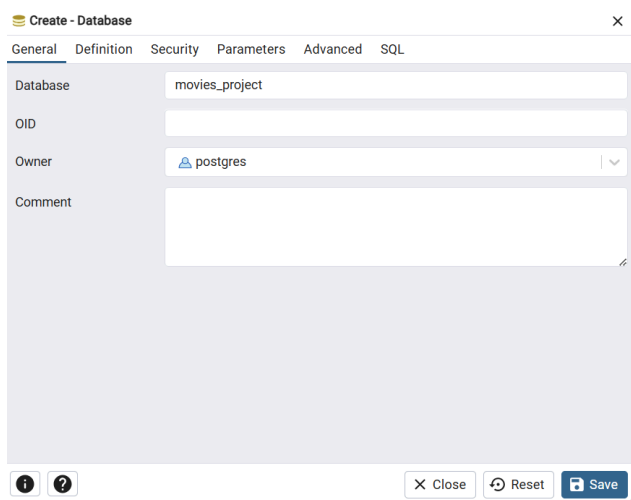
**Dataset Source:** <https://grouplens.org/datasets/movielens/latest/>

**Datasets:** movies.csv, ratings.csv, tags.csv, links.csv, genome\_scores.csv, genome\_tags.csv

## Step-by-Step Project Log

### Step 1: Create Database

**Screenshot:**



**Query:**

```
CREATE DATABASE movies_project
WITH
OWNER = postgres
ENCODING = 'UTF8'
LOCALE_PROVIDER = 'libc'
CONNECTION LIMIT = -1
IS_TEMPLATE = False;
```

### Step 2: Create Tables

**Screenshot:**

Query	Query History	Query	Query History	Query	Query History
1 2 3 4 5	<pre>CREATE table genome_scores (     movieId INT,     tagId INT,     relevance DOUBLE PRECISION );</pre>	1 2 3 4	<pre>CREATE table genome_tags (     tagID INT PRIMARY KEY,     tag VARCHAR );</pre>	1 2 3 4 5	<pre>CREATE table links (     movieId INT,     imdbId INT,     tmdbId INT );</pre>

Query	Query History
1	CREATE table movies (
2	movieId INT PRIMARY KEY,
3	title VARCHAR,
4	genres TEXT
5	);

Query	Query History
1	CREATE table ratings (
2	userId INT,
3	movieId INT,
4	rating REAL,
5	timestamp BIGINT
6	);

Query	Query History
1	CREATE table tags (
2	userId INT,
3	movieId INT,
4	tag VARCHAR,
5	timestamp BIGINT
6	);

### Query:

```
CREATE table genome_scores (
    movieId INT,
    tagId INT,
    relevance DOUBLE PRECISION
);
```

```
CREATE table genome_tags (
    tagID INT PRIMARY KEY,
    tag VARCHAR
);
```

```
CREATE table links (
    movieId INT,
    imdbId INT,
    tmdbId INT
);
```

```
CREATE table movies (
    movieId INT PRIMARY KEY,
    title VARCHAR,
    genres TEXT
);
```

```
CREATE table ratings (
    userId INT,
    movieId INT,
    rating REAL,
    timestamp BIGINT
);
```

```
CREATE table tags (
    userId INT,
    movieId INT,
    tag VARCHAR,
    timestamp BIGINT
);
```

## Step 3: Import CSV Data

Right-click on each table → Import/Export.

Set Format: CSV

Select the appropriate file

Encoding: UTF8

Header: Yes

Delimiter: ,

Click OK

## Phase 1: Simple Analysis

### 1. List the first 10 movies with their genres.

Query:

```
SELECT * FROM movies LIMIT 10;
```

Screenshot:

	movieid [PK] integer	title character varying	genres text
1	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
2	2	Jumanji (1995)	Adventure Children Fantasy
3	3	Grumpier Old Men (1995)	Comedy Romance
4	4	Waiting to Exhale (1995)	Comedy Drama Romance
5	5	Father of the Bride Part II (1995)	Comedy
6	6	Heat (1995)	Action Crime Thriller
7	7	Sabrina (1995)	Comedy Romance
8	8	Tom and Huck (1995)	Adventure Children
9	9	Sudden Death (1995)	Action
10	10	GoldenEye (1995)	Action Adventure Thriller

### 2. Show all distinct genres in the dataset.

Query:

```
SELECT DISTINCT unnest(string_to_array(genres, '|')) AS genre
FROM movies
WHERE genres IS NOT NULL
ORDER BY genre;
```

