

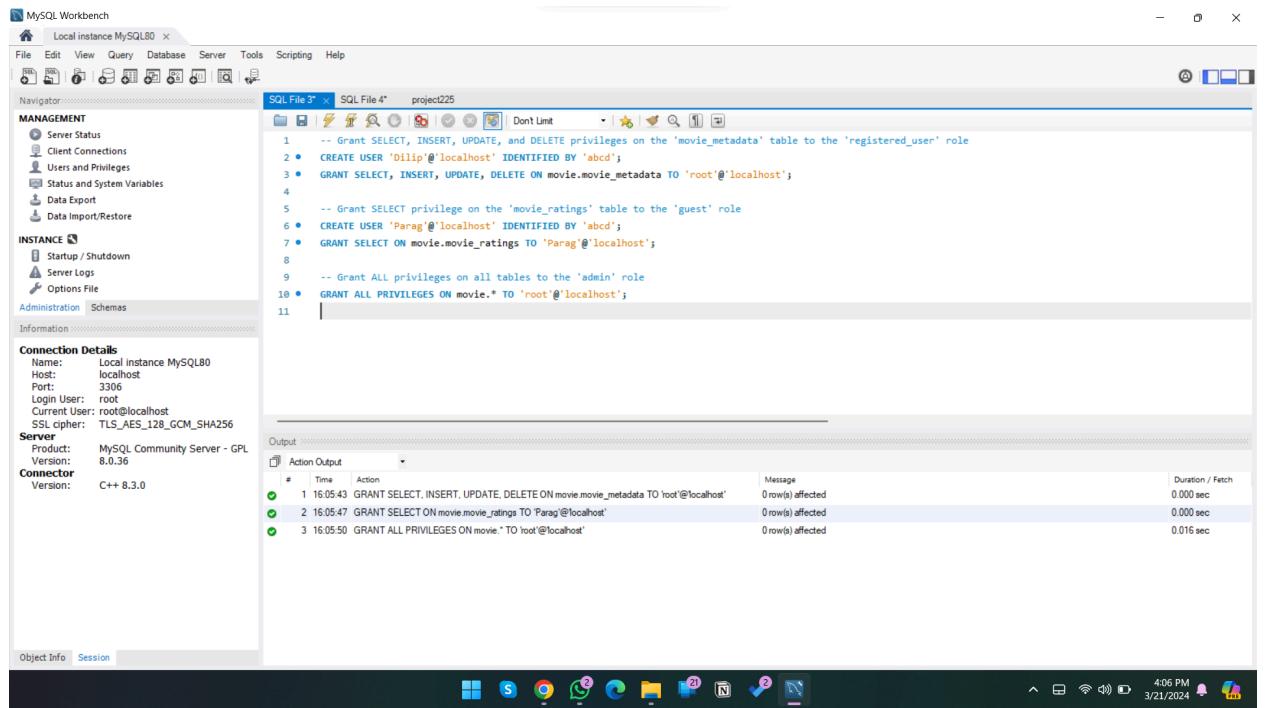
SQL Queries And Output

Lab - 1

- Access Privileges:

```
-- Grant SELECT, INSERT, UPDATE, and DELETE privileges on the 'movie_metadata' table to 'Dilip'  
CREATE USER 'Dilip'@'localhost' IDENTIFIED BY 'abcd';  
GRANT SELECT, INSERT, UPDATE, DELETE ON movie.movie_metadata TO 'Dilip'@'localhost';  
  
-- Grant SELECT privilege on the 'movie_ratings' table to user 'Parag'  
CREATE USER 'Parag'@'localhost' IDENTIFIED BY 'abcd';  
GRANT SELECT ON movie.movie_ratings TO 'Parag'@'localhost';  
  
-- Grant ALL privileges on all tables to the 'admin' role  
GRANT ALL PRIVILEGES ON movie.* TO 'root'@'localhost';
```

Output:



The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The main window has two tabs: "SQL File 3*" and "SQL File 4*", both containing the same SQL queries listed above. Below the tabs is a toolbar with various icons. To the left is a "Navigator" pane with sections for MANAGEMENT (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), INSTANCE (Startup / Shutdown, Server Logs, Options File), Administration, Schemas, and Information. The bottom section is the "Output" pane, which displays the results of the executed queries. The results table has columns for # (row number), Time (execution time), Action (query text), Message (status message), and Duration / Fetch (execution duration). The output shows three rows corresponding to the queries: Row 1: GRANT SELECT, INSERT, UPDATE, DELETE ON movie.movie_metadata TO 'root'@'localhost'; 0 row(s) affected, Duration 0.000 sec. Row 2: GRANT SELECT ON movie.movie_ratings TO 'Parag'@'localhost'; 0 row(s) affected, Duration 0.000 sec. Row 3: GRANT ALL PRIVILEGES ON movie.* TO 'root'@'localhost'; 0 row(s) affected, Duration 0.016 sec.

