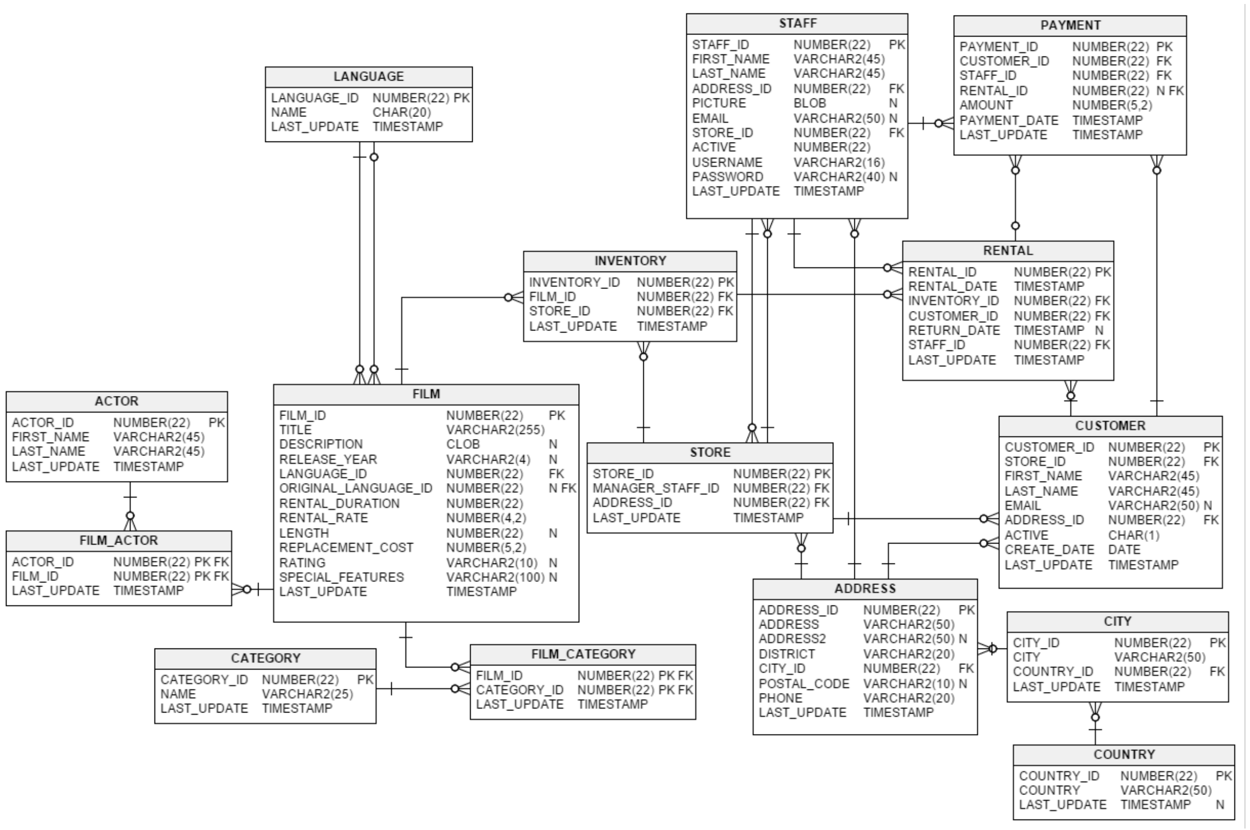
Introduction

The Sakila database is a nicely normalised schema modelling a DVD rental store, featuring things like films, actors, film-actor relationships, and a central inventory table that connects films, stores, and rentals.



Installation

Download from https://downloads.mysql.com/docs/sakila-db.zip

A downloadable archive is available in compressed **tar** file or Zip format. The archive contains three files: `sakila-schema.sql`, `sakila-data.sql`, and `sakila.mwb`.

The `sakila-schema.sql` file contains all the `CREATE` statements required to create the structure of the Sakila database including tables, views, stored procedures, and triggers.

The `sakila-data.sql` file contains the `INSERT` statements required to populate the structure created by the `sakila-schema.sql` file, along with definitions for triggers that must be created after the initial data load.

The `sakila.mwb` file is a MySQL Workbench data model that you can open within MySQL Workbench to examine the database structure

To install the Sakila sample database, follow these steps:

1. Extract the installation archive to a temporary location such as `C:\temp\` or `/tmp/`. When you unpack the archive, it creates a directory named `sakila-db` that contains the `sakila-schema.sql` and `sakila-data.sql` files.
2. Connect to the MySQL server using the **mysql** command-line client with the following command:

   ```
   $> mysql -u root -p
   ```

   Enter your password when prompted.

3. Execute the `sakila-schema.sql` script to create the database structure, and execute the `sakila-data.sql` script to populate the database structure, by using the following commands:

   ```
   mysql> SOURCE C:/temp/sakila-db/sakila-schema.sql;

   mysql> SOURCE C:/temp/sakila-db/sakila-data.sql;
   ```

   Replace the paths to the `sakila-schema.sql` and `sakila-data.sql` files with the actual paths on your system.

4. Confirm that the sample database is installed correctly. Execute the following statements. You should see output similar to that shown here.

```
mysql> USE sakila;
Database changed

mysql> SHOW FULL TABLES;
+----------------------------+------------+
| Tables_in_sakila           | Table_type |
+----------------------------+------------+
| actor                      | BASE TABLE |
| actor_info                 | VIEW       |
| address                    | BASE TABLE |
| category                   | BASE TABLE |
| city                       | BASE TABLE |
| country                    | BASE TABLE |
| customer                   | BASE TABLE |
| customer_list              | VIEW       |
| film                       | BASE TABLE |
| film_actor                 | BASE TABLE |
| film_category              | BASE TABLE |
| film_list                  | VIEW       |
| film_text                  | BASE TABLE |
| inventory                  | BASE TABLE |
| language                   | BASE TABLE |
| nicer_but_slower_film_list | VIEW       |
| payment                    | BASE TABLE |
| rental                     | BASE TABLE |
| sales_by_film_category     | VIEW       |
| sales_by_store             | VIEW       |
| staff                      | BASE TABLE |
| staff_list                 | VIEW       |
| store                      | BASE TABLE |
+----------------------------+------------+
23 rows in set (0.01 sec)
```

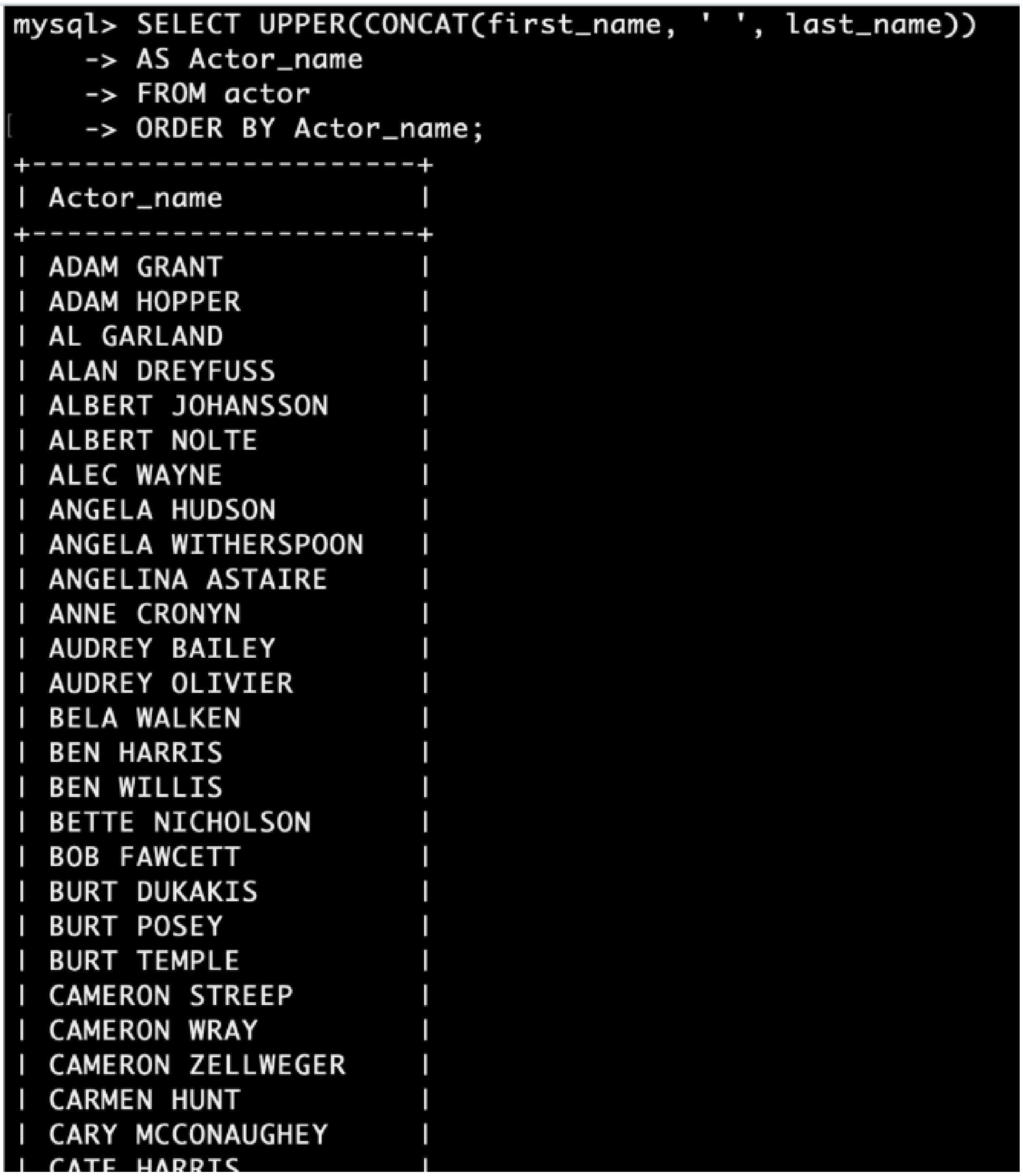
```
mysql> SELECT COUNT(*) FROM film;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM film_text;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)
```

Tables

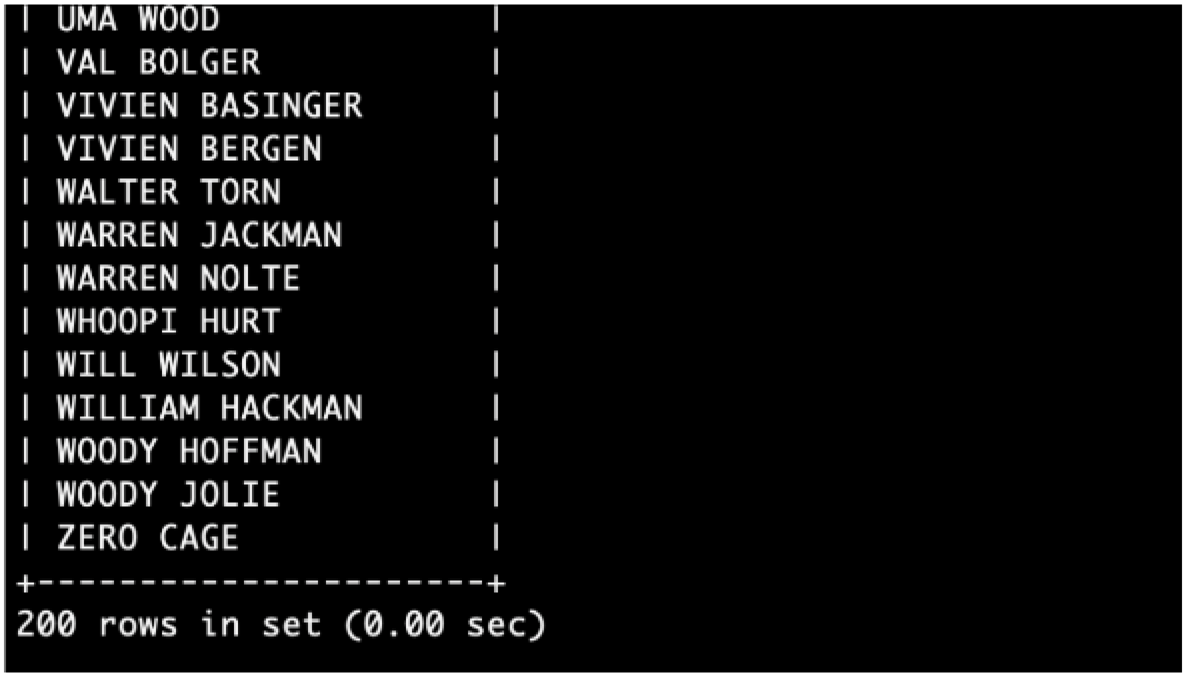
`https://dev.mysql.com/doc/sakila/en/sakila-structure-tables.html`

Exercises

1. Display the first and last name of each actor in a single column in upper case letters in alphabetic order. Name the column Actor Name.