Screenshot:



	genre text 
1	(no genres listed)
2	Action
3	Adventure
4	Animation
5	Children
6	Comedy
7	Crime
8	Documentary
9	Drama
10	Fantasy
11	Film-Noir
12	Horror
13	IMAX
14	Musical
15	Mystery
16	Romance
17	Sci-Fi
18	Thriller
19	War
20	Western

**3. Find all movies that include “Comedy” in their genre.**

**Query:**

```
SELECT *
FROM movies
WHERE genres ILIKE '%Comedy%';
```

**Screenshot:**

	movieid [PK] integer 	title character varying 	genres text 
1	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	7	Sabrina (1995)	Comedy Romance
6	11	American President, The (1995)	Comedy Drama Romance
7	12	Dracula: Dead and Loving It (1995)	Comedy Horror
8	18	Four Rooms (1995)	Comedy
9	19	Ace Ventura: When Nature Calls (1995)	Comedy
10	20	Money Train (1995)	Action Comedy Crime Drama Thriller
11	21	Get Shorty (1995)	Comedy Crime Thriller
12	38	It Takes Two (1995)	Children Comedy
13	39	Clueless (1995)	Comedy Romance
14	45	To Die For (1995)	Comedy Drama Thriller
15	52	Mighty Aphrodite (1995)	Comedy Drama Romance
Total rows: 22830		Query complete 00:00:00.243	

**4. Count how many movies belong to the ‘Action’ genre.**

**Query:**

```
SELECT COUNT(*) AS action_movie_count
FROM movies
```

WHERE genres ILIKE '%Action%';

**Screenshot:**

	action_movie_count bigint
1	9563

**5. Find the average rating for the movie "Toy Story (1995)".**

**Query:**

```
SELECT AVG(r.rating) AS avg_rating
FROM ratings r
JOIN movies m ON r.movieId = m.movieId
WHERE m.title = 'Toy Story (1995)';
```

**Screenshot:**

	avg_rating double precision
1	3.8935076093890357

**6. List the top 5 highest-rated movies (by average rating), with their movie titles.**

**Query:**

```
SELECT m.title, ROUND(AVG(r.rating)::NUMERIC, 2) AS avg_rating
FROM ratings r
JOIN movies m ON r.movieId = m.movieId
GROUP BY m.title
ORDER BY avg_rating DESC
LIMIT 5;
```

**Screenshot:**


	title character varying	avg_rating numeric
1	11 September Vragen (2016)	5.00
2	'Tis the Season to be Merry (2021)	5.00
3	\$uperthief: Inside America's Biggest Bank Score (2012)	5.00
4	¡Se armó el belén! (1970)	5.00
5	1984 Revolution (2011)	5.00

**7. Count how many unique users have rated movies.**

**Query:**

```
SELECT COUNT(DISTINCT userId) AS unique_users
FROM ratings;
```

**Screenshot:**


	unique_users bigint 
1	330975

## 8. List all tags used by user ID 10.

### Query:

```
SELECT tag
FROM tags
WHERE userId = 10;
```

### Screenshot:


	tag character varying 
1	good vs evil
2	Harrison Ford
3	sci-fi

## 9. Find all movies tagged with "sci-fi".

### Query:

```
SELECT DISTINCT m.title
FROM movies m
JOIN tags t ON m.movieId = t.movieId
WHERE LOWER(t.tag) = 'sci-fi';
```

### Screenshot:

	title character varying 
1	*batteries not included (1987)
2	1984 (Nineteen Eighty-Four) (1984)
3	2001: A Space Odyssey (1968)
4	2010: The Year We Make Contact (1984)
5	2012 (2009)
6	2036: Nexus Dawn (2017)
7	2048: Nowhere to Run (2017)
8	2067 (2020)
9	28 Days Later (2002)
10	5th Passenger (2018)
11	65 (2023)
12	6th Day, The (2000)
13	9 (2009)
14	A Blaster In The Right Hands: A Star Wars Story (2021)
15	A Crimson Man (2017)
Total rows: 854    Query complete 00:00:00.668	

**10. Get the total number of ratings per movie, sorted by the most-rated movies.**

**Query:**

```
SELECT m.title, COUNT(r.rating) AS total_ratings
FROM ratings r
JOIN movies m ON r.movieId = m.movieId
GROUP BY m.title
ORDER BY total_ratings DESC;
```