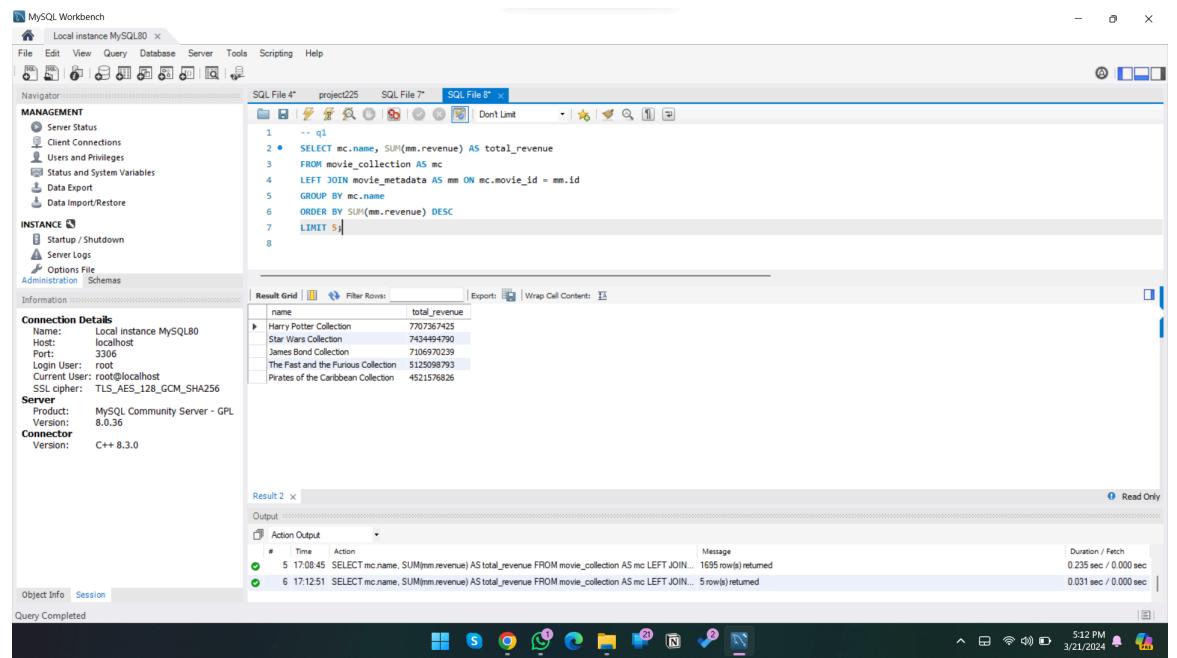
#	Time	Action	Message	Duration / Fetch
1	16:05:43	GRANT SELECT, INSERT, UPDATE, DELETE ON movie.movie_metadata TO 'root'@'localhost'	0 row(s) affected	0.000 sec
2	16:05:47	GRANT SELECT ON movie.movie_ratings TO 'Parag'@'localhost'	0 row(s) affected	0.000 sec
3	16:05:50	GRANT ALL PRIVILEGES ON movie.* TO 'root'@'localhost'	0 row(s) affected	0.016 sec

- SQL Code Queries:

1. Get the top 5 movies with total revenue for each movie

```
SELECT mc.name, SUM(mm.revenue) AS total_revenue
FROM movie_collection AS mc
LEFT JOIN movie_metadata AS mm ON mc.movie_id = mm.id
GROUP BY mc.name
ORDER BY SUM(mm.revenue) DESC
LIMIT 5;
```

Output:



The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Local instance MySQL80, Management, INSTANCE, Administration, Schemas.
- SQL Editor:** Contains the SQL query:

```
-- q1
SELECT mc.name, SUM(mm.revenue) AS total_revenue
FROM movie_collection AS mc
LEFT JOIN movie_metadata AS mm ON mc.movie_id = mm.id
GROUP BY mc.name
ORDER BY SUM(mm.revenue) DESC
LIMIT 5;
```
- Result Grid:** Shows the results of the query:

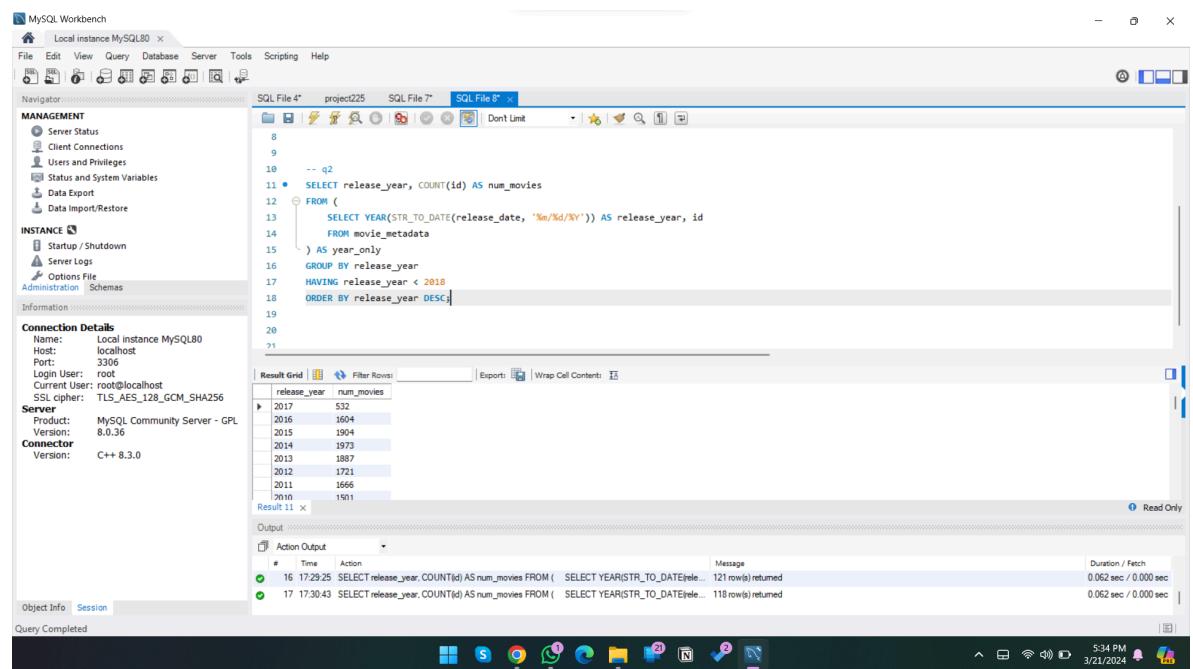
name	total_revenue
Harry Potter Collection	7707367425
Star Wars Collection	743494790
James Bond Collection	7106970239
The Fast and the Furious Collection	5125096793
Pirates of the Caribbean Collection	4521576826
- Action Output:** Shows the log of actions taken:

#	Time	Action	Message	Duration / Fetch
5	17:08:45	SELECT mc.name, SUM(mm.revenue) AS total_revenue FROM movie_collection AS mc LEFT JOIN...	1695 row(s) returned	0.235 sec / 0.000 sec
6	17:12:51	SELECT mc.name, SUM(mm.revenue) AS total_revenue FROM movie_collection AS mc LEFT JOIN...	5 row(s) returned	0.031 sec / 0.000 sec