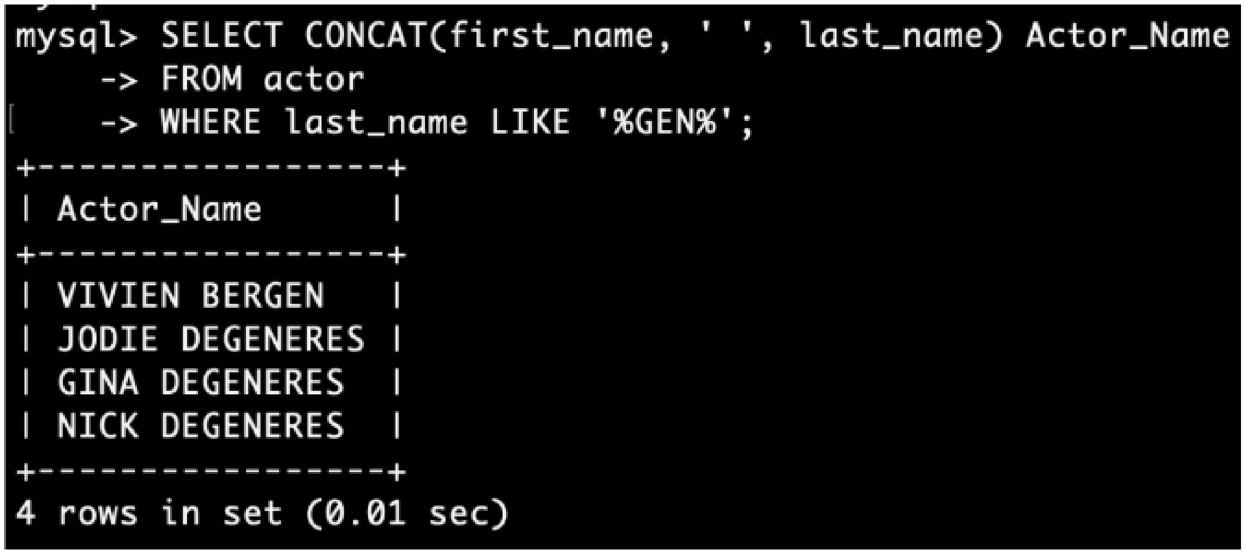


```
mysql> SELECT UPPER(CONCAT(first_name, ' ', last_name))
    -> AS Actor_name
    -> FROM actor
    -> ORDER BY Actor_name;
+----------------------+
| Actor_name           |
+----------------------+
| ADAM GRANT           |
| ADAM HOPPER          |
| AL GARLAND           |
| ALAN DREYFUSS        |
| ALBERT JOHANSSON     |
| ALBERT NOLTE         |
| ALEC WAYNE           |
| ANGELA HUDSON        |
| ANGELA WITHERSPOON   |
| ANGELINA ASTAIRE     |
| ANNE CRONYN          |
| AUDREY BAILEY        |
| AUDREY OLIVIER       |
| BELA WALKEN          |
| BEN HARRIS           |
| BEN WILLIS           |
| BETTE NICHOLSON      |
| BOB FAWCETT          |
| BURT DUKAKIS         |
| BURT POSEY           |
| BURT TEMPLE          |
| CAMERON STREEP       |
| CAMERON WRAY         |
| CAMERON ZELLWEGER    |
| CARMEN HUNT          |
| CARY MCCONAUGHEY     |
| CATE HARRIS          |
```



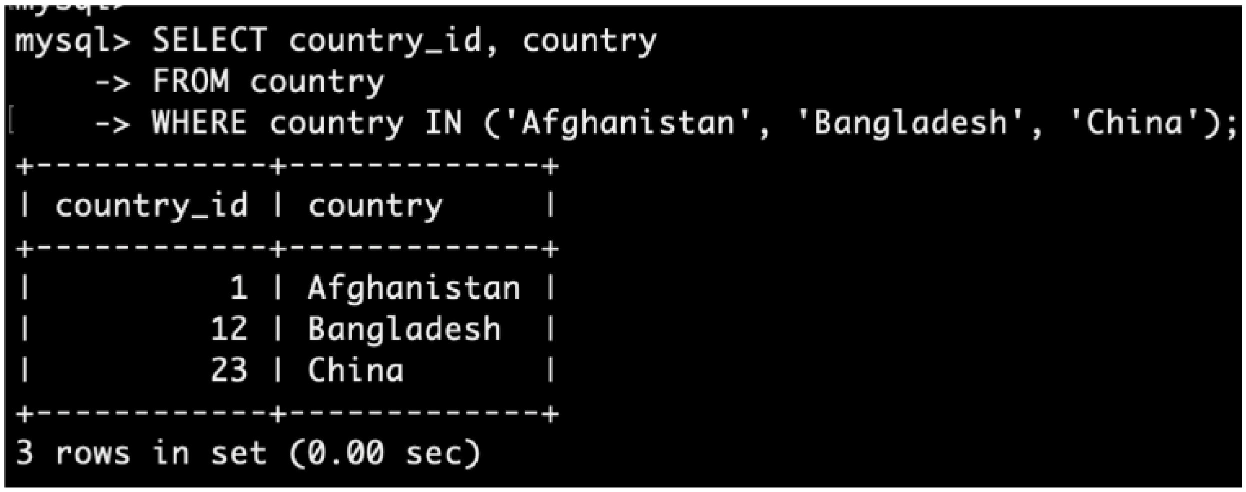
```
| UMA WOOD              |
| VAL BOLGER            |
| VIVIEN BASINGER       |
| VIVIEN BERGEN         |
| WALTER TORN           |
| WARREN JACKMAN        |
| WARREN NOLTE          |
| WHOOPI HURT           |
| WILL WILSON           |
| WILLIAM HACKMAN       |
| WOODY HOFFMAN         |
| WOODY JOLIE           |
| ZERO CAGE             |
+-----------------------+
200 rows in set (0.00 sec)
```

2. Find all actors whose last name contain the letters GEN:



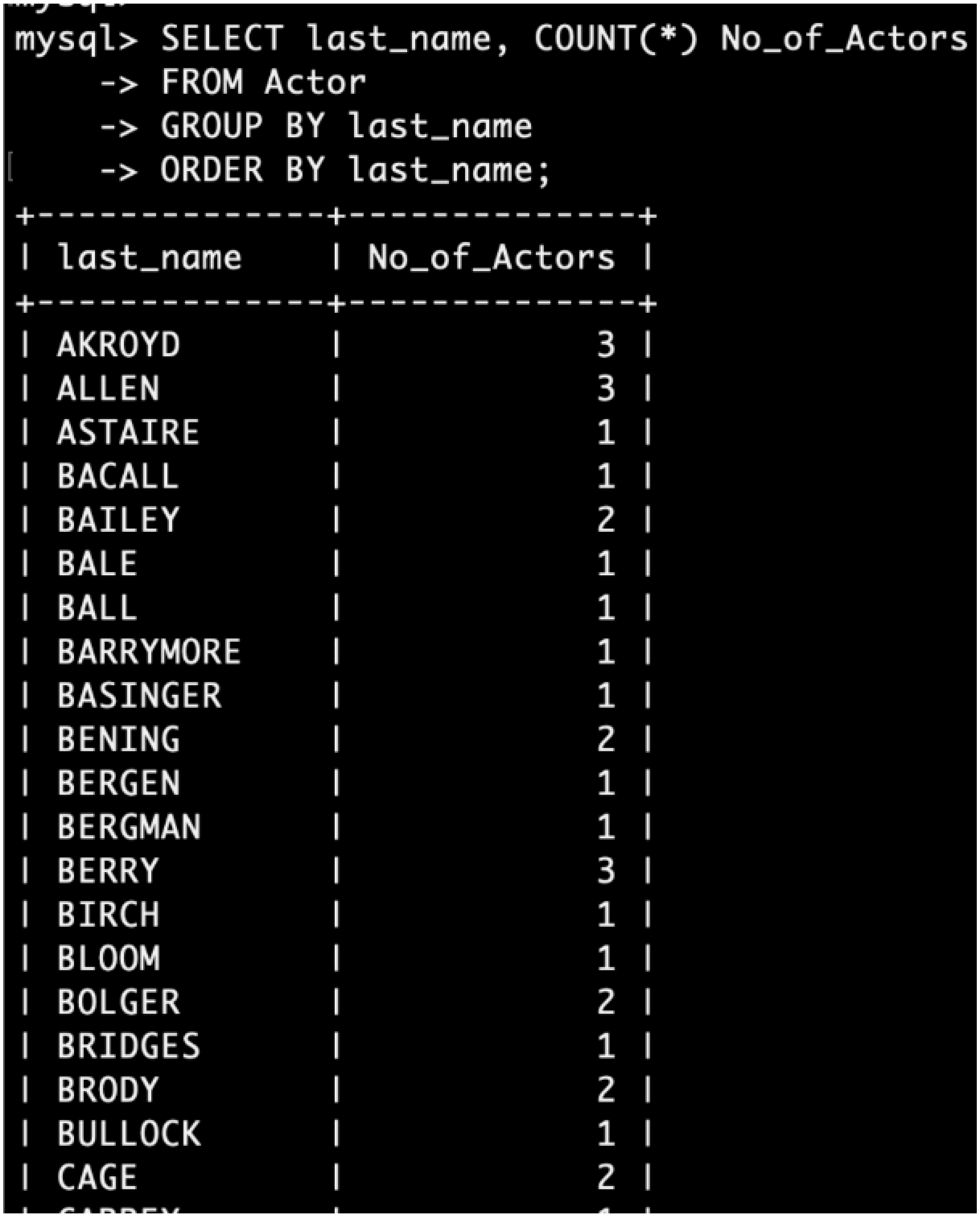
```
mysql> SELECT CONCAT(first_name, ' ', last_name) Actor_Name
    -> FROM actor
    -> WHERE last_name LIKE '%GEN%';
+-----------------+
| Actor_Name      |
+-----------------+
| VIVIEN BERGEN   |
| JODIE DEGENERES |
| GINA DEGENERES  |
| NICK DEGENERES  |
+-----------------+
4 rows in set (0.01 sec)
```

3.Using IN, display the country_id and country columns of the following countries: Afghanistan, Bangladesh, and China:

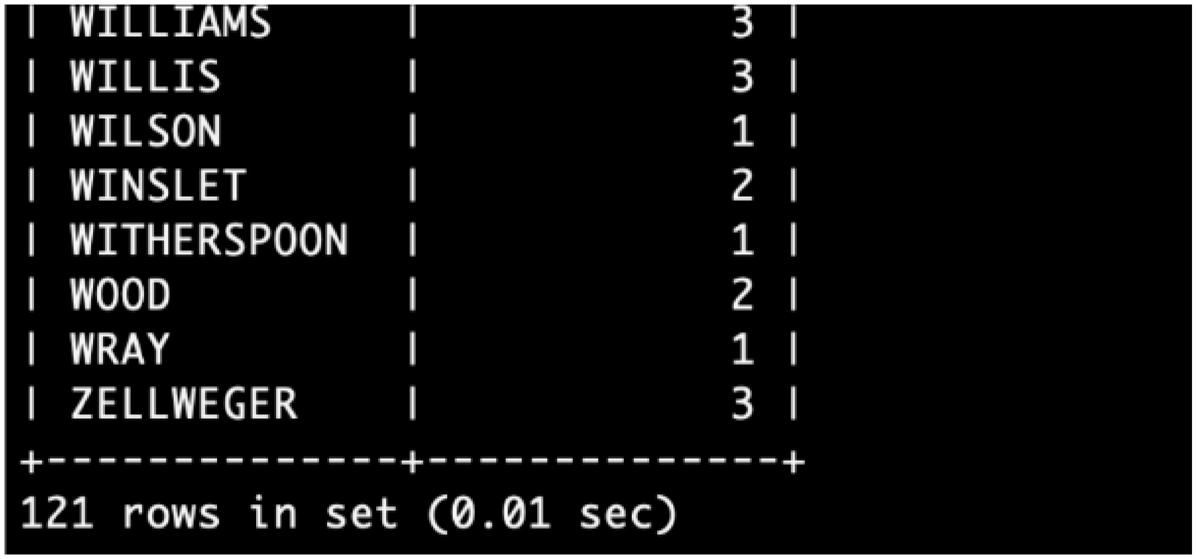


```
mysql> SELECT country_id, country
    -> FROM country
    -> WHERE country IN ('Afghanistan', 'Bangladesh', 'China');
+------------+-------------+
| country_id | country     |
+------------+-------------+
|          1 | Afghanistan |
|         12 | Bangladesh  |
|         23 | China       |
+------------+-------------+
3 rows in set (0.00 sec)
```