**Screenshot:**

	title character varying	total_ratings bigint
1	Shawshank Redemption, The (1994)	122296
2	Forrest Gump (1994)	113581
3	Pulp Fiction (1994)	108756
4	Matrix, The (1999)	107056
5	Silence of the Lambs, The (1991)	101802
6	Star Wars: Episode IV - A New Hope (1977)	97202
7	Fight Club (1999)	86207
8	Schindler's List (1993)	84232
9	Jurassic Park (1993)	83026
10	Star Wars: Episode V - The Empire Strikes Back (1980)	80200
11	Lord of the Rings: The Fellowship of the Ring, The (2001)	79940
12	Toy Story (1995)	76813
13	Star Wars: Episode VI - Return of the Jedi (1983)	76773
14	Braveheart (1995)	75514
15	Lord of the Rings: The Return of the King, The (2003)	75512
Total rows: 83043    Query complete 00:00:08.509		

## Phase 2: Intermediate Analysis

**1. Join movies with their average ratings and show top 10 highest-rated movies with at least 100 ratings.**

**Query:**

```
SELECT
    m.title,
    ROUND(AVG(r.rating)::NUMERIC, 2) AS avg_rating,
    COUNT(r.rating) AS rating_count
FROM movies m
JOIN ratings r ON m.movieId = r.movieId
GROUP BY m.title
HAVING COUNT(r.rating) >= 100
ORDER BY avg_rating DESC
LIMIT 10;
```

**Screenshot:**

	title character varying 🔒	avg_rating numeric 🔒	rating_count bigint 🔒
1	Planet Earth II (2016)	4.45	2041
2	Planet Earth (2006)	4.45	3015
3	Shawshank Redemption, The (1994)	4.42	122296
4	Band of Brothers (2001)	4.42	2835
5	Cosmos	4.34	625
6	Parasite (2019)	4.33	12399
7	Godfather, The (1972)	4.33	75004
8	Blue Planet II (2017)	4.31	1267
9	Twelve Angry Men (1954)	4.31	332
10	Twin Peaks (1989)	4.30	1132

**2. Display userId and number of tags they have added, sorted in descending order.**

**Query:**

```
SELECT
    userId,
    COUNT(*) AS tag_count
FROM tags
GROUP BY userId
ORDER BY tag_count DESC;
```

**Screenshot:**

	userid integer 🔒	tag_count bigint 🔒
1	215490	723473
2	229729	20317
3	63350	19345
4	126357	17990
5	137062	16739
6	183281	15467
7	274127	15431
8	298708	13838
9	234824	13700
10	95828	12681
11	48766	12587
12	9138	12238
13	66383	11551
14	190434	10582
15	10730	10251
Total rows: 25280		Query complete

**3. Using a CASE statement, classify movies as 'Low', 'Medium', or 'High' rated based on average rating.**

**Query:**

```
SELECT
    m.title,
```



```

ROUND(AVG(r.rating)::NUMERIC, 2) AS avg_rating,
CASE
    WHEN AVG(r.rating) < 2 THEN 'Low'
    WHEN AVG(r.rating) BETWEEN 2 AND 4 THEN 'Medium'
    ELSE 'High'
END AS rating_category
FROM movies m
JOIN ratings r ON m.movieId = r.movieId
GROUP BY m.title
ORDER BY avg_rating DESC;

```

**Screenshot:**

	title character varying	avg_rating numeric	rating_category text
1	Bonga, o Vagabundo (1969)	5.00	High
2	Oh My Ghosts (2009)	5.00	High
3	How Not to Propose (2015)	5.00	High
4	Naked Angels (1969)	5.00	High
5	Debi (2018)	5.00	High
6	How I Met Your Murderer (2021)	5.00	High
7	Alonzo Bodden: Historically Incorrect (2016)	5.00	High
8	Blue Summer (1973)	5.00	High
9	Naledi: A Baby Elephant's Tale (2016)	5.00	High
10	Deconstructing the Beatles' Abbey Road: Side One	5.00	High
11	Deconstructing the Beatles' White Album (2016)	5.00	High
Total rows: 83043		Query complete 00:00:08.815	

**4. Get the top 3 tags applied most frequently across all movies.**

**Query:**

```

SELECT
    tag,
    COUNT(*) AS tag_count
FROM tags
GROUP BY tag
ORDER BY tag_count DESC
LIMIT 3;

```

**Screenshot:**

	tag character varying	tag_count bigint
1	sci-fi	14319
2	atmospheric	12172
3	action	10683