2. Count the number of movies released each year before 2018 and order by release year in descending.

```
SELECT release_year, COUNT(id) AS num_movies
FROM (
    SELECT YEAR(STR_TO_DATE(release_date, '%m/%d/%Y')) AS
        release_year, id
    FROM movie_metadata
) AS year_only
GROUP BY release_year
HAVING release_year < 2018
ORDER BY release_year DESC;
```

Output:



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the current connection is "Local instance MySQL80".
- SQL Editor:** Contains the query provided in the previous code block.
- Result Grid:** Displays the results of the query, showing the count of movies released each year from 2017 down to 2001.
- Output:** Shows the log of actions taken, including the execution of the query at 17:29:25 and 17:30:43.

release_year	num_movies
2017	532
2016	1604
2015	1904
2014	1973
2013	1887
2012	1721
2011	1666
2010	1501

3. Get the movies in which the spoken languages is korean

```
SELECT m.title  
FROM movie_metadata m  
JOIN movie_spoken_language s ON m.id = s.movie_id  
WHERE s.iso_639_1 = 'ko';
```

Output:

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays various management and instance-related sections like Server Status, Client Connections, and Data Export. The Connection Details section shows the connection is to Local instance MySQL80, Host is localhost, Port is 3306, and Current User is root@localhost. The SQL Editor tab contains the query:

```
19  
20  
21 -- q3  
22 • SELECT m.title  
23   FROM movie_metadata m  
24   JOIN movie_spoken_language s ON m.id = s.movie_id  
25 WHERE s.iso_639_1 = "ko";
```

The Result Grid pane shows the output of the query:

title
Eye for an Eye
Die Hard
301, 302
Deep Rising
The Rescuers Down Under
The City
Falling Down
House to Hide
Training Day
Mystic River

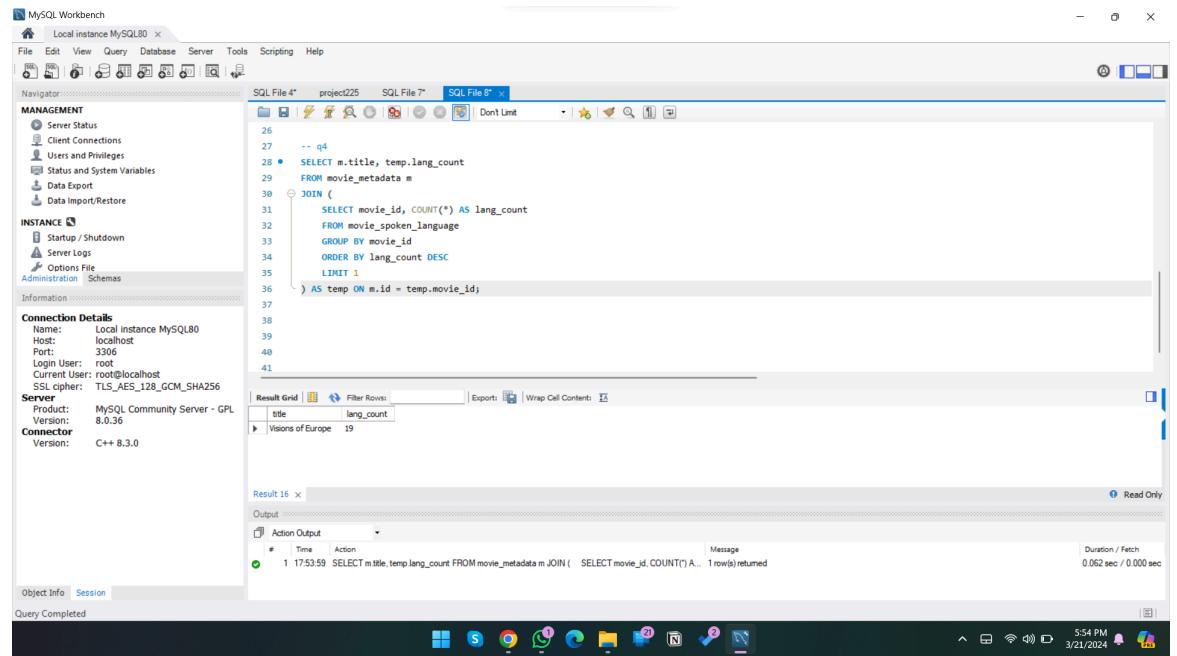
The Output pane at the bottom shows the log entries for the query execution:

#	Time	Action	Message	Duration / Fetch
18	17:41:58	Action	SELECT title FROM movie_metadata WHERE id IN (SELECT movie_id FROM movie_spoken_lang... 28727 row(s) returned)	0.312 sec / 0.000 sec
19	17:47:31	Action	SELECT m.title FROM movie_metadata m JOIN movie_spoken_language s ON m.id = s.movie_id W... 542 row(s) returned	0.031 sec / 0.000 sec

4. Select the movie which has highest number of spoken languages

```
SELECT m.title, temp.lang_count
FROM movie_metadata m
JOIN (
    SELECT movie_id, COUNT(*) AS lang_count
    FROM movie_spoken_language
    GROUP BY movie_id
    ORDER BY lang_count DESC
    LIMIT 1
) AS temp ON m.id = temp.movie_id;
```

Output:



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** MANAGEMENT (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), INSTANCE (Startup / Shutdown, Server Logs, Options File), Administration, Schemas.
- Connection Details:** Name: Local instance MySQL80, Host: localhost, Port: 3306, Login User: root, Current User: root@localhost, SSL cipher: TLS_AES_128_GCM_SHA256.
- Server:** Product: MySQL Community Server - GPL, Version: 8.0.36, Connector: C++ 8.3.0.
- SQL Editor:** SQL File 4* (containing the query), SQL File 7*.
- Result Grid:** title | lang_count
Visions of Europe | 19
- Action Output:** 1 17:53:59 SELECT m.title,temp.lang_count FROM movie_metadata m JOIN (SELECT movie_id, COUNT(*) AS lang_count FROM movie_spoken_language GROUP BY movie_id ORDER BY lang_count DESC LIMIT 1) AS temp ON m.id = temp.movie_id; 1 row(s) returned
- System Bar:** Read Only, Duration / Fetch 0.062 sec / 0.000 sec, 5:54 PM, 3/21/2024.