4. List the last names of actors, as well as how many actors have that last name

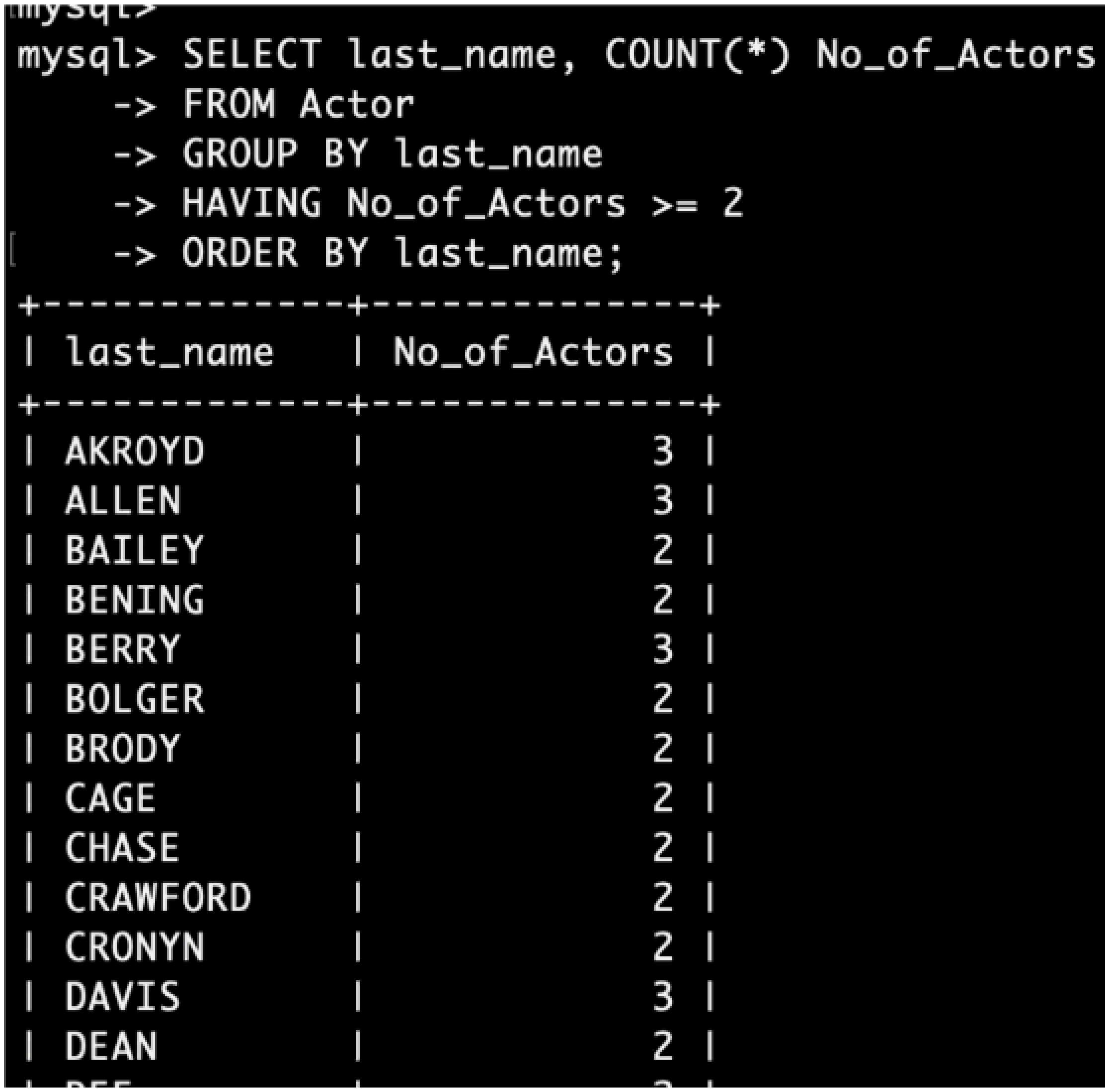


```
mysql> SELECT last_name, COUNT(*) No_of_Actors
    -> FROM Actor
    -> GROUP BY last_name
    -> ORDER BY last_name;
+-------------+--------------+
| last_name   | No_of_Actors |
+-------------+--------------+
| AKROYD      |            3 |
| ALLEN       |            3 |
| ASTAIRE     |            1 |
| BACALL      |            1 |
| BAILEY      |            2 |
| BALE        |            1 |
| BALL        |            1 |
| BARRYMORE   |            1 |
| BASINGER    |            1 |
| BENING      |            2 |
| BERGEN      |            1 |
| BERGMAN     |            1 |
| BERRY       |            3 |
| BIRCH       |            1 |
| BLOOM       |            1 |
| BOLGER      |            2 |
| BRIDGES     |            1 |
| BRODY       |            2 |
| BULLOCK     |            1 |
| CAGE        |            2 |
```

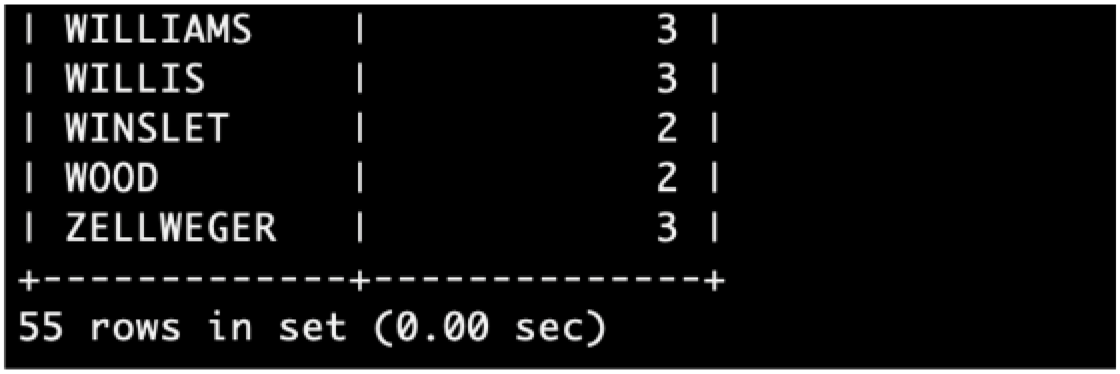


```
| WILLIAMS     |            3 |
| WILLIS       |            3 |
| WILSON       |            1 |
| WINSLET      |            2 |
| WITHERSPOON  |            1 |
| WOOD         |            2 |
| WRAY         |            1 |
| ZELLWEGER    |            3 |
+--------------+--------------+
121 rows in set (0.01 sec)
```

5 .   List last names of actors and the number of actors who have that last name, but only for names that are shared by at least two actors

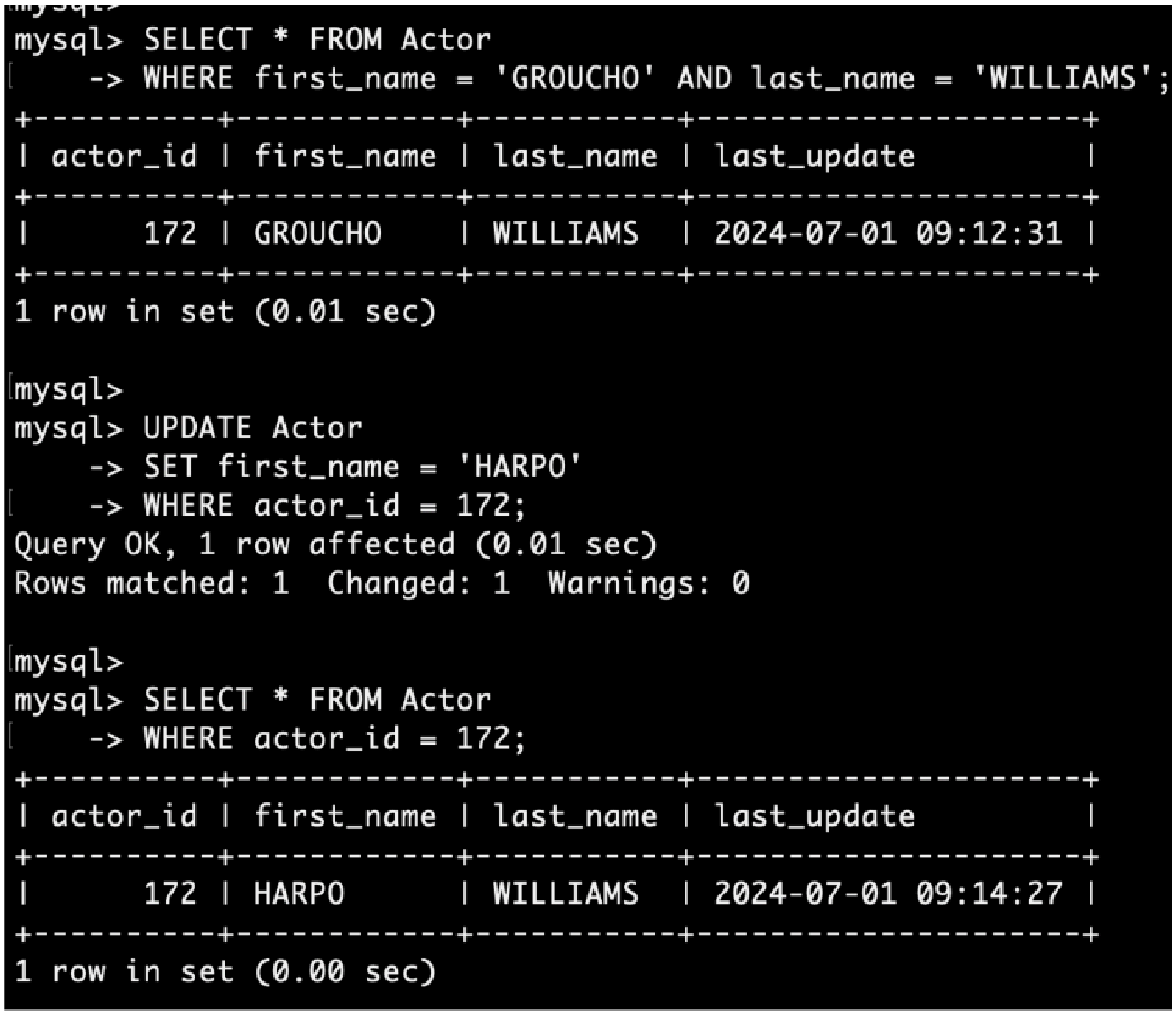


```
mysql>
mysql> SELECT last_name, COUNT(*) No_of_Actors
    -> FROM Actor
    -> GROUP BY last_name
    -> HAVING No_of_Actors >= 2
    -> ORDER BY last_name;
+------------+-------------+
| last_name  | No_of_Actors |
+------------+-------------+
| AKROYD     |           3 |
| ALLEN      |           3 |
| BAILEY     |           2 |
| BENING     |           2 |
| BERRY      |           3 |
| BOLGER     |           2 |
| BRODY      |           2 |
| CAGE       |           2 |
| CHASE      |           2 |
| CRAWFORD   |           2 |
| CRONYN     |           2 |
| DAVIS      |           3 |
| DEAN       |           2 |
```



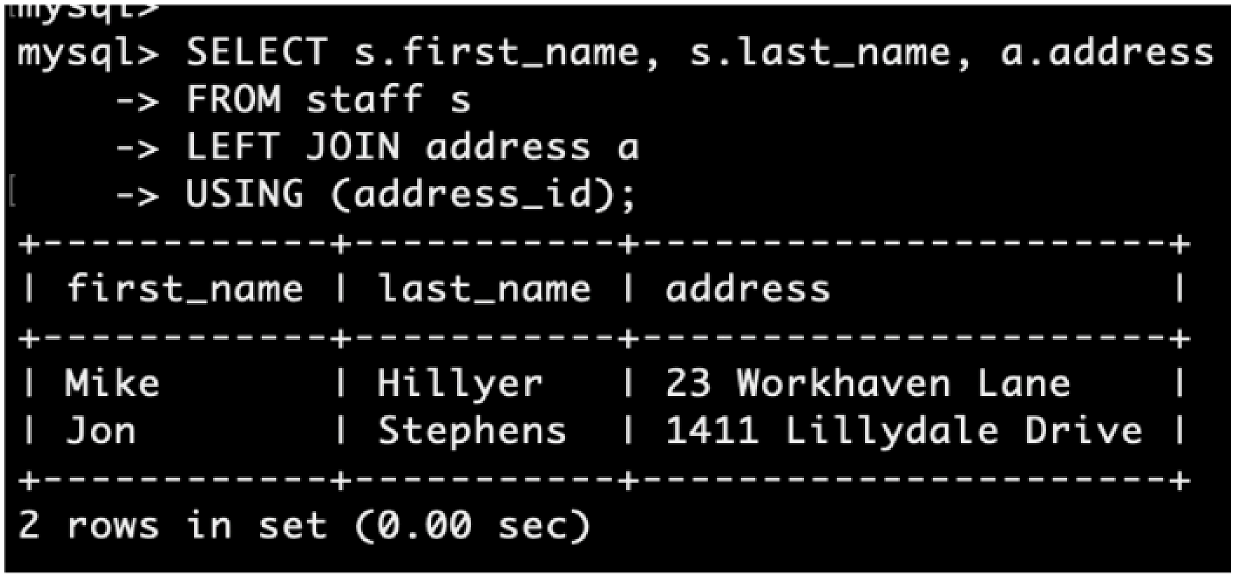
```
| WILLIAMS   |           3 |
| WILLIS     |           3 |
| WINSLET    |           2 |
| WOOD       |           2 |
| ZELLWEGER  |           3 |
+------------+-------------+
55 rows in set (0.00 sec)
```

6. The actor HARPO WILLIAMS was accidentally entered in the actor table as GROUCHO WILLIAMS. Write a query to fix the record.



