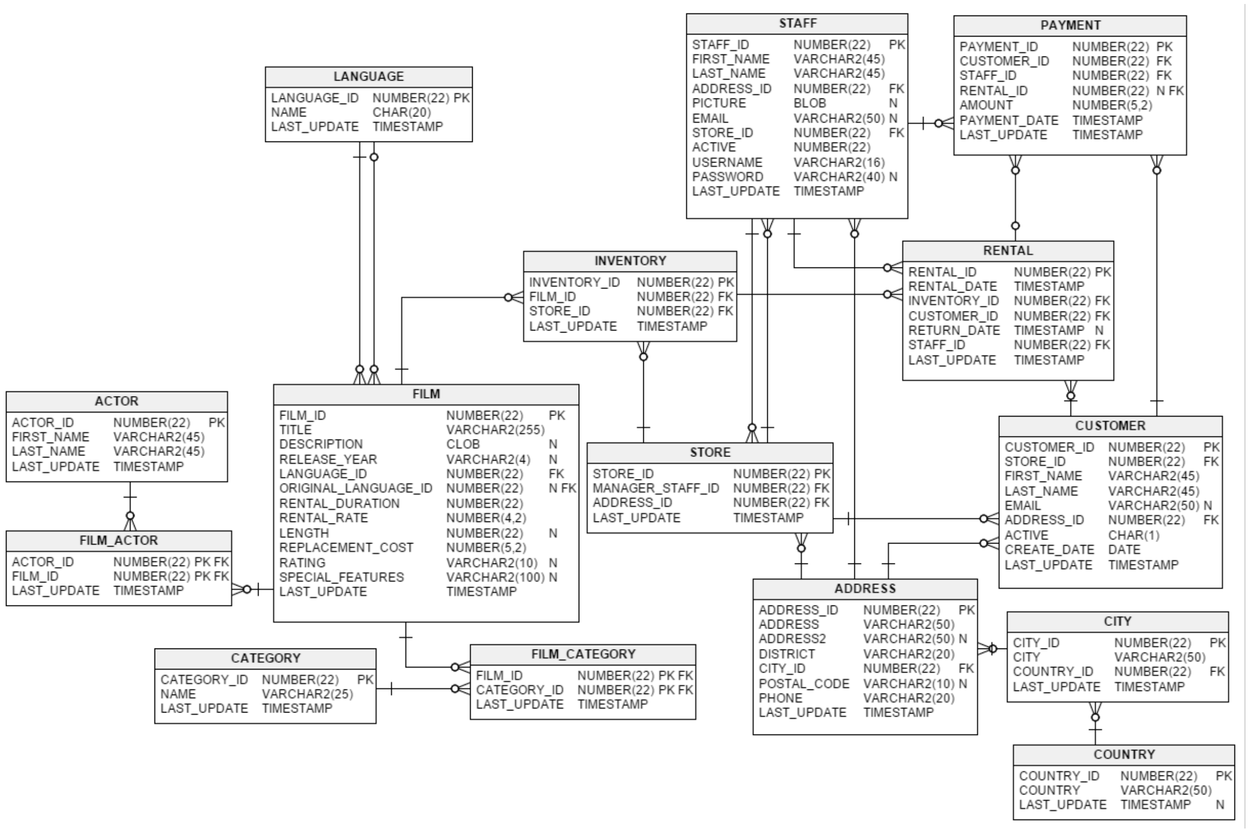
Introduction

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)
```