**5. List the most relevant tag (highest relevance score) for each movie from the genome\_scores and genome\_tags tables.**

**Query:**

```

WITH ranked_tags AS (

```

```

SELECT
    gs.movieId,
    gt.tag,
    gs.relevance,
    ROW_NUMBER() OVER (PARTITION BY gs.movieId ORDER BY gs.relevance DESC) AS
rn
FROM genome_scores gs
JOIN genome_tags gt ON gs.tagId = gt.tagId
)
SELECT movieId, tag, relevance
FROM ranked_tags
WHERE rn = 1;

```

#### Screenshot:

	movieid integer	tag character varying	relevance double precision
1	1	toys	0.9995
2	2	adventure	0.9737499999999999
3	3	good sequel	0.9870000000000001
4	4	women	0.96625
5	5	good sequel	0.9722500000000001
6	6	crime	0.9802500000000001
7	7	remake	0.98075
8	8	runaway	0.8545
9	9	action	0.9830000000000001
10	10	007 (series)	1
11	11	president	0.9962500000000001
Total rows: 16376		Query complete 00:00:18.821	

#### 6. List the top 5 users who rated the most number of movies.

##### Query:

```

SELECT userId, COUNT(*) AS total_ratings
FROM ratings
GROUP BY userId
ORDER BY total_ratings DESC
LIMIT 5;

```

#### Screenshot:

	userid integer	total_ratings bigint
1	189614	33332
2	48766	9554
3	207216	9178
4	175998	9016
5	76618	8919

## 7. Find movies that have not been rated by any user.

### Query:

```
SELECT m.movieId, m.title
FROM movies m
LEFT JOIN ratings r ON m.movieId = r.movieId
WHERE r.movieId IS NULL;
```

### Screenshot:

	movieid [PK] integer	title character varying
1	211307	Delta Delta Die! (2003)
2	158793	Allegro Barbaro (1979)
3	167214	Three Golden Serpents (1967)
4	133683	Mustang! (1959)
5	222238	Great Catherine (1968)
6	162538	Under Oath (1997)
7	170415	Young Dillinger (1965)
8	174989	Cosmopolitan (2003)
9	262597	First (2012)
10	237800	Entombed (2020)
11	237636	The Big Fat Lie (2018)
Total rows: 3298		Query complete 00:00:07.239

## 8. Using a CTE, find the average rating of each movie, and then filter for those with avg rating > 4.5.

### Query:

```
WITH avg_ratings AS (
    SELECT
        movieId,
        AVG(rating) AS avg_rating
    FROM ratings
    GROUP BY movieId
)
SELECT
    m.movieId,
    m.title,
    ROUND(a.avg_rating::numeric, 2) AS avg_rating
FROM avg_ratings a
JOIN movies m ON a.movieId = m.movieId
WHERE a.avg_rating > 4.5
ORDER BY a.avg_rating DESC;
```

### Screenshot:

	movieid [PK] integer	title character varying	avg_rating numeric
1	258245	Christmas In Evergreen: Tidings of Joy (2019)	5.00
2	281246	Anatomy of a Relationship (1976)	5.00
3	281362	Mariupolis 2 (2022)	5.00
4	281624	Hidden (2020)	5.00
5	281708	Come Back Anytime (2021)	5.00
6	281804	Only a Stonecutter (2008)	5.00
7	281846	Messidor (1979)	5.00
8	281876	Hanukkah on Rye (2022)	5.00
9	281956	ManFish (2022)	5.00
10	282021	Mr. Krueger's Christmas (1980)	5.00
11	282073	Mal de Ojo (2022)	5.00
Total rows: 1360		Query complete 00:00:03.876	

**9. Find the 5 most recently tagged movies (based on timestamp).**

**Query:**

```
SELECT
    t.movieId,
    m.title,
    t.tag,
    t.timestamp
FROM tags t
JOIN movies m ON t.movieId = m.movieId
ORDER BY t.timestamp DESC
LIMIT 5;
```

**Screenshot:**

	movieid integer	title character varying	tag character varying	timestamp bigint
1	72489	Ninja Assassin (2009)	JMS	1689841404
2	63062	Changeling (2008)	JMS	1689841295
3	104879	Prisoners (2013)	maze	1689840750
4	104879	Prisoners (2013)	candlelight vigil	1689840748
5	104879	Prisoners (2013)	snakes	1689840744