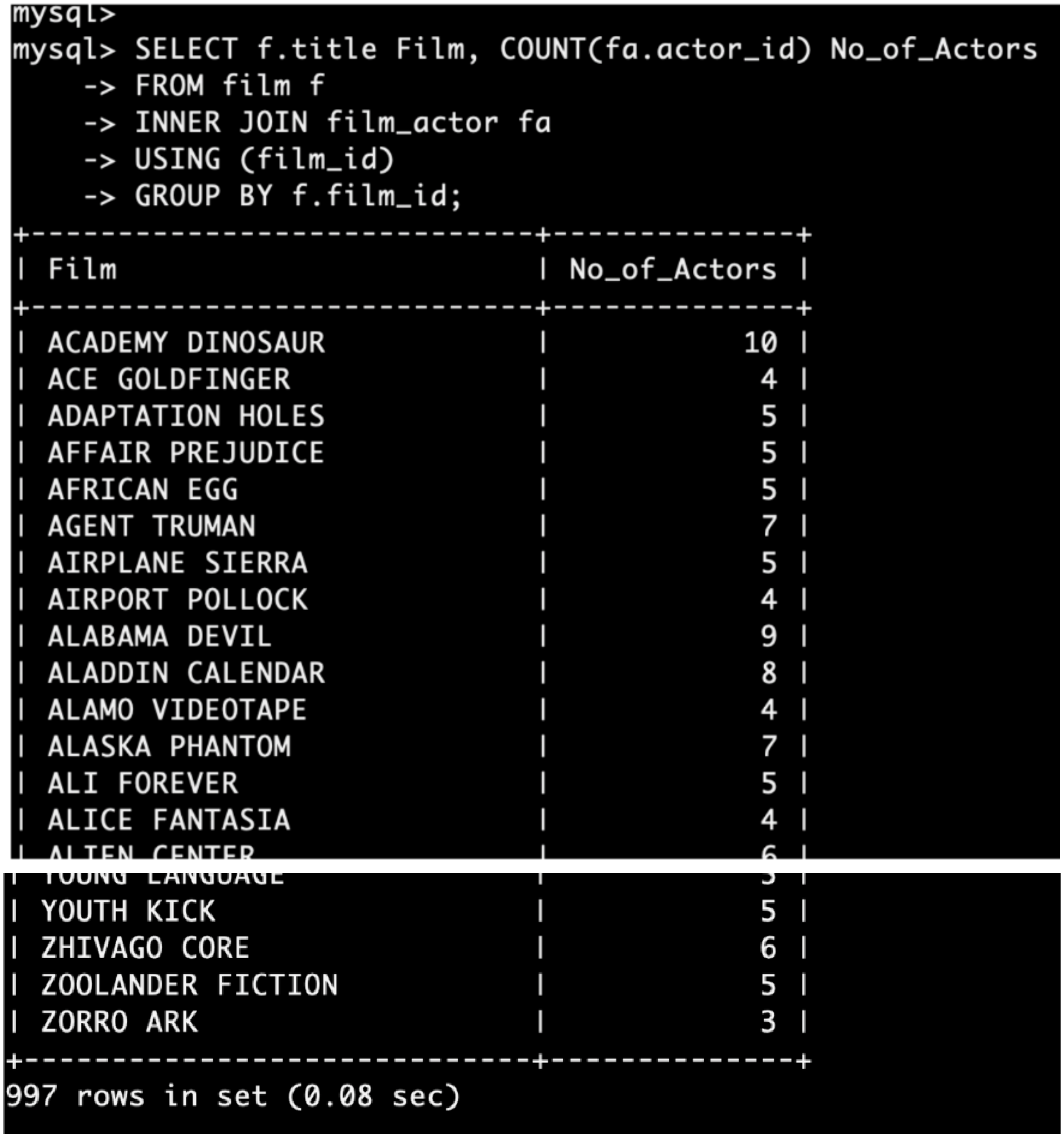
```
mysql>
mysql> SELECT * FROM Actor
    -> WHERE first_name = 'GROUCHO' AND last_name = 'WILLIAMS';
+----------+------------+-----------+---------------------+
| actor_id | first_name | last_name | last_update         |
+----------+------------+-----------+---------------------+
|      172 | GROUCHO    | WILLIAMS  | 2024-07-01 09:12:31 |
+----------+------------+-----------+---------------------+
1 row in set (0.01 sec)

mysql>
mysql> UPDATE Actor
    -> SET first_name = 'HARPO'
    -> WHERE actor_id = 172;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> SELECT * FROM Actor
    -> WHERE actor_id = 172;
+----------+------------+-----------+---------------------+
| actor_id | first_name | last_name | last_update         |
+----------+------------+-----------+---------------------+
|      172 | HARPO      | WILLIAMS  | 2024-07-01 09:14:27 |
+----------+------------+-----------+---------------------+
1 row in set (0.00 sec)
```

7. Use JOIN to display the first and last names, as well as the address, of each staff member. Use the tables staff and address:



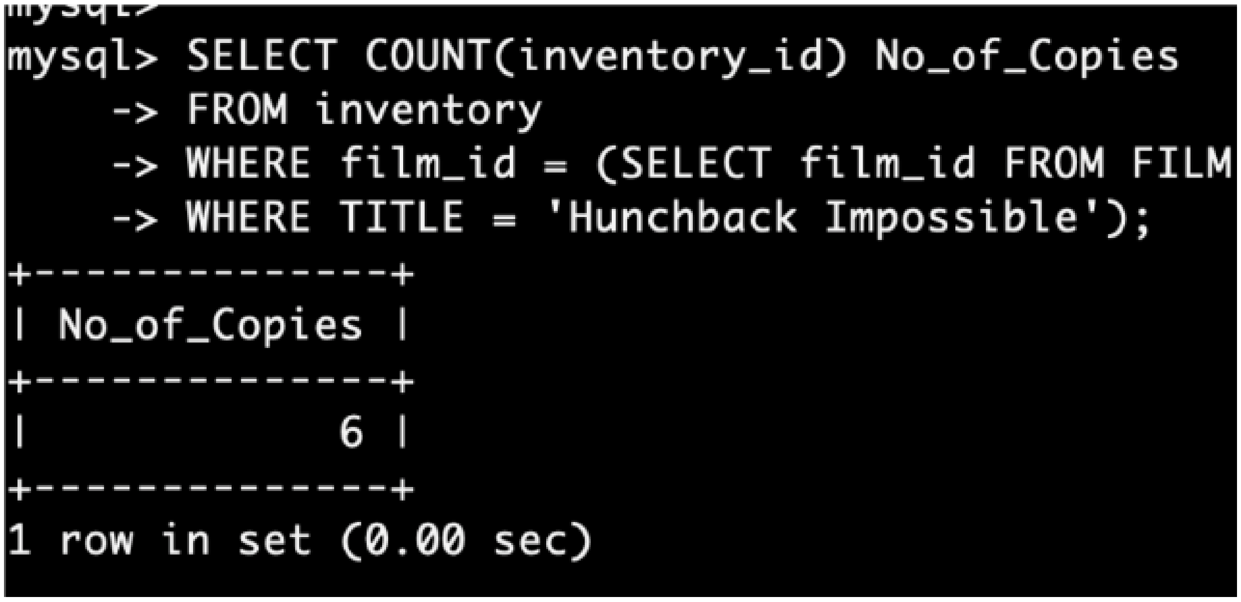
```
mysql> SELECT s.first_name, s.last_name, a.address
    -> FROM staff s
    -> LEFT JOIN address a
    -> USING (address_id);
+------------+-----------+----------------------+
| first_name | last_name | address              |
+------------+-----------+----------------------+
| Mike       | Hillyer   | 23 Workhaven Lane    |
| Jon        | Stephens  | 1411 Lillydale Drive |
+------------+-----------+----------------------+
2 rows in set (0.00 sec)
```

8. List each film and the number of actors who are listed for that film. Use tables film_actor and film. Use inner join.



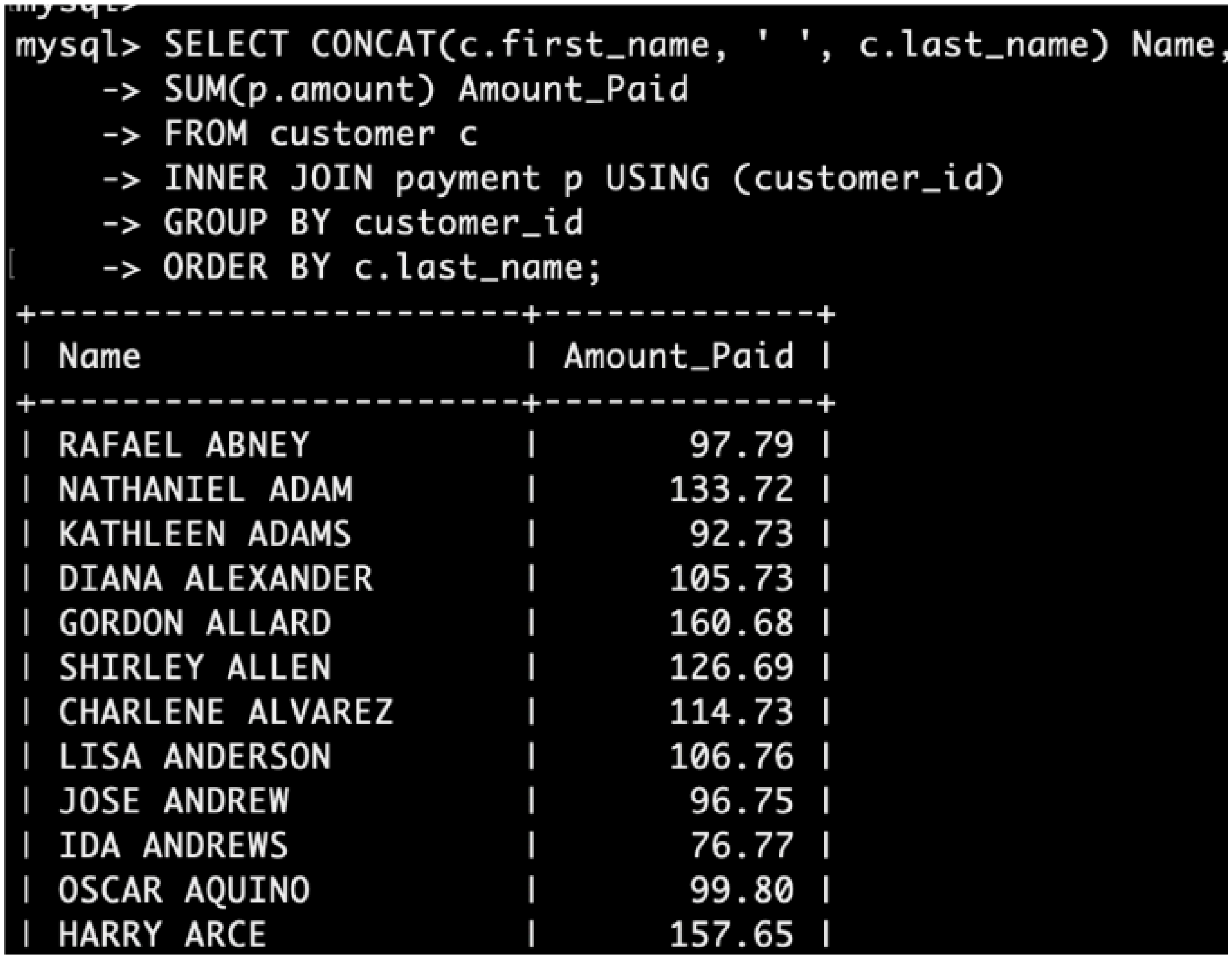
```
mysql>
mysql> SELECT f.title Film, COUNT(fa.actor_id) No_of_Actors
    -> FROM film f
    -> INNER JOIN film_actor fa
    -> USING (film_id)
    -> GROUP BY f.film_id;
+-----------------------------+--------------+
| Film                        | No_of_Actors |
+-----------------------------+--------------+
| ACADEMY DINOSAUR            |           10 |
| ACE GOLDFINGER              |            4 |
| ADAPTATION HOLES            |            5 |
| AFFAIR PREJUDICE            |            5 |
| AFRICAN EGG                 |            5 |
| AGENT TRUMAN                |            7 |
| AIRPLANE SIERRA             |            5 |
| AIRPORT POLLOCK             |            4 |
| ALABAMA DEVIL               |            9 |
| ALADDIN CALENDAR            |            8 |
| ALAMO VIDEOTAPE             |            4 |
| ALASKA PHANTOM              |            7 |
| ALI FOREVER                 |            5 |
| ALICE FANTASIA              |            4 |
| ALIEN CENTER                |            6 |
| YOUNG LANGUAGE              |            5 |
| YOUTH KICK                  |            5 |
| ZHIVAGO CORE                |            6 |
| ZOOLANDER FICTION           |            5 |
| ZORRO ARK                   |            3 |
+-----------------------------+--------------+
997 rows in set (0.08 sec)
```