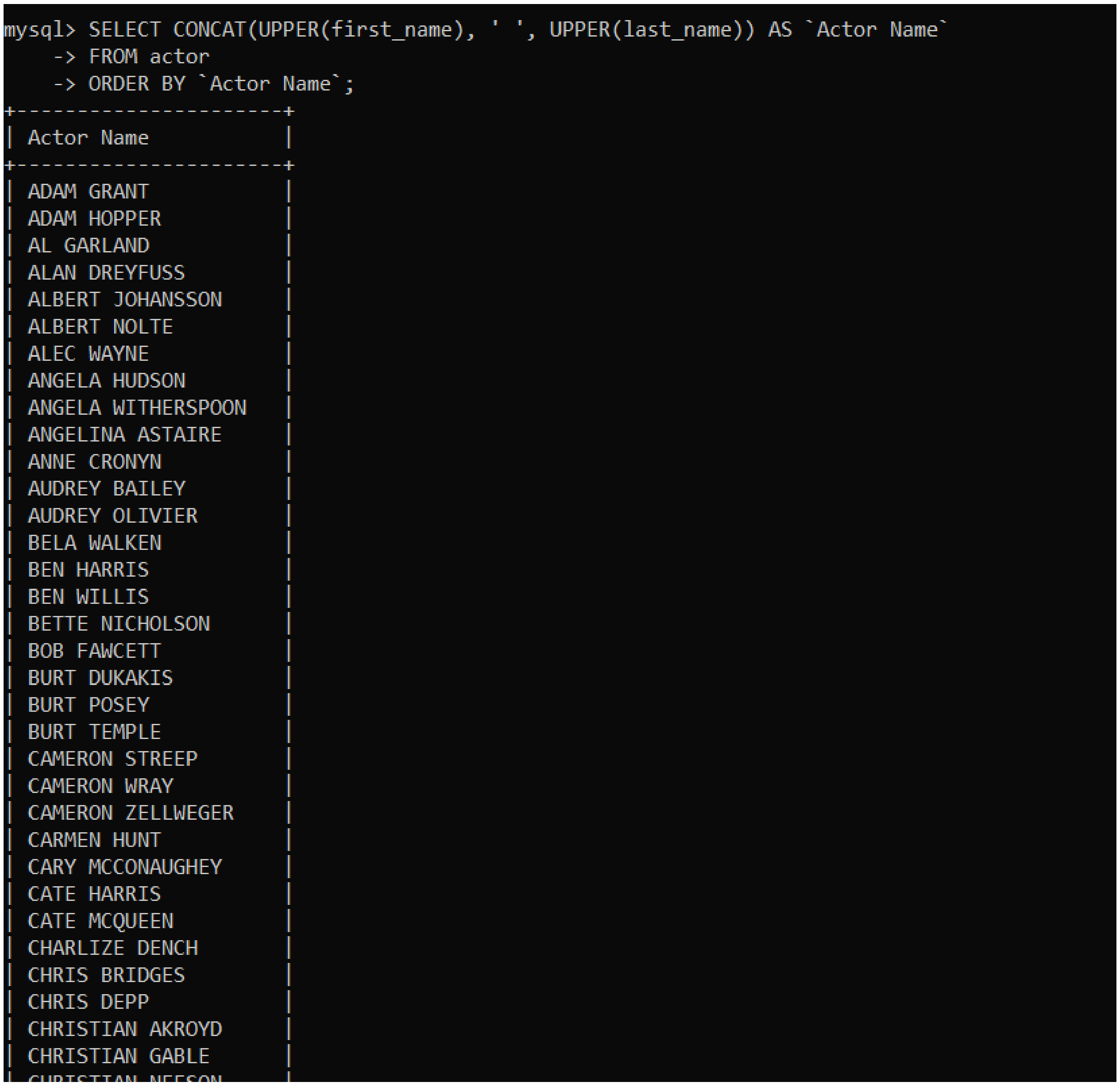
```
mysql> SELECT COUNT(*) FROM film;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