5. Select movie with highest number of cast members

```
SELECT title, crew_count
FROM movie_metadata
JOIN (
    SELECT movie_id, COUNT(*) AS crew_count
    FROM crew
    GROUP BY movie_id
    ORDER BY crew_count DESC
    LIMIT 1
) AS temp ON movie_metadata.id = temp.movie_id;
```

Output:

The screenshot shows the MySQL Workbench interface. The left sidebar displays management and instance details, including connection information for 'Local instance MySQL80' (Host: localhost, Port: 3306, User: root). The main area shows the SQL editor with the query from step 5. The result grid shows one row: 'Jurassic World' with a crew count of 455. The output pane at the bottom shows the query was executed at 17:59:28 and returned 1 row(s).

title	crew_count
Jurassic World	455

```
# Time Action Message Duration / Fetch
1 17:59:28 SELECT title, crew_count FROM movie_metadata JOIN (    SELECT movie_id, COUNT(*) AS crew_count    FROM crew    GROUP BY movie_id    ORDER BY crew_count DESC    LIMIT 1  ) AS temp ON movie_metadata.id = temp.movie_id; 1 row(s) returned 0.235 sec / 0.000 sec
```

6. Retrieve the previous, current, and next movie titles based on their popularity.

```
SELECT * FROM (SELECT
    LAG(title) OVER (ORDER BY popularity) AS previous_movie,
    title AS current_movie,
    LEAD(title) OVER (ORDER BY popularity) AS next_movie
  FROM
    movie_metadata) AS subquery
WHERE previous_movie IS NOT NULL AND current_movie IS NOT NULL
AND next_movie IS NOT NULL;
```

Output:

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
SELECT * FROM (SELECT
    LAG(title) OVER (ORDER BY popularity) AS previous_movie,
    title AS current_movie,
    LEAD(title) OVER (ORDER BY popularity) AS next_movie
  FROM
    movie_metadata) AS subquery
WHERE previous_movie IS NOT NULL AND current_movie IS NOT NULL
AND next_movie IS NOT NULL;
```

The results grid displays the following data:

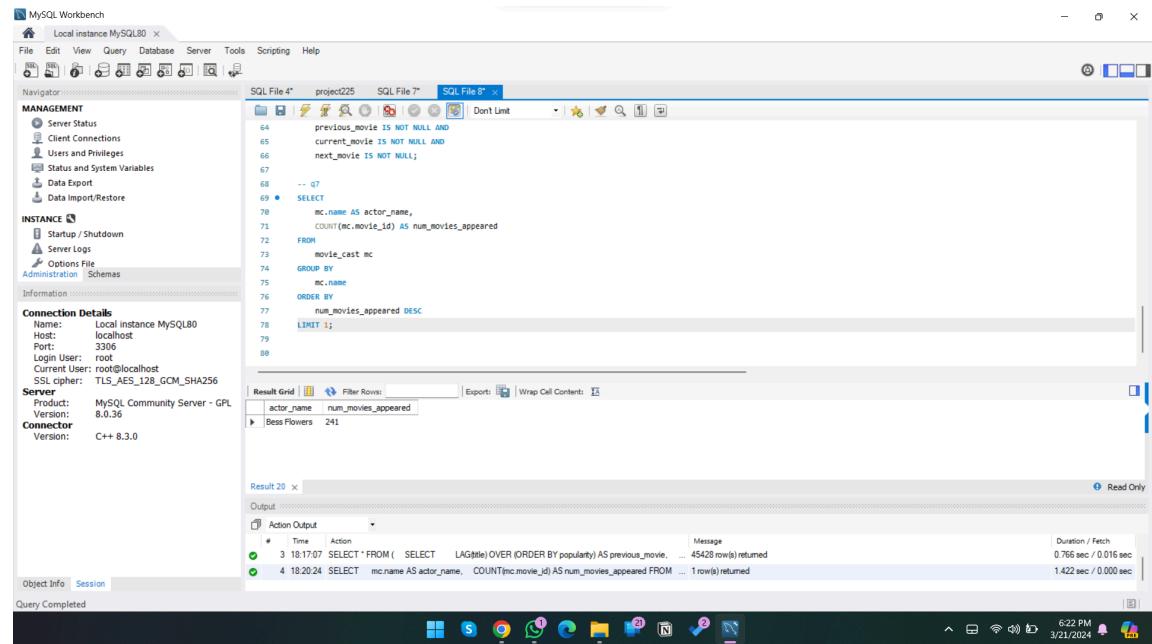
previous_movie	current_movie	next_movie
Pleasure Party	Business as Usual	Double Game
Business as Usual	Double Game	The Horseplayer
Double Game	The Horseplayer	Bling: A Planet Rock
The Horseplayer	Bling: A Planet Rock	The Castle of Clouds
Bling: A Planet Rock	The Castle of Clouds	Allende en su laberinto

The status bar at the bottom right shows the date and time: 6:17 PM 3/21/2024.

7. Identify the actor/actress who have appeared in the most movies

```
SELECT
    mc.name AS actor_name,
    COUNT(mc.movie_id) AS num_movies_appeared
FROM
    movie_cast mc
GROUP BY
    mc.name
ORDER BY
    num_movies_appeared DESC
LIMIT 1;
```

Output:



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
64     previous_movie IS NOT NULL AND
65     current_movie IS NOT NULL AND
66     next_movie IS NOT NULL;
67
68 -- q7
69 • SELECT
70     mc.name AS actor_name,
71     COUNT(mc.movie_id) AS num_movies_appeared
72     FROM
73     movie_cast mc
74     GROUP BY
75     mc.name
76     ORDER BY
77     num_movies_appeared DESC
78     LIMIT 1;
79
80
```

The Result Grid shows the output:

actor_name	num_movies_appeared
Bess Flowers	241

The Output pane shows the execution log:

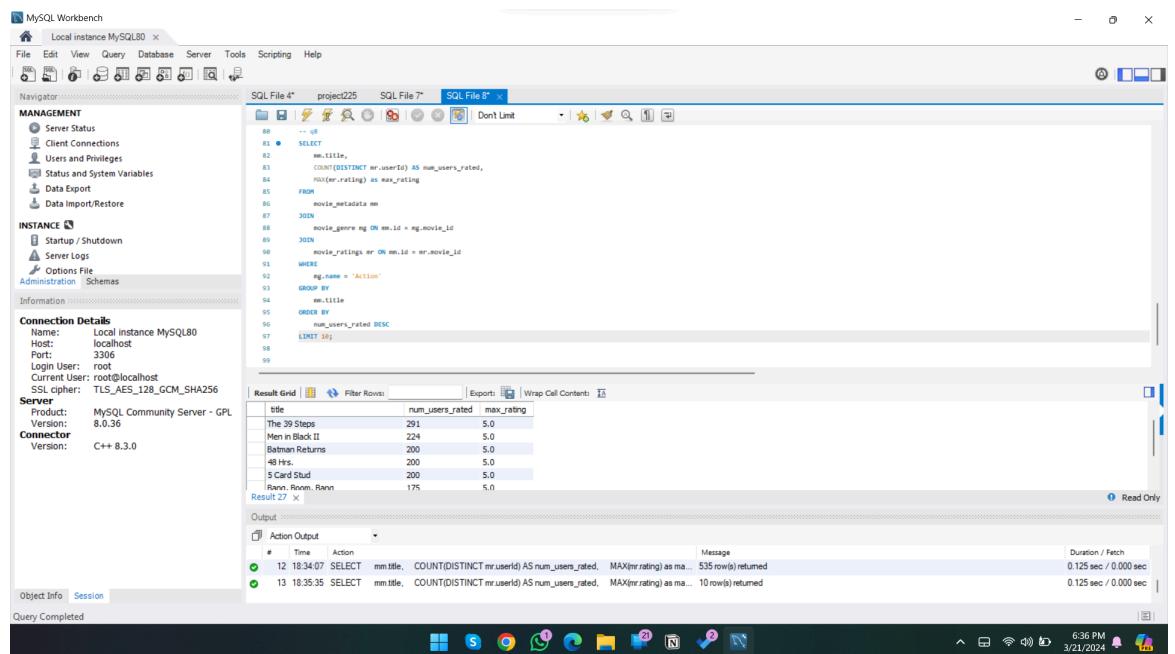
```
3 18:17:07 SELECT * FROM ( SELECT LAG(title) OVER (ORDER BY popularity) AS previous_movie, ... 45428 row(s) returned
4 18:20:24 SELECT mc.name AS actor_name, COUNT(mc.movie_id) AS num_movies_appeared FROM ... 1 row(s) returned
```

8. Top 10 movies with highest rating with most numbers of users who have rated the movie

```

SELECT
    mm.title,
    COUNT(DISTINCT mr.userId) AS num_users_rated,
    MAX(mr.rating) AS max_rating
FROM
    movie_metadata mm
JOIN
    movie_genre mg ON mm.id = mg.movie_id
JOIN
    movie_ratings mr ON mm.id = mr.movie_id
WHERE
    mg.name = 'Action'
GROUP BY
    mm.title
ORDER BY
    num_users_rated DESC
  
```

Output:



The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** MANAGEMENT (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), INSTANCE (Startup / Shutdown, Server Logs, Options File, Administration, Schemas), Information.
- Connection Details:**
 - Name: Local instance MySQL80
 - Host: localhost
 - Port: 3306
 - Login User: root
 - Current User: root@localhost
 - SSL cipher: TLS_AES_128_GCM_SHA256
 - Server:
 - Product: MySQL Community Server - GPL
 - Version: 8.0.36
 - Connector:
 - Version: C++ 8.3.0
- SQL Editor:** SQL File 4*, project225, SQL File 7*, SQL File 8*. The query is pasted here.
- Result Grid:** Shows the results of the executed query. The columns are title, num_users_rated, and max_rating. The data is as follows:

title	num_users_rated	max_rating
The 39 Steps	291	5.0
Men in Black II	224	5.0
Batman Returns	200	5.0
48 Hrs.	200	5.0
S Card Stud	200	5.0
Ran, Ran, Ran!	175	5.0

- Output:** Action Output shows two log entries:
 - 12 18:34:07 SELECT mm.title, COUNT(DISTINCT mr.userId) AS num_users_rated, MAX(mr.rating) AS ma... 535 row(s) returned
 - 13 18:35:35 SELECT mm.title, COUNT(DISTINCT mr.userId) AS num_users_rated, MAX(mr.rating) AS ma... 10 row(s) returned
- Status Bar:** Read Only, Duration / Fetch: 0.125 sec / 0.000 sec, 6:36 PM, 3/21/2024.

9. Display movies having greater than 2 collections

```
SELECT
    mm.title AS movie_title,
    COUNT(mc.collection_id) AS num_collections
FROM
    movie_metadata mm
JOIN
    movie_collection mc ON mm.id = mc.movie_id
GROUP BY
    mm.title
HAVING
    num_collections > 2
ORDER BY
    num_collections DESC;
```

Output:

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
SELECT
    mm.title AS movie_title,
    COUNT(mc.collection_id) AS num_collections
FROM
    movie_metadata mm
JOIN
    movie_collection mc ON mm.id = mc.movie_id
GROUP BY
    mm.title
HAVING
    num_collections > 2
ORDER BY
    num_collections DESC;
```

The results grid shows one row:

movie_title	num_collections
The Mummy	3

The status bar at the bottom right indicates the session was completed at 6:47 PM on 3/21/2024.

10. Display the total number of movies each country has released.