9. How many copies of the film Hunchback Impossible exist in the inventory system?

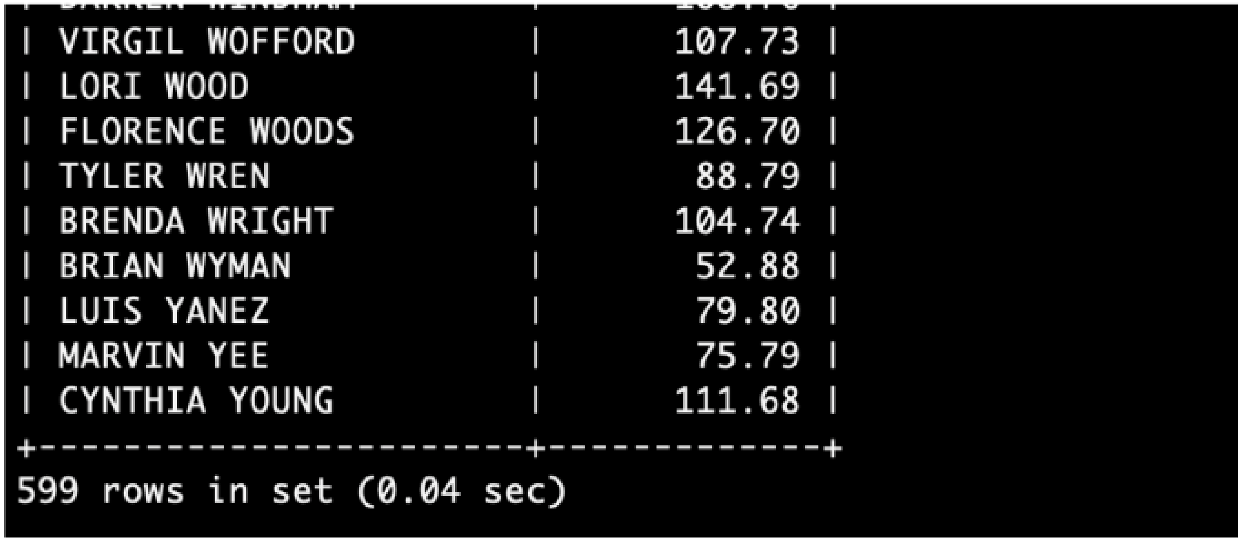


```
mysql> SELECT COUNT(inventory_id) No_of_Copies
    -> FROM inventory
    -> WHERE film_id = (SELECT film_id FROM FILM
    -> WHERE TITLE = 'Hunchback Impossible');
+--------------+
| No_of_Copies |
+--------------+
|            6 |
+--------------+
1 row in set (0.00 sec)
```

10 . Using the tables payment and customer and the JOIN command, list the total paid by each customer. List the customers alphabetically by last name

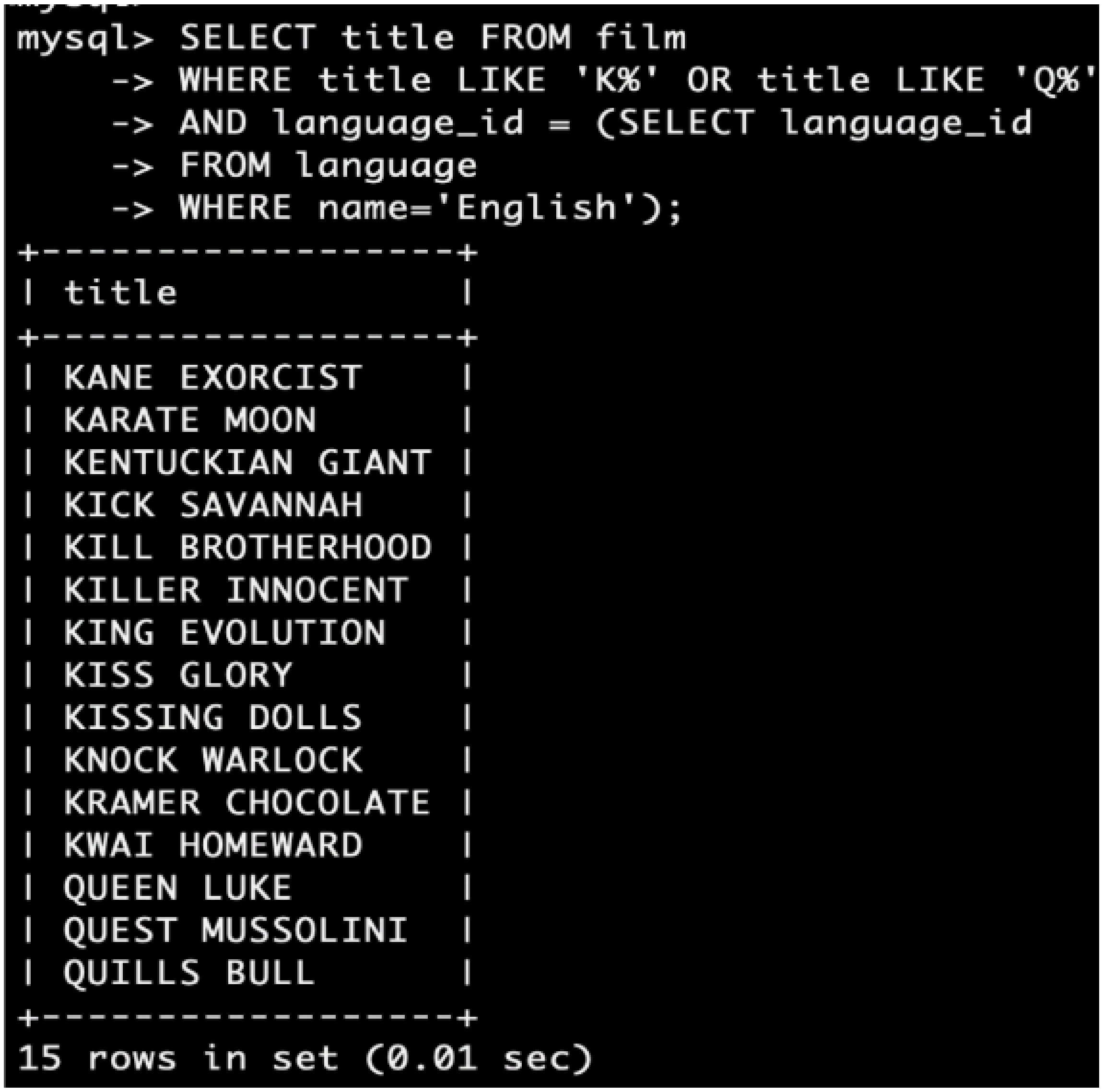


```
mysql> SELECT CONCAT(c.first_name, ' ', c.last_name) Name,
    -> SUM(p.amount) Amount_Paid
    -> FROM customer c
    -> INNER JOIN payment p USING (customer_id)
    -> GROUP BY customer_id
    -> ORDER BY c.last_name;
+-----------------------+-------------+
| Name                  | Amount_Paid |
+-----------------------+-------------+
| RAFAEL ABNEY          |       97.79 |
| NATHANIEL ADAM        |      133.72 |
| KATHLEEN ADAMS        |       92.73 |
| DIANA ALEXANDER       |      105.73 |
| GORDON ALLARD         |      160.68 |
| SHIRLEY ALLEN         |      126.69 |
| CHARLENE ALVAREZ      |      114.73 |
| LISA ANDERSON         |      106.76 |
| JOSE ANDREW           |       96.75 |
| IDA ANDREWS           |       76.77 |
| OSCAR AQUINO          |       99.80 |
| HARRY ARCE            |      157.65 |
```



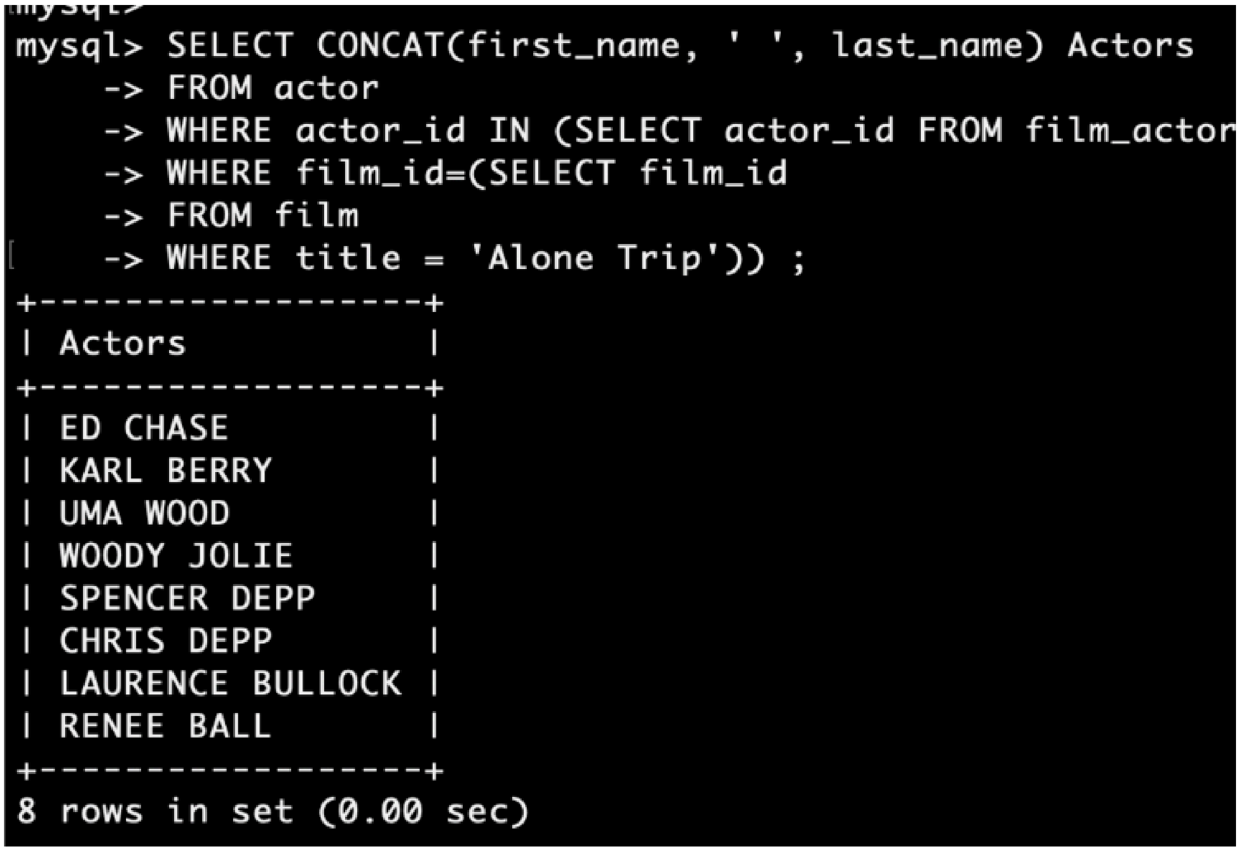
```
| VIRGIL WOFFORD        |      107.73 |
| LORI WOOD             |      141.69 |
| FLORENCE WOODS        |      126.70 |
| TYLER WREN            |       88.79 |
| BRENDA WRIGHT         |      104.74 |
| BRIAN WYMAN           |       52.88 |
| LUIS YANEZ            |       79.80 |
| MARVIN YEE            |       75.79 |
| CYNTHIA YOUNG         |      111.68 |
+-----------------------+-------------+
599 rows in set (0.04 sec)
```

11. The music of Queen and Kris Kristofferson have seen an unlikely resurgence. As an unintended consequence, films starting with the letters `K` and `Q` have also soared in popularity.Use subqueries to display the titles of movies starting with the letters `K` and `Q` whose language is English.