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`

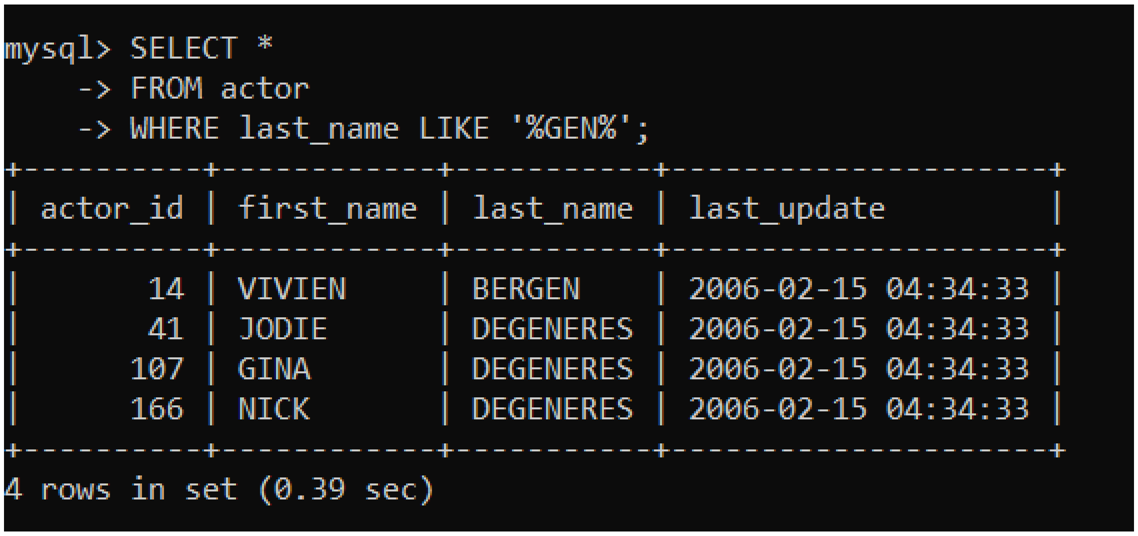
Exercises

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.



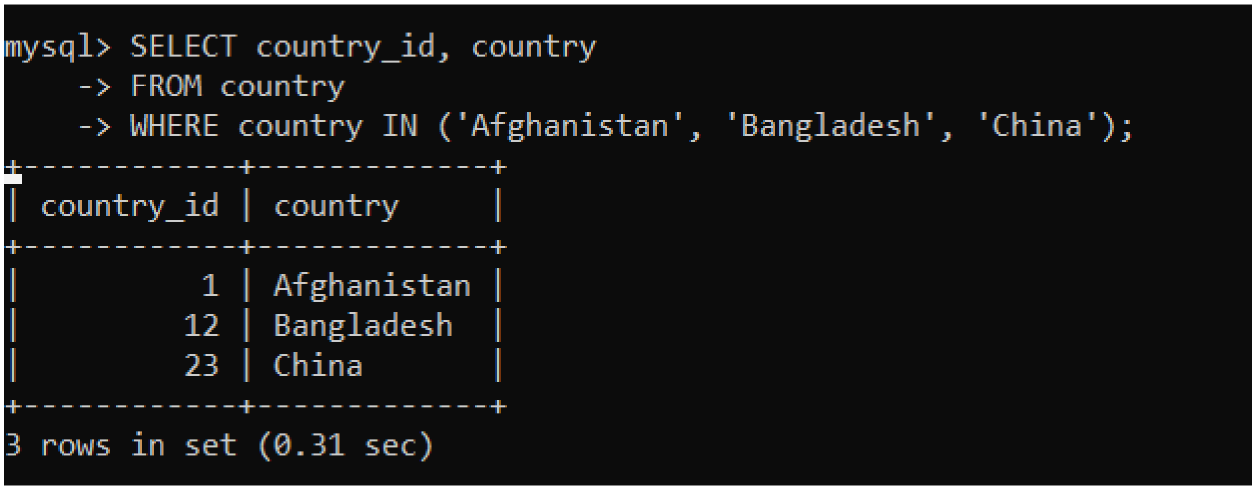
```
mysql> SELECT CONCAT(UPPER(first_name), ' ', UPPER(last_name)) AS `Actor Name`
    -> FROM actor
    -> ORDER BY `Actor Name`;
+-----------------------+
| Actor Name            |
+-----------------------+
| ADAM GRANT            |
| ADAM HOPPER           |
| AL GARLAND            |
| ALAN DREYFUSS         |
| 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           |
| CATE MCQUEEN          |
| CHARLIZE DENCH        |
| CHRIS BRIDGES         |
| CHRIS DEPP            |
| CHRISTIAN AKROYD      |
| CHRISTIAN GABLE       |
```