```
SELECT
    pc.name AS production_country,
    COUNT(mm.id) AS num_movies_released
FROM
    movie_production_countries pc
JOIN
    movie_metadata mm ON pc.movie_id = mm.id
GROUP BY
    pc.name
ORDER BY
    num_movies_released DESC;
```

Output:

The screenshot shows the MySQL Workbench interface. The left pane displays the Navigator with various schemas and objects. The central pane contains the SQL editor with the query from step 10. The results are shown in a grid below, listing countries and their movie counts. The bottom pane shows the output log with two entries corresponding to the query execution.

production_country	num_movies_released
United States of America	21153
United Kingdom	4094
France	3940
Germany	2254
Italy	2169
Canada	1745

Output Log:

#	Time	Action	Message	Duration / Fetch
16	18:53:03	SELECT pc.name AS production_country, COUNT(mm.id) AS num_movies_released FROM movie_production_countries pc JOIN movie_metadata mm ON pc.movie_id = mm.id GROUP BY pc.name ORDER BY num_movies_released DESC;	Error Code: 1054. Unknown column 'num_movies_released' in 'order clause'	0.000 sec
17	18:53:37	SELECT pc.name AS production_country, COUNT(mm.id) AS num_movies_released FROM movie_production_countries pc JOIN movie_metadata mm ON pc.movie_id = mm.id GROUP BY pc.name ORDER BY num_movies_released DESC;	160 row(s) returned	0.203 sec / 0.000 sec

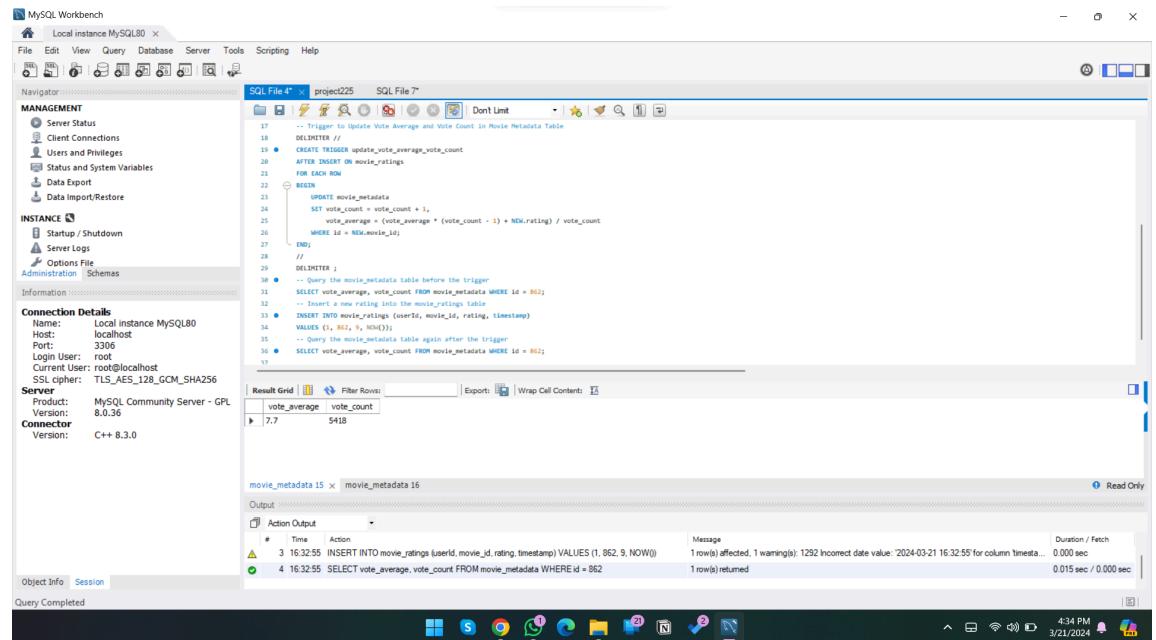
- Triggers:
 1. Trigger to Update Vote Average and Vote Count in Movie Metadata Table.

```

DELIMITER //
CREATE TRIGGER update_vote_average_vote_count
AFTER INSERT ON movie_ratings
FOR EACH ROW
BEGIN
  UPDATE movie_metadata
  SET vote_count = vote_count + 1,
      vote_average = (vote_average * (vote_count - 1) + NEW.rating) / vote_count
  WHERE id = NEW.movie_id;
END;
//
DELIMITER ;
-- Query the movie_metadata table before the trigger
SELECT vote_average, vote_count FROM movie_metadata WHERE id = 862;
-- Insert a new rating into the movie_ratings table
INSERT INTO movie_ratings (userId, movie_id, rating, timestamp)
VALUES (1, 862, 9, NOW());
-- Query the movie_metadata table again after the trigger
SELECT vote_average, vote_count FROM movie_metadata WHERE id = 862;

```

Output:



- Design a trigger that should activate after each update operation on the movie_ratings table. Upon update of a rating record, it should print the timestamp of the update, the action performed (update), the name of the affected table (movie_ratings), and details about the updated record, including its ID, the old rating, and the new rating. Implement this trigger to directly display the changes for each update operation. (Includes Logging)

```
DELIMITER //

CREATE TRIGGER ratings_update_trigger
AFTER UPDATE ON movie_ratings
FOR EACH ROW
BEGIN
    INSERT INTO rating_changes (ratings_id, old_rating, new_rating,
change_timestamp)
    VALUES (OLD.ratings_id, OLD.rating, NEW.rating, NOW());
END;
//

DELIMITER ;

INSERT INTO movie_ratings (userId, movie_id, rating, timestamp) VALUES
(2, 1234, 4.5, NOW());

SELECT userId, movie_id, rating FROM movie_ratings
WHERE userId = 2 AND movie_id = 1234;

UPDATE movie_ratings SET rating = 5.0 WHERE userId = 2 AND movie_id
= 1234;

SELECT userId, movie_id, rating FROM movie_ratings
WHERE userId = 2 AND movie_id = 1234;
```

Output Before Update:

```
DELIMITER //

CREATE TRIGGER rating_update_trigger
AFTER UPDATE ON movie_ratings
FOR EACH ROW
BEGIN
    INSERT INTO rating_changes (rating_id, old_rating, new_rating, change_timestamp)
    VALUES (OLD.rating_id, OLD.rating, NEW.rating, NOW());
END;

DELIMITER ;
```

Table: movie_ratings

Columns:

- ratings_id int AI PK
- user_id int
- movie_id int
- rating decimal(3,1)
- timestamp date

Result Grid

user_id	movie_id	rating
2	1234	4.5

Object Info Session

Output

Action Output

#	Time	Action	Message	Duration / Fetch
12	10:45:34	INSERT INTO movie_ratings (user_id, movie_id, rating, timestamp) VALUES (2, 1234, 4.5, NOW())	1 row(s) affected, 1 warning(s); 1292 Incorrect date value: '2024-03-22 10:45:34' for column timestamp	0.031 sec
13	10:45:38	SELECT user_id, movie_id, rating FROM movie_ratings WHERE user_id = 2 AND movie_id = 1234	1 row(s) returned	0.000 sec / 0.000 sec

Output After Update:

```
DELIMITER //

CREATE TRIGGER rating_update_trigger
AFTER UPDATE ON movie_ratings
FOR EACH ROW
BEGIN
    INSERT INTO rating_changes (rating_id, old_rating, new_rating, change_timestamp)
    VALUES (OLD.rating_id, OLD.rating, NEW.rating, NOW());
END;

DELIMITER ;
```

Table: movie_ratings

Columns:

- ratings_id int AI PK
- user_id int
- movie_id int
- rating decimal(3,1)
- timestamp date

Result Grid

user_id	movie_id	rating
2	1234	5.0

Object Info Session

Output

Action Output

#	Time	Action	Message	Duration / Fetch
16	10:46:49	UPDATE movie_ratings SET rating = 5.0 WHERE user_id = 2 AND movie_id = 1234	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.157 sec
17	10:46:54	SELECT user_id, movie_id, rating FROM movie_ratings WHERE user_id = 2 AND movie_id = 1234	1 row(s) returned	0.031 sec / 0.000 sec

3. Design a trigger in MySQL for the provided database schema to monitor the movie_links table. The trigger should activate after each insertion operation on the movie_links table. Upon insertion of a new movie link record, it should print a message indicating the successful insertion. (Includes Logging)

```

DELIMITER //
CREATE TRIGGER links_trig
AFTER INSERT ON movie_links
FOR EACH ROW
BEGIN
    INSERT INTO trigger_log_links_trig (timestamp, message)
    VALUES (NOW(), 'New movie link record inserted successfully.');
END;
//
DELIMITER ;
INSERT INTO movie_links (movie_id, imdbId, tmdbId) VALUES (999123, 123456, 789012);
SELECT * FROM trigger_log_links_trig;

```

Output:

The screenshot shows the MySQL Workbench interface with several tabs open. The 'SQL' tab contains the trigger definition and the insert statement. The 'Result Grid' tab shows the log entry from the trigger table. The 'Action Output' tab shows the history of actions taken.

```

Table: movie_links
Columns:
movie_id int PK
imdbId int
tmdbId int

trigger_log_links_trig
id timestamp message
1 2024-03-22 11:18:00 New movie link record inserted successfully.

```

#	Time	Action	Message	Duration / Fetch
1	11:18:00	INSERT INTO movie_links (movie_id, imdbId, tmdbId) VALUES (999123, 123456, 789012)	1 row(s) affected	0.000 sec
2	11:18:03	SELECT * FROM trigger_log_links_trig	1 row(s) returned	0.000 sec / 0.000 sec

4. Design a trigger in MySql for adding a new production house into the Production_name table. A production_name_log table is created for logging a new production house and the records will be stored in the log table, An example is taken where ‘Sanjana Productions’ is the table inserted into the Production_name table. This record is stored in the production_name_log as shown in the figure.

use movie;

```

DROP TABLE IF EXISTS prod_updates_log;
CREATE TABLE IF NOT EXISTS prod_updates_log (
log_id INT AUTO_INCREMENT PRIMARY KEY,
prod_id INT,
prod_name VARCHAR(255),
inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
DELIMITER //
drop trigger if exists new_production_company //
CREATE TRIGGER new_production_company
AFTER INSERT ON production_companies
FOR EACH ROW
BEGIN
INSERT INTO prod_updates_log (prod_id, prod_name)
VALUES (NEW.id, NEW.name);
END//
DELIMITER ;
insert into production_companies(movie_id, name, id) values ('17584645', 'Sanjana Productions' ,
'123123');
select * from prod_updates_log;

```

Output:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** movie_catalogue_queries
- Query Editor:** Contains the SQL code for creating the trigger and inserting data into the production_companies table.
- Result Grid:** Shows the output of the query, including the insertion of a new row into the production_companies table and the corresponding log entry in the prod_updates_log table.
- Information:** Shows the structure of the movie_collection table.
- Action Output:** Displays the log of actions performed, including the creation of the trigger and the insertion of the company record.

log_id	prod_id	prod_name	inserted_at
1	123123	Sanjana Productions	2024-03-22 17:36:42

#	Time	Action	Message	Duration / Fetch
78	17:35:24	CREATE TRIGGER new_production_company AFTER INSERT ON production_companies FOR EACH	0 rows(a)ffected	0.016 sec
79	17:35:42	insert into production_companies(movie_id, name, id) values ('17584645', 'Sanjana Productions', '123123')	Error Code: 1136. Column count doesn't match value count at row 1	0.000 sec
80	17:36:36	drop trigger if exists new_production_company	0 rows(a)ffected	0.015 sec
81	17:36:36	CREATE TRIGGER new_production_company AFTER INSERT ON production_companies FOR EACH	0 rows(a)ffected	0.000 sec
82	17:36:42	insert into production_companies(movie_id, name, id) values ('17584645', 'Sanjana Productions', '123123')	1 rows(a)ffected	0.000 sec
83	17:36:42	select * from prod_updates_log LIMIT 0, 1000	1 rows(s) returned	0.000 sec / 0.000 sec

5. To keep track of deletions from the production_companies table, create the log table prod_delete_log. Additionally, it defines a trigger called production_companies_delete_trigger, which will cause deletions from production_companies to be automatically logged into the log table anytime a record is removed. Finally, it shows how to run a deletion statement to remove a particular entry from the production_companies database and read the deletion log, as well as how to deactivate SQL-safe updates.