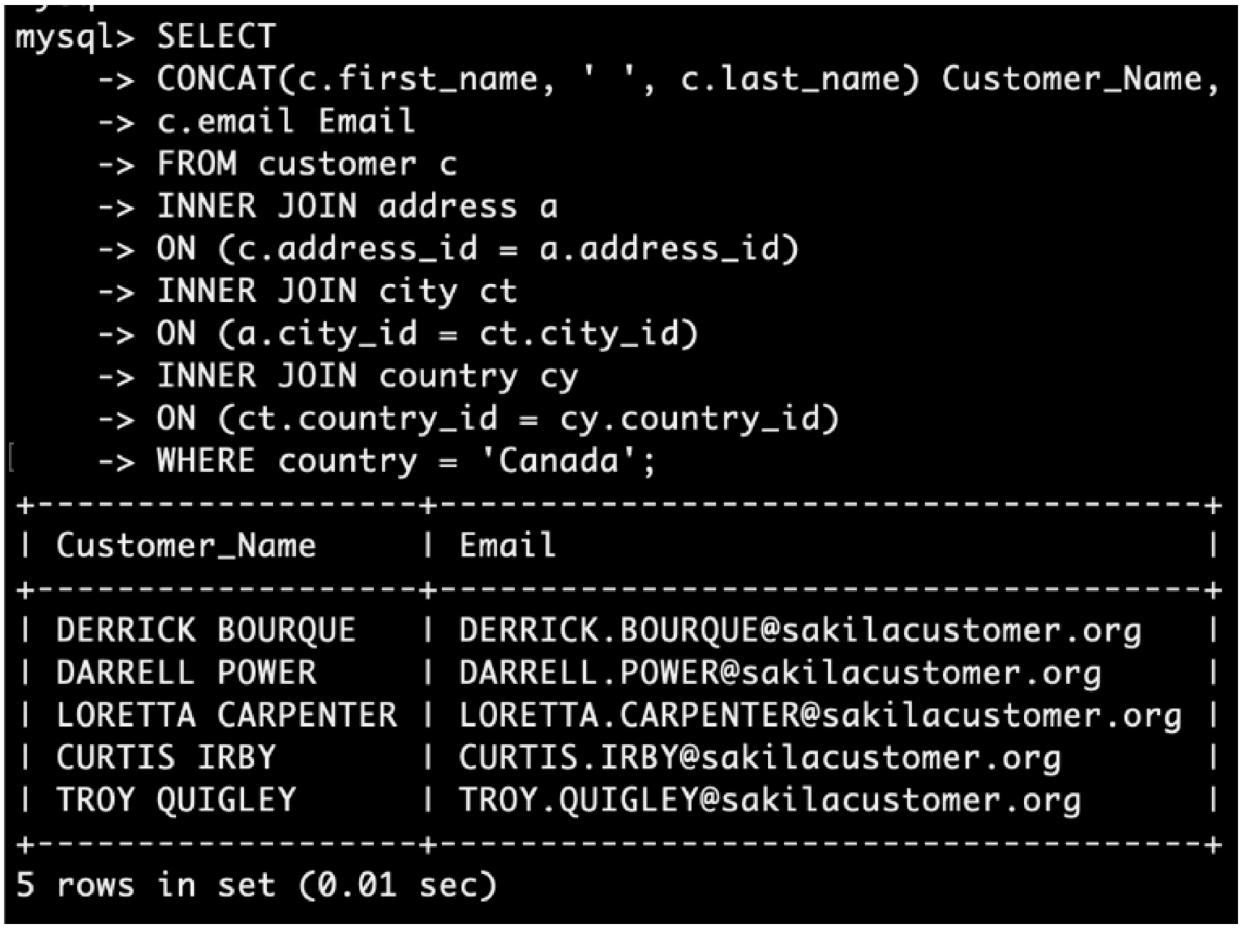
```
mysql> SELECT title FROM film
    -> WHERE title LIKE 'K%' OR title LIKE 'Q%'
    -> AND language_id = (SELECT language_id
    -> FROM language
    -> WHERE name='English');
+-------------------+
| title             |
+-------------------+
| KANE EXORCIST     |
| KARATE MOON       |
| KENTUCKIAN GIANT  |
| KICK SAVANNAH     |
| KILL BROTHERHOOD  |
| KILLER INNOCENT   |
| KING EVOLUTION    |
| KISS GLORY        |
| KISSING DOLLS     |
| KNOCK WARLOCK     |
| KRAMER CHOCOLATE  |
| KWAI HOMEWARD     |
| QUEEN LUKE        |
| QUEST MUSSOLINI   |
| QUILLS BULL       |
+-------------------+
15 rows in set (0.01 sec)
```

12 .Use subqueries to display all actors who appear in the film `Alone Trip`.



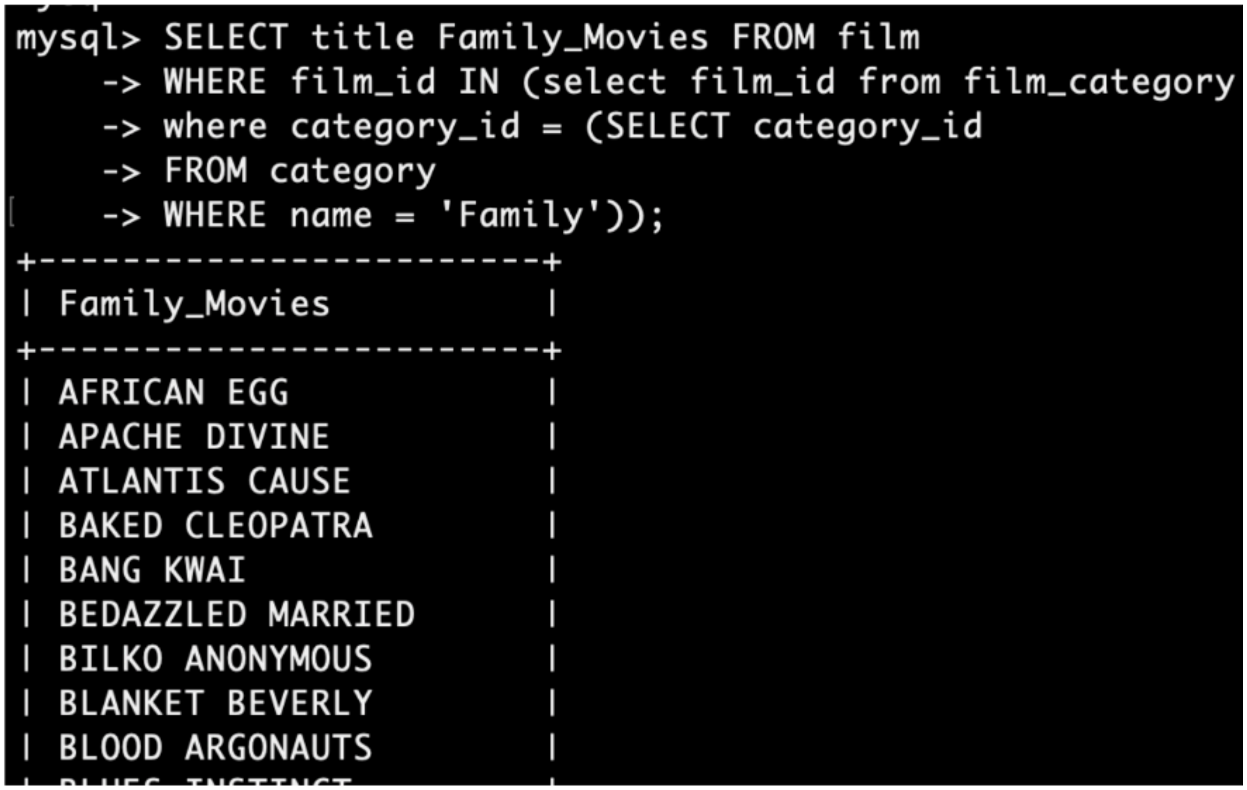
```
mysql> SELECT CONCAT(first_name, ' ', last_name) Actors
    -> FROM actor
    -> WHERE actor_id IN (SELECT actor_id FROM film_actor
    -> WHERE film_id=(SELECT film_id
    -> FROM film
    -> WHERE title = 'Alone Trip')) ;
+------------------+
| Actors           |
+------------------+
| ED CHASE         |
| KARL BERRY       |
| UMA WOOD         |
| WOODY JOLIE      |
| SPENCER DEPP     |
| CHRIS DEPP       |
| LAURENCE BULLOCK |
| RENEE BALL       |
+------------------+
8 rows in set (0.00 sec)
```

13. You want to run an email marketing campaign in Canada, for which you will need the names and email addresses of all Canadian customers. Use joins to retrieve this information.

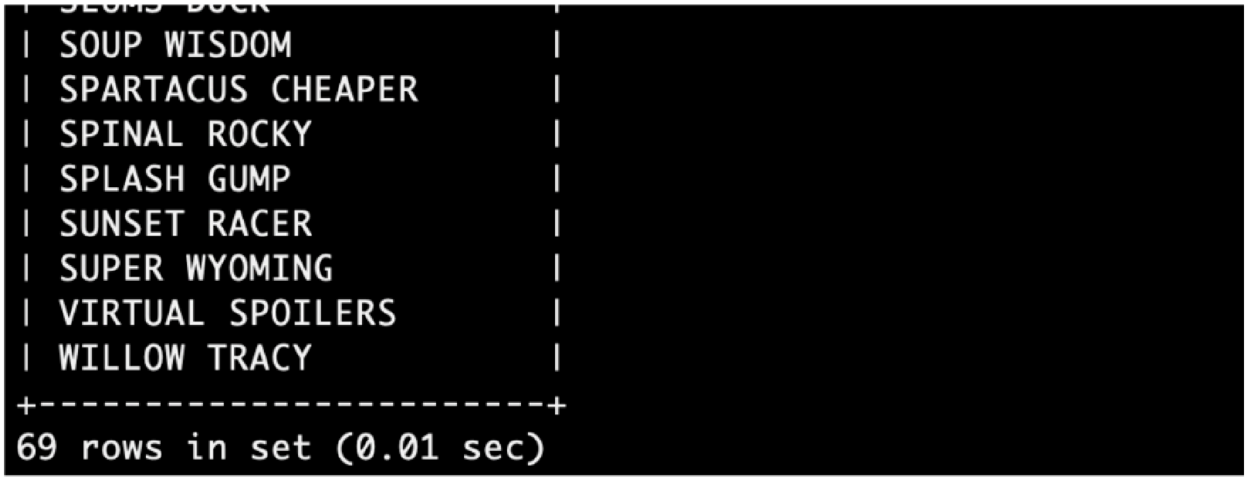


```
mysql> SELECT
    -> CONCAT(c.first_name, ' ', c.last_name) Customer_Name,
    -> c.email Email
    -> FROM customer c
    -> INNER JOIN address a
    -> ON (c.address_id = a.address_id)
    -> INNER JOIN city ct
    -> ON (a.city_id = ct.city_id)
    -> INNER JOIN country cy
    -> ON (ct.country_id = cy.country_id)
    -> WHERE country = 'Canada';
+-------------------+---------------------------------------+
| Customer_Name     | Email                                 |
+-------------------+---------------------------------------+
| DERRICK BOURQUE   | DERRICK.BOURQUE@sakilacustomer.org    |
| DARRELL POWER     | DARRELL.POWER@sakilacustomer.org      |
| LORETTA CARPENTER | LORETTA.CARPENTER@sakilacustomer.org  |
| CURTIS IRBY       | CURTIS.IRBY@sakilacustomer.org        |
| TROY QUIGLEY      | TROY.QUIGLEY@sakilacustomer.org       |
+-------------------+---------------------------------------+
5 rows in set (0.01 sec)
```

14. Sales have been lagging among young families, and you wish to target all family movies for a promotion. Identify all movies categorized as famiy films