**10. Use a window function to rank all movies by average rating within their genres.**

**Query:**

```
WITH avg_ratings AS (
    SELECT
        movieId,
        AVG(rating) AS avg_rating
    FROM ratings
    GROUP BY movieId
),
split_genres AS (
    SELECT
```

```

        m.movieId,
        m.title,
        unnest(string_to_array(m.genres, '|')) AS genre,
        a.avg_rating
    FROM movies m
    JOIN avg_ratings a ON m.movieId = a.movieId
),
ranked_movies AS (
    SELECT
        movieId,
        title,
        genre,
        avg_rating,
        RANK() OVER (PARTITION BY genre ORDER BY avg_rating DESC) AS genre_rank
    FROM split_genres
)
SELECT *
FROM ranked_movies
ORDER BY genre, genre_rank
LIMIT 50;

```

#### Screenshot:

	movieid [PK] integer	title character varying	genre text	avg_rating double precision	genre_rank bigint
1	147615	The Battle of Love's Return (1971)	(no genres listed)	5	1
2	140121	Sam Steele and the Junior Detective Agency (2011)	(no genres listed)	5	1
3	137313	37 (2014)	(no genres listed)	5	1
4	178739	Kiss Me Kate (2003)	(no genres listed)	5	1
5	136876	The Village Had No Walls (1995)	(no genres listed)	5	1
6	140192	Targeting (2015)	(no genres listed)	5	1
7	177551	Victoria (2008)	(no genres listed)	5	1
8	136878	Paradh (2010)	(no genres listed)	5	1
9	159125	Anybody's Son Will Do	(no genres listed)	5	1
10	137222	Boppin' at the Glue Factory (2009)	(no genres listed)	5	1
11	167458	Vehshi Jatt (1975)	(no genres listed)	5	1
Total rows: 50		Query complete 00:00:04.152			

## Phase 3: Advanced Analysis

**1. Create a view that shows movie title, genre, average rating, and number of ratings.**

#### Query:

```

CREATE VIEW movie_summary_view AS
WITH avg_ratings AS (
    SELECT
        movieId,
        ROUND(AVG(rating)::numeric, 2) AS avg_rating,
        COUNT(*) AS rating_count

```

```

FROM ratings
GROUP BY movieId
),
split_genres AS (
    SELECT
        m.movieId,
        m.title,
        unnest(string_to_array(m.genres, '|')) AS genre
    FROM movies m
)
SELECT
    sg.title,
    sg.genre,
    ar.avg_rating,
    ar.rating_count
FROM split_genres sg
JOIN avg_ratings ar ON sg.movieId = ar.movieId;

```

```
SELECT * FROM movie_summary_view;
```

#### Screenshot:

	title character varying	genre text	avg_rating numeric	rating_count bigint
1	Toy Story (1995)	Adventure	3.89	76813
2	Toy Story (1995)	Animation	3.89	76813
3	Toy Story (1995)	Children	3.89	76813
4	Toy Story (1995)	Comedy	3.89	76813
5	Toy Story (1995)	Fantasy	3.89	76813
6	Jumanji (1995)	Adventure	3.28	30209
7	Jumanji (1995)	Children	3.28	30209
8	Jumanji (1995)	Fantasy	3.28	30209
9	Grumpier Old Men (1995)	Comedy	3.17	15820
10	Grumpier Old Men (1995)	Romance	3.17	15820
11	Waiting to Exhale (1995)	Comedy	2.87	3028
12	Waiting to Exhale (1995)	Drama	2.87	3028
13	Waiting to Exhale (1995)	Romance	2.87	3028
Total rows: 147230		Query complete 00:00:03.974		

**2. Write a stored function that accepts a genre and returns the top 3 rated movies in that genre.**

#### Query:

```

CREATE OR REPLACE FUNCTION topRatedMoviesByGenre(inputGenre TEXT)
RETURNS TABLE (
    title TEXT,
    avg_rating NUMERIC
)
AS $$
BEGIN
    RETURN QUERY

```

```

SELECT
    m.title::TEXT, -- Explicit cast to TEXT
    ROUND(AVG(r.rating)::NUMERIC, 2) AS avg_rating
FROM movies m
JOIN ratings r ON m.movieId = r.movieId
WHERE m.genres ILIKE '%' || input_genre || '%'
GROUP BY m.title
ORDER BY avg_rating DESC
LIMIT 3;
END;
$$ LANGUAGE plpgsql;