```

CREATE TABLE IF NOT EXISTS prod_delete_log (
log_id INT AUTO_INCREMENT PRIMARY KEY,
production_id INT,
production_name VARCHAR(255),
deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
DELIMITER //
drop trigger if exists production_companies_delete_trigger //
CREATE TRIGGER production_companies_delete_trigger
AFTER DELETE ON production_companies
FOR EACH ROW
BEGIN
INSERT INTO prod_delete_log (production_id, production_name)
VALUES (old.id, old.name);
END//
DELIMITER ;
set sql_safe_updates = 0;
DELETE FROM production_companies
WHERE movie_id = '17584645' AND name = 'Sanjana Productions' AND id = '74586';
select * from prod_delete_log;

```

Output:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The schema tree shows tables like movie_collection, movie_genre, movie_keywords, key_id, keyword_name, indexes, triggers, movie_lines, and metadata.
- Table: movie_collection:** Shows columns collection_id (AI PK), id, name, poster_path, backdrop_path, and movie_id.
- Query Editor:** The SQL tab contains the code provided in the previous section, including the creation of the prod_delete_log table, the creation of the trigger, and the execution of a delete query.
- Result Grid:** The Results tab displays the data from the prod_delete_log table, showing one row with log_id 1, production_id 74586, production_name 'Sanjana Productions', and deleted_at 2024-03-22 14:55:56.
- Output Tab:** The Output tab shows the execution history of the commands, including the creation of the trigger, the insert into the log table, and the execution of the delete command.

- Stored procedures:

1. List the top 10 popular movies available in movie_metadata table, considering their release dates are on or before today, and each movie has a positive popularity rating and vote average is greater than 5?

```

DELIMITER //

CREATE PROCEDURE get_popular_movies()
BEGIN
    SELECT title, original_language, release_date, popularity, vote_average
    FROM movie_metadata
    WHERE release_date <= CURRENT_DATE()
    AND popularity > 0
    AND vote_average > 5
    ORDER BY popularity DESC
    LIMIT 10;
END//;
movie_metadata
DELIMITER ;
CALL get_popular_movies();

```

Output:

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:**
 - MANAGEMENT:** Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore.
 - INSTANCE:** Startup / Shutdown, Server Logs, Options File.
 - Administration:** Schemas.
 - Information:** Connection Details, Object Info, Session.
- SQL Editor:** SQL File 3*, SQL File 4*, project225


```

1 DELIMITER //
2 CREATE PROCEDURE get_popular_movies()
3 BEGIN
4     SELECT title, original_language, release_date, popularity, vote_average
5     FROM movie_metadata
6     WHERE release_date <= CURRENT_DATE()
7     AND popularity > 0
8     AND vote_average > 5
9     ORDER BY popularity DESC
10    LIMIT 10;
11 END//;
12 movie_metadata
13 DELIMITER ;
14 CALL get_popular_movies();
      
```
- Result Grid:** Shows the results of the executed query, listing 10 movies with their details:

	title	original_language	release_date	popularity	vote_average
1	Minions	en	6/17/2015	547.488298	6.4
2	Wonder Woman	en	5/20/2017	294.337037	7.2
3	Beauty and the Beast	en	3/16/2017	287.253654	6.8
4	Baby Driver	en	6/28/2017	228.869744	7.2
5	Big Hero 6	en	10/03/2014	213.849704	7.8
6	Dadpool	en	2/8/2016	187.860492	7.4
7	Guardians of the Galaxy Vol. 2	en	4/19/2017	185.330992	7.6
8	Avatar	en	12/10/2009	185.070892	7.2
9	John Wick	en	10/22/2014	183.870374	7.0
10	Gone Girl	en	10/1/2014	154.801009	7.9
- Action Output:**
 - 3 15:57:43 CREATE PROCEDURE get_popular_movies() BEGIN SELECT title,original_language,release_d... 0 row(s) affected
 - 4 15:57:43 CALL get_popular_movies() 10 row(s) returned

2. Write a stored procedure named GetMovieDetails that takes a movie ID as input and returns details about the movie, including its title, release date, budget, revenue, and average rating. If the movie ID does not exist, the procedure should return a message indicating that the movie does not exist.

```

DELIMITER //
CREATE PROCEDURE GetMovieDetails (
    IN movie_id INT)
BEGIN
    DECLARE movie_count INT;
    SELECT COUNT(*) INTO movie_count FROM movie_metadata WHERE id = movie_id;
    IF movie_count = 0 THEN
        SELECT 'Movie with ID ' || movie_id || ' does not exist.' AS Message;
    ELSE
        SELECT title, release_date, budget, revenue, vote_average AS average_rating
        FROM movie_metadata
        WHERE id = movie_id;
    END IF;
END;
//
DELIMITER ;
CALL GetMovieDetails(87);

```

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Shows the database schema with tables like movie_cast, movie_collection, movie_crew, movie_genre, movie_keywords, movie_landscape, movie_metadata, movie_production_countries, movie_ratings, movie_spoken_language, and production_companies.
- SQL Editor:** Contains the SQL code for the stored procedure GetMovieDetails, which checks if a movie exists by ID and returns its details or a message if it doesn't.
- Result Grid:** Displays the result of the query `SELECT * FROM movie_metadata WHERE id = 87;`, showing one row for "Indiana Jones and the Temple of Doom" with the following values:

title	release_date	budget	revenue	average_rating
Indiana Jones and the Temple of Doom	5/23/1984	2800000	333000000	7.1
- Output Window:** Shows the execution log with two entries:

#	Action	Message	Duration / Fetch
14	17:45:02 select * from movie_metadata	45433 row(s) returned	0.015 sec / 0.405 sec
15	17:45:41 CALL GetMovieDetails(87)	1 row(s) returned	0.000 sec / 0.000 sec

- Logging Mechanism:

1. When a user rates a movie. The user's actions are automatically logged and a log table is maintained with the information given below.

A log table is created which takes log_id (Primary key with TIMESTAMP as unique identifier), user_id, actions performed, affect on table, and details as input. A trigger is created to automate the process when a user inputs rating.

```
CREATE TABLE db_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id INT,
    action VARCHAR(255),
    table_affected VARCHAR(255),
    details TEXT
);
DELIMITER //
CREATE TRIGGER log_after_insert
AFTER INSERT ON movie_ratings
FOR EACH ROW
BEGIN
    INSERT INTO db_log (user_id, action, table_affected, details)
    VALUES (NEW.user_id, 'INSERT', 'movie_ratings', CONCAT('New row inserted:', 
    NEW.column1, ', ', NEW.column2));
END //
DELIMITER ;
```