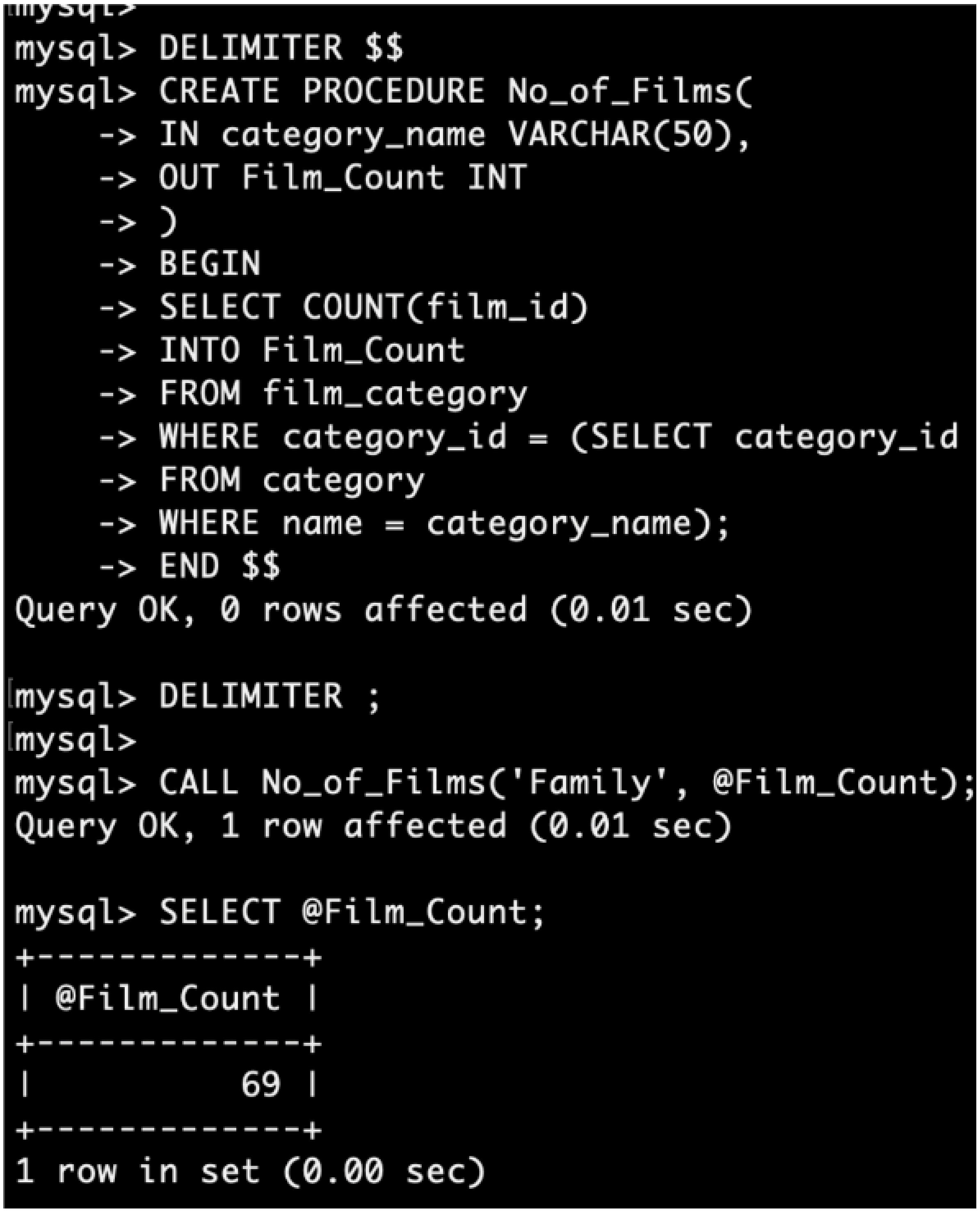


```
mysql> SELECT title Family_Movies FROM film
    -> WHERE film_id IN (select film_id from film_category
    -> where category_id = (SELECT category_id
    -> FROM category
    -> WHERE name = 'Family'));
+------------------------+
| Family_Movies          |
+------------------------+
| AFRICAN EGG            |
| APACHE DIVINE          |
| ATLANTIS CAUSE         |
| BAKED CLEOPATRA        |
| BANG KWAI              |
| BEDAZZLED MARRIED      |
| BILKO ANONYMOUS        |
| BLANKET BEVERLY        |
| BLOOD ARGONAUTS        |
| BLUES INSTINCT         |
```



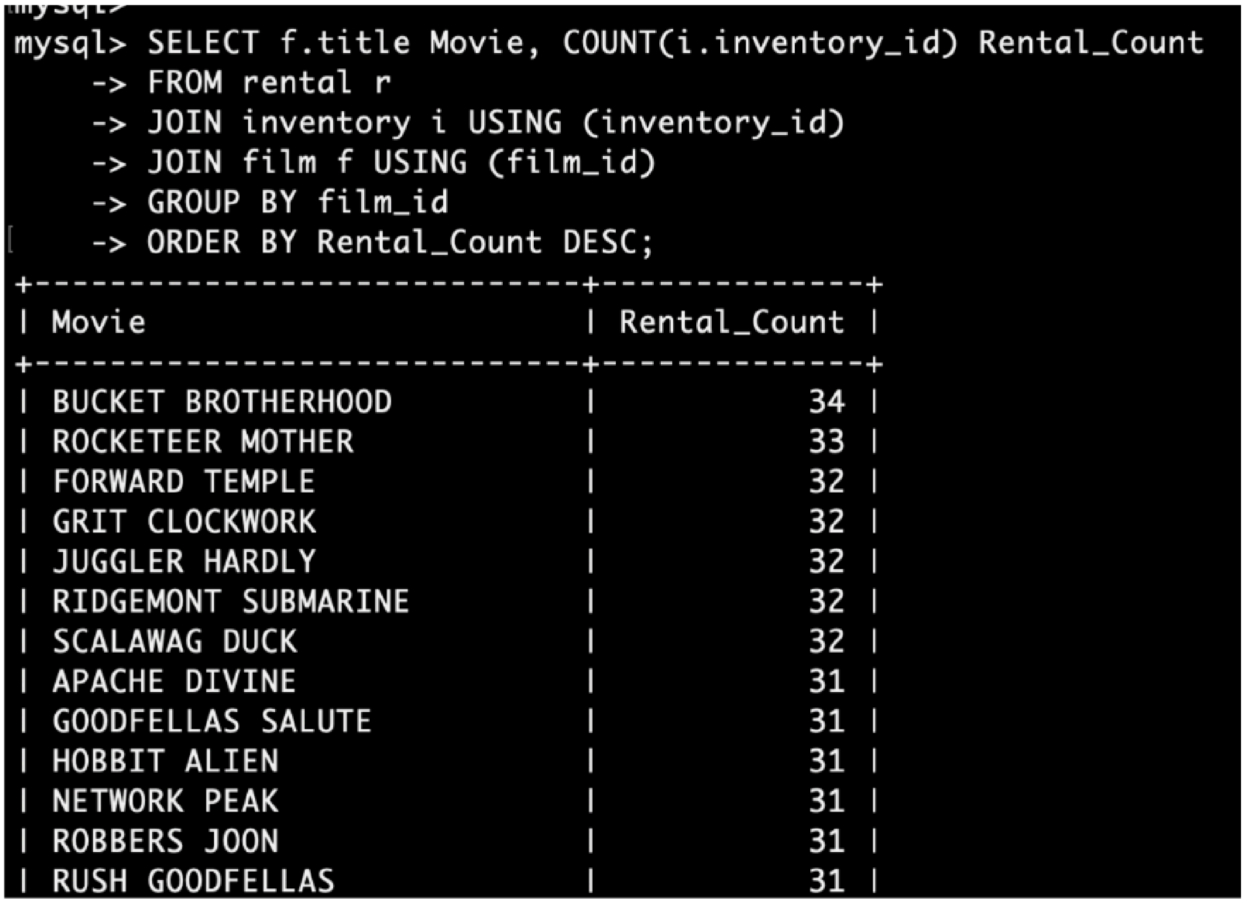
```
| SLUMS DUCK             |
| SOUP WISDOM            |
| SPARTACUS CHEAPER      |
| SPINAL ROCKY           |
| SPLASH GUMP            |
| SUNSET RACER           |
| SUPER WYOMING          |
| VIRTUAL SPOILERS       |
| WILLOW TRACY           |
+------------------------+
69 rows in set (0.01 sec)
```

15. Create a Stored procedure to get the count of films in the input category (IN category_name, OUT count)

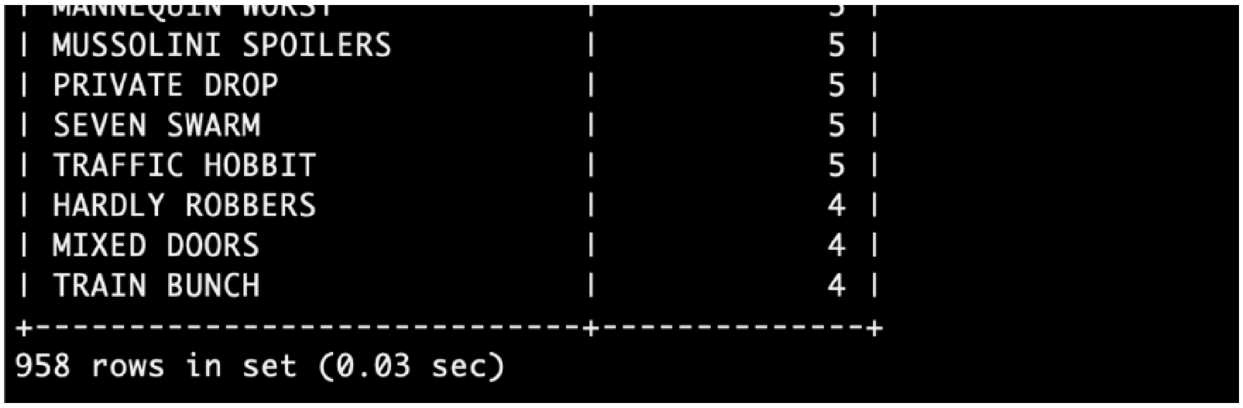


```
mysql>
mysql> DELIMITER $$
mysql> CREATE PROCEDURE No_of_Films(
    -> IN category_name VARCHAR(50),
    -> OUT Film_Count INT
    -> )
    -> BEGIN
    -> SELECT COUNT(film_id)
    -> INTO Film_Count
    -> FROM film_category
    -> WHERE category_id = (SELECT category_id
    -> FROM category
    -> WHERE name = category_name);
    -> END $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> CALL No_of_Films('Family', @Film_Count);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT @Film_Count;
+-------------+
| @Film_Count |
+-------------+
|          69 |
+-------------+
1 row in set (0.00 sec)
```

16. Display the most frequently rented movies in descending order.

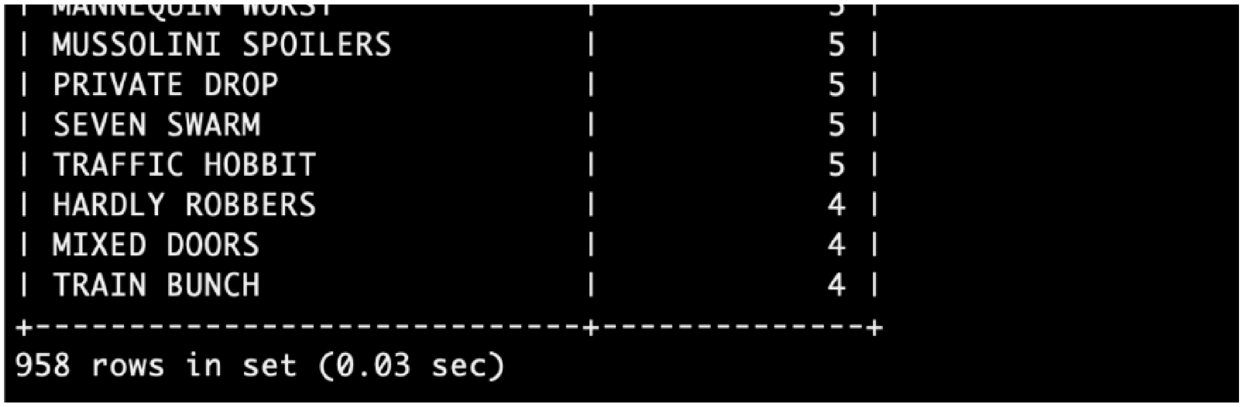


```
mysql>
mysql> SELECT f.title Movie, COUNT(i.inventory_id) Rental_Count
    -> FROM rental r
    -> JOIN inventory i USING (inventory_id)
    -> JOIN film f USING (film_id)
    -> GROUP BY film_id
    -> ORDER BY Rental_Count DESC;
+-----------------------------+--------------+
| Movie                       | Rental_Count |
+-----------------------------+--------------+
| BUCKET BROTHERHOOD          |           34 |
| ROCKETEER MOTHER            |           33 |
| FORWARD TEMPLE              |           32 |
| GRIT CLOCKWORK              |           32 |
| JUGGLER HARDLY              |           32 |
| RIDGEMONT SUBMARINE         |           32 |
| SCALAWAG DUCK               |           32 |
| APACHE DIVINE               |           31 |
| GOODFELLAS SALUTE           |           31 |
| HOBBIT ALIEN                |           31 |
| NETWORK PEAK                |           31 |
| ROBBERS JOON                |           31 |
| RUSH GOODFELLAS             |           31 |
```