```

```
SELECT * FROM top_rated_movies_by_genre('Action');
```

**Screenshot:**

	title text	avg_rating numeric
1	97 Minutes (2023)	5.00
2	A Police Inspector Calls (1974)	5.00
3	Alien Girl (2010)	5.00

**3. Create a temporary table to store movies with an average rating above 4.0 and use it in a JOIN.**

**Query:**

```
-- Step 1: Create a temporary table with movies having avg rating > 4.0
```

```

CREATE TEMP TABLE high_rated_movies AS
SELECT
    movieId,
    ROUND(AVG(rating)::NUMERIC, 2) AS avg_rating
FROM ratings
GROUP BY movieId
HAVING AVG(rating) > 4.0;

```

```
-- Step 2: Join this temp table with movies to get titles and genres
```

```

SELECT
    m.title,
    m.genres,
    h.avg_rating
FROM high_rated_movies h
JOIN movies m ON h.movieId = m.movieId
ORDER BY h.avg_rating DESC
LIMIT 10;

```

**Screenshot:**

	title character varying	genres text	avg_rating numeric
1	London Conspiracy (1974)	Action Mystery	5.00
2	Bronx Obama (2014)	Documentary	5.00
3	Latin Music USA (2009)	Documentary Musical	5.00
4	Woman's Tale, A (1991)	Drama	5.00
5	Deadly Delicious (Shuang Shi Ji) (2008)	Drama Mystery	5.00
6	Pursuit of Unhappiness, The (Anleitung zum Unglücklichsein) (2012)	Romance	5.00
7	Always a Bridesmaid (2000)	Documentary	5.00
8	As Seen Through These Eyes (2008)	Documentary	5.00
9	Parasites, Les (1999)	Comedy	5.00
10	The Secret Country: The First Australians Fight Back (1986)	Documentary	5.00

#### 4. Create a materialized view for movieId and its most relevant tag from genome\_scores.

##### Query:

```
CREATE MATERIALIZED VIEW most_relevant_tag_per_movie AS
SELECT
    gs.movieId,
    gt.tag,
    gs.relevance
FROM genome_scores gs
JOIN genome_tags gt ON gs.tagId = gt.tagId
WHERE (gs.movieId, gs.relevance) IN (
    SELECT movieId, MAX(relevance)
    FROM genome_scores
    GROUP BY movieId
);
```

```
SELECT *
FROM most_relevant_tag_per_movie
LIMIT 20;
```

##### Screenshot:



	movieid integer	tag character varying	relevance double precision
1	1	toys	0.9995
2	11	president	0.9962500000000001
3	12	spoof	0.99275
4	13	animation	0.9884999999999999
5	14	biographical	0.9797499999999999
6	15	swashbuckler	0.998
7	15	treasure hunt	0.998
8	16	organized crime	0.9990000000000001
9	17	adapted from:book	0.9764999999999999
10	18	off-beat comedy	0.94
11	19	comedy	0.9842500000000001
Total rows: 20		Query complete 00:00:00.104	

## 5. Use EXPLAIN ANALYZE to evaluate the performance of a JOIN query between ratings and movies.

### Query:

```
EXPLAIN ANALYZE
SELECT
    m.title,
    r.rating
FROM
    ratings r
JOIN
    movies m ON r.movieId = m.movieId
LIMIT 100;
```

### Screenshot:

	QUERY PLAN text
1	Limit (cost=0.30..4.45 rows=100 width=30) (actual time=12.199..12.518 rows=100 loops=1)
2	-> Nested Loop (cost=0.30..1401981.97 rows=33834640 width=30) (actual time=12.197..12.506 rows=100 loops=1)
3	-> Seq Scan on ratings r (cost=0.00..553882.40 rows=33834640 width=8) (actual time=0.027..0.035 rows=100 loops=1)
4	-> Memoize (cost=0.30..0.32 rows=1 width=30) (actual time=0.124..0.124 rows=1 loops=100)
5	Cache Key: r.movieid
6	Cache Mode: logical
7	Hits: 3 Misses: 97 Evictions: 0 Overflows: 0 Memory Usage: 13kB
8	-> Index Scan using movies_pkey on movies m (cost=0.29..0.31 rows=1 width=30) (actual time=0.003..0.003 rows=1...
9	Index Cond: (movieid = r.movieid)
10	Planning Time: 0.383 ms
11	Execution Time: 12.570 ms

## 6. Find users who have rated both "Toy Story (1995)" and "Jumanji (1995)".

### Query:


```
SELECT userId
```

```

FROM ratings
WHERE movieId IN (
    SELECT movieId FROM movies WHERE title IN ('Toy Story (1995)', 'Jumanji
(1995)')
)
GROUP BY userId
HAVING COUNT(DISTINCT movieId) = 2;

```

**Screenshot:**

	userid 
1	2
2	14
3	21
4	51
5	72
6	79
7	82
8	148
9	149
10	174
11	177
12	198
13	210
Total rows: 19366	

**7. Normalize genres (currently pipe-separated) into a new table with one row per genre per movie (using string functions or unnest).**

**Query:**

```

CREATE TABLE movie_genres (
    movieId INT,
    genre TEXT
);

INSERT INTO movie_genres (movieId, genre)
SELECT
    movieId,
    TRIM(unnest(string_to_array(genres, '|'))) AS genre
FROM movies
WHERE genres IS NOT NULL;

SELECT *

```

```
FROM movie_genres
```

```
LIMIT 20;
```

**Screenshot:**

	movieid integer	genre text
1	1	Adventure
2	1	Animation
3	1	Children
4	1	Comedy
5	1	Fantasy
6	2	Adventure
7	2	Children
8	2	Fantasy
9	3	Comedy
10	3	Romance
11	4	Comedy
12	4	Drama
13	4	Romance
Total rows: 20		Query complete 00:00:00.043

**8. Generate a report using a recursive CTE: all tags that include the substring 'war' and related tagIds.**

**Query:**

```
WITH RECURSIVE war_tags_cte AS (  
    -- Anchor member: all tags containing 'war'  
    SELECT tagId, tag  
    FROM genome_tags  
    WHERE tag ILIKE '%war%'  
  
    UNION ALL  
  
    -- Recursive member: pretend we're expanding similar tags (for demo, assume  
    same word prefix)  
    SELECT gt.tagId, gt.tag  
    FROM genome_tags gt  
    JOIN war_tags_cte wt ON gt.tag ILIKE wt.tag || '%'  
    WHERE gt.tagId != wt.tagId  
)  
SELECT DISTINCT *  
FROM war_tags_cte  
ORDER BY tagId;
```

**Screenshot:**

	tagid integer	tag character varying
1	56	american civil war
2	69	anti-war
3	127	best war films
4	214	civil war
5	226	cold war
6	441	global warming
7	480	gulf war
8	497	heartwarming
9	561	iraq war
10	724	nuclear war
11	880	saturn award (best science fiction film)
12	881	saturn award (best special effects)
13	949	spanish civil war
Total rows: 20		Query complete 00:00:00.060

**9. Create a user-defined function that returns the number of ratings a movie received above a given threshold.**

**Query:**

```
CREATE OR REPLACE FUNCTION count_high_ratings(movie_title TEXT, threshold FLOAT)
RETURNS INTEGER AS $$
DECLARE
    rating_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO rating_count
    FROM ratings r
    JOIN movies m ON r.movieId = m.movieId
    WHERE m.title = movie_title AND r.rating > threshold;

    RETURN rating_count;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT count_high_ratings('Toy Story (1995)', 4.0);
```

**Screenshot:**

	count_high_ratings integer
1	25417

**10. Simulate a transaction: insert a new rating, then rollback to demonstrate ACID behavior.**

**Query:**

```
-- Step 1: Start the transaction
BEGIN;
```

```
-- Step 2: Insert a new rating (simulate a user rating a movie)
```

```
INSERT INTO ratings (userId, movieId, rating, timestamp)
VALUES (9999, 1, 4.5, EXTRACT(EPOCH FROM NOW())); -- 1 = Toy Story (1995)
```

```
-- Step 3: Check that the data is inserted (run a SELECT)
```

```
SELECT * FROM ratings
```

```
WHERE userId = 9999 AND movieId = 1;
```

```
-- Step 4: Rollback the transaction
```

```
ROLLBACK;
```

```
-- Step 5: Verify rollback (should return 0 rows)
```

```
SELECT * FROM ratings
```

```
WHERE userId = 9999 AND movieId = 1;
```

**Screenshot:**

	userid	movieid	rating	timestamp
	integer	integer	real	bigint