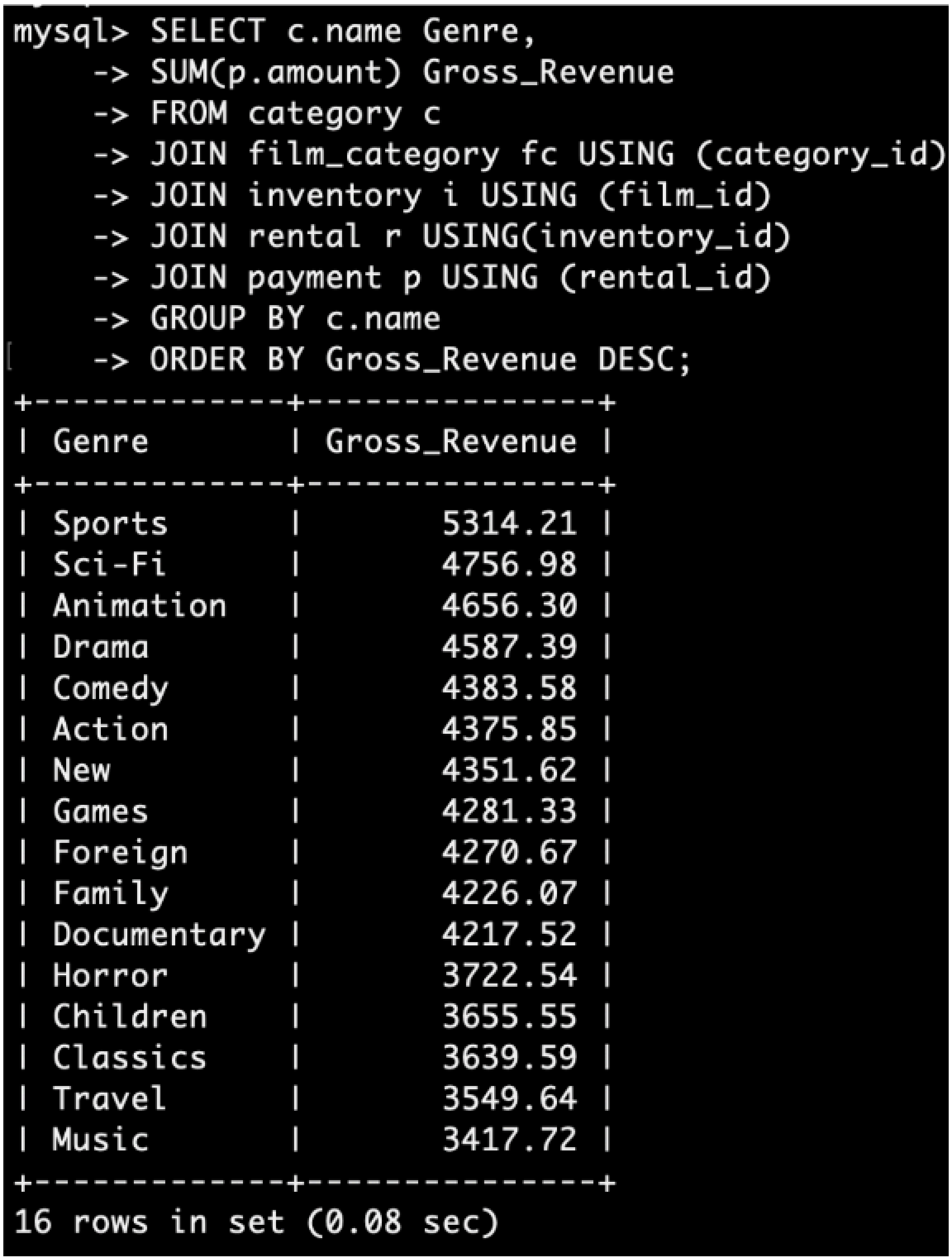
```
| MANNEQUIN WORST             |            5 |
| MUSSOLINI SPOILERS          |            5 |
| PRIVATE DROP                |            5 |
| SEVEN SWARM                 |            5 |
| TRAFFIC HOBBIT              |            5 |
| HARDLY ROBBERS              |            4 |
| MIXED DOORS                 |            4 |
| TRAIN BUNCH                 |            4 |
+-----------------------------+--------------+
958 rows in set (0.03 sec)
```

17. Write a query to display for each store its store ID, city, and country.



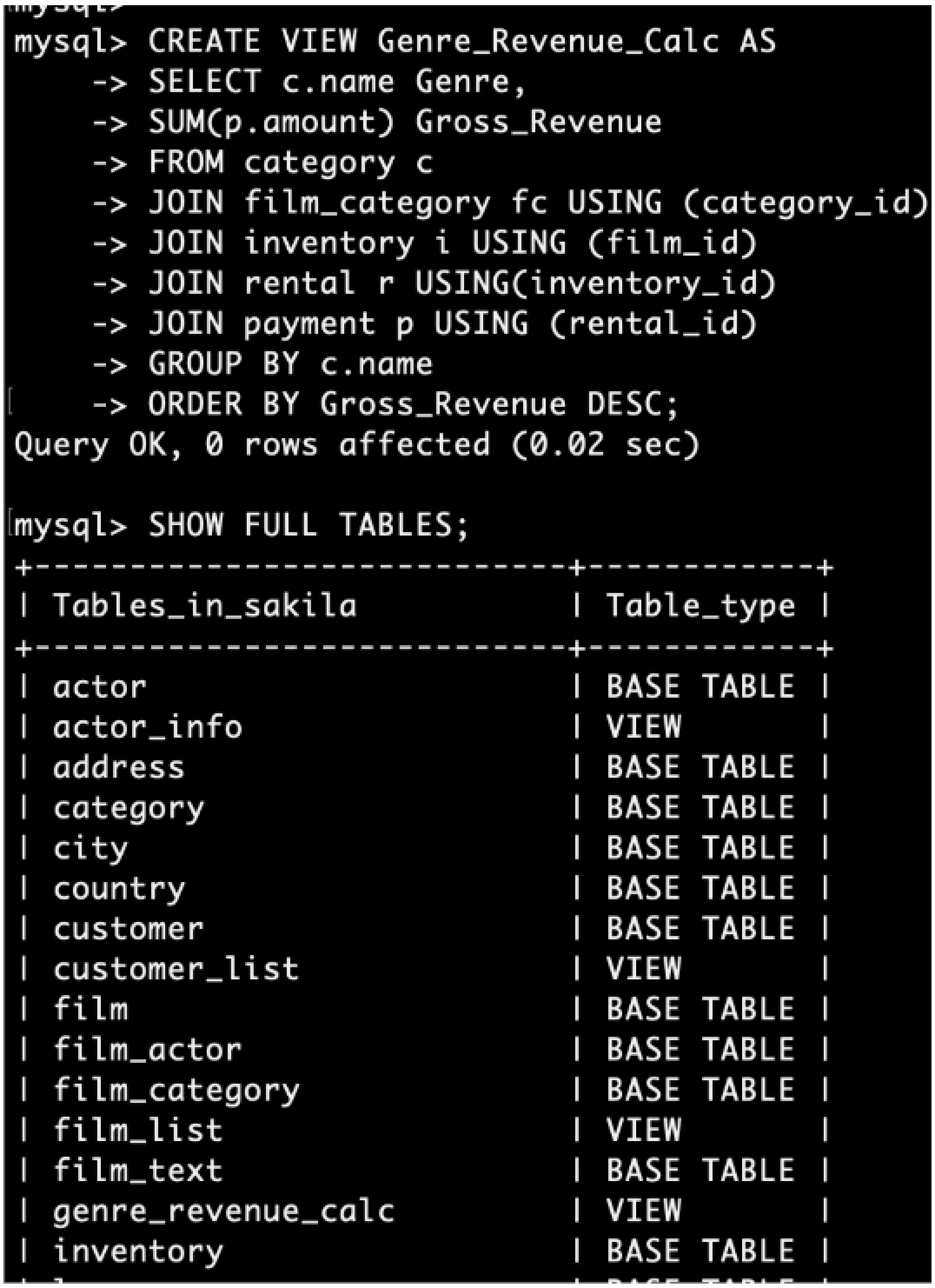
```
| MANNEQUIN WORST           |            5 |
| MUSSOLINI SPOILERS        |            5 |
| PRIVATE DROP              |            5 |
| SEVEN SWARM               |            5 |
| TRAFFIC HOBBIT            |            5 |
| HARDLY ROBBERS            |            4 |
| MIXED DOORS               |            4 |
| TRAIN BUNCH               |            4 |
+---------------------------+-------------+
958 rows in set (0.03 sec)
```

18. List the genres and its gross revenue.



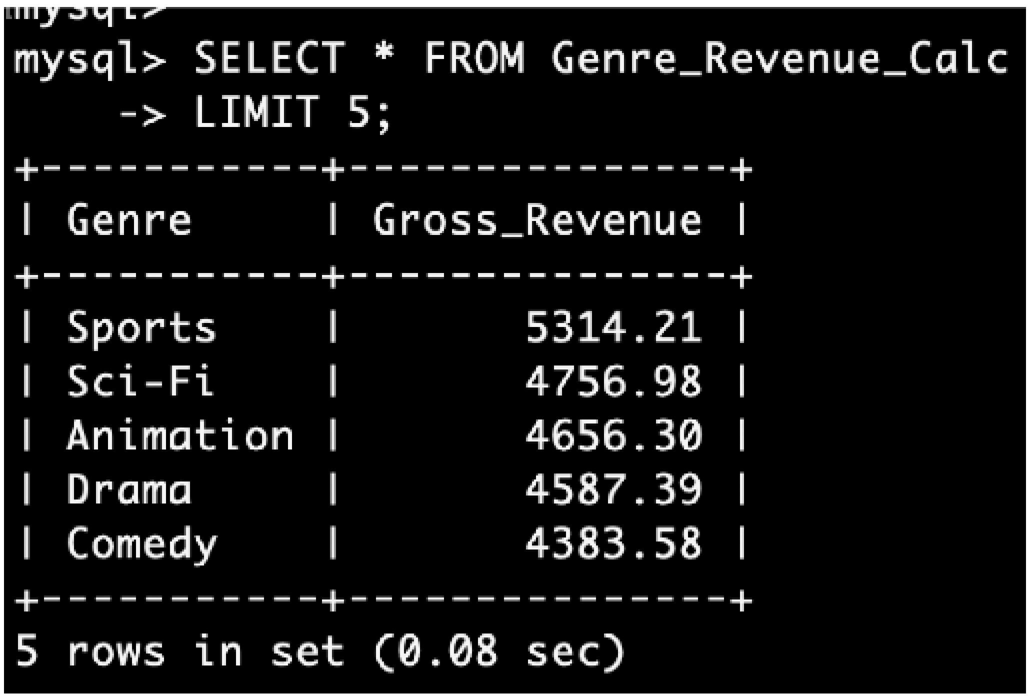
```
mysql> SELECT c.name Genre,
    -> SUM(p.amount) Gross_Revenue
    -> FROM category c
    -> JOIN film_category fc USING (category_id)
    -> JOIN inventory i USING (film_id)
    -> JOIN rental r USING(inventory_id)
    -> JOIN payment p USING (rental_id)
    -> GROUP BY c.name
    -> ORDER BY Gross_Revenue DESC;
+-------------+---------------+
| Genre       | Gross_Revenue |
+-------------+---------------+
| Sports      |       5314.21 |
| Sci-Fi      |       4756.98 |
| Animation   |       4656.30 |
| Drama       |       4587.39 |
| Comedy      |       4383.58 |
| Action      |       4375.85 |
| New         |       4351.62 |
| Games       |       4281.33 |
| Foreign     |       4270.67 |
| Family      |       4226.07 |
| Documentary |       4217.52 |
| Horror      |       3722.54 |
| Children    |       3655.55 |
| Classics    |       3639.59 |
| Travel      |       3549.64 |
| Music       |       3417.72 |
+-------------+---------------+
16 rows in set (0.08 sec)
```

19. Create a View for the above query(18)



```
mysql> CREATE VIEW Genre_Revenue_Calc AS
    -> SELECT c.name Genre,
    -> SUM(p.amount) Gross_Revenue
    -> FROM category c
    -> JOIN film_category fc USING (category_id)
    -> JOIN inventory i USING (film_id)
    -> JOIN rental r USING(inventory_id)
    -> JOIN payment p USING (rental_id)
    -> GROUP BY c.name
    -> ORDER BY Gross_Revenue DESC;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW FULL TABLES;
+----------------------------+------------+
| Tables_in_sakila           | Table_type |
+----------------------------+------------+
| actor                      | BASE TABLE |
| actor_info                 | VIEW       |
| address                    | BASE TABLE |
| category                   | BASE TABLE |
| city                       | BASE TABLE |
| country                    | BASE TABLE |
| customer                   | BASE TABLE |
| customer_list              | VIEW       |
| film                       | BASE TABLE |
| film_actor                 | BASE TABLE |
| film_category              | BASE TABLE |
| film_list                  | VIEW       |
| film_text                  | BASE TABLE |
| genre_revenue_calc         | VIEW       |
| inventory                  | BASE TABLE |
```

20. Select top 5 genres in gross revenue view.



```
mysql>
mysql> SELECT * FROM Genre_Revenue_Calc
    -> LIMIT 5;
+-----------+---------------+
| Genre     | Gross_Revenue |
+-----------+---------------+
| Sports    |       5314.21 |
| Sci-Fi    |       4756.98 |
| Animation |       4656.30 |
| Drama     |       4587.39 |
| Comedy    |       4383.58 |
+-----------+---------------+
5 rows in set (0.08 sec)
```