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



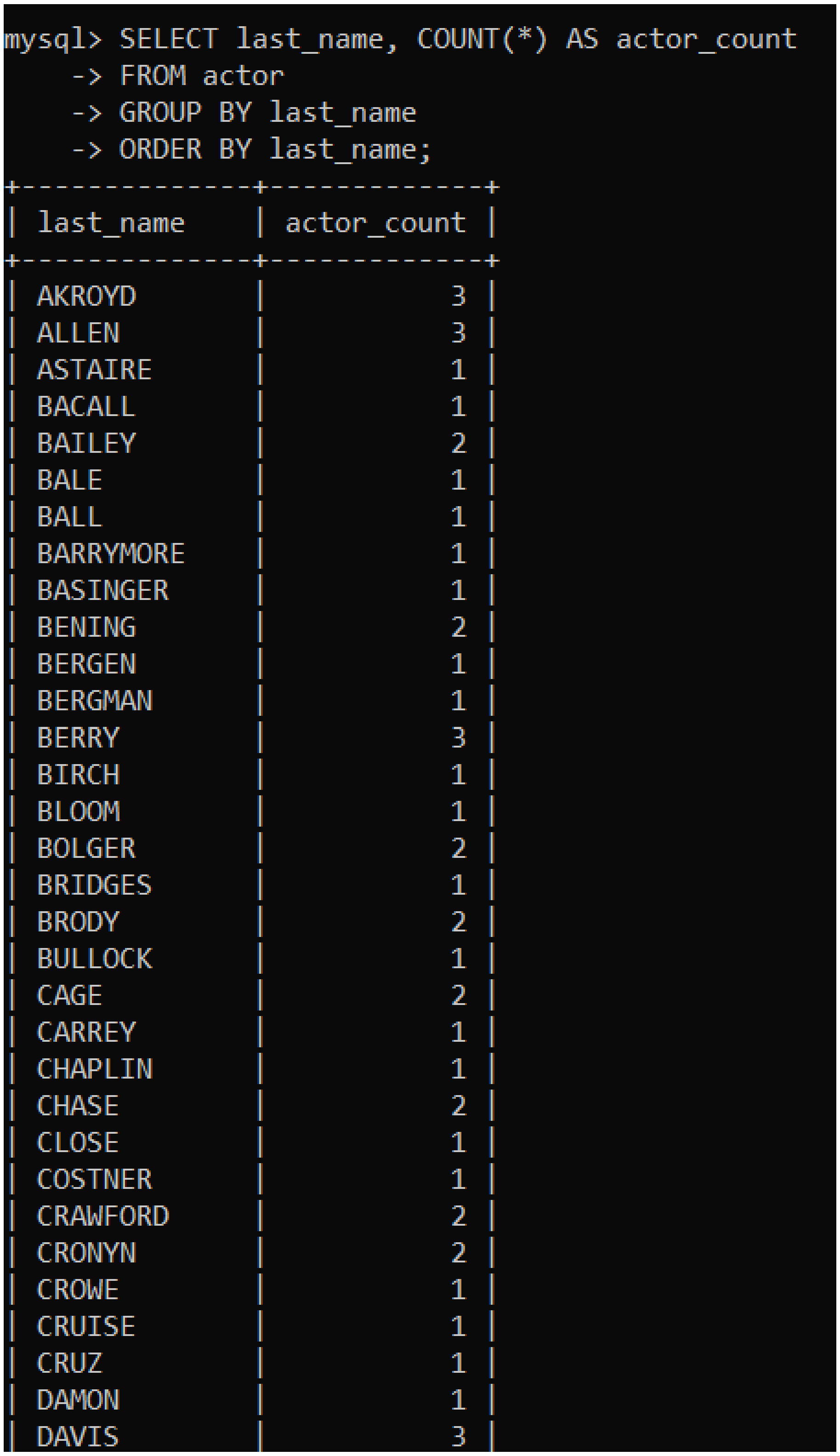
```
mysql> SELECT *
    -> FROM actor
    -> WHERE last_name LIKE '%GEN%';
+----------+------------+-----------+---------------------+
| actor_id | first_name | last_name | last_update         |
+----------+------------+-----------+---------------------+
|       14 | VIVIEN     | BERGEN    | 2006-02-15 04:34:33 |
|       41 | JODIE      | DEGENERES | 2006-02-15 04:34:33 |
|      107 | GINA       | DEGENERES | 2006-02-15 04:34:33 |
|      166 | NICK       | DEGENERES | 2006-02-15 04:34:33 |
+----------+------------+-----------+---------------------+
4 rows in set (0.39 sec)
```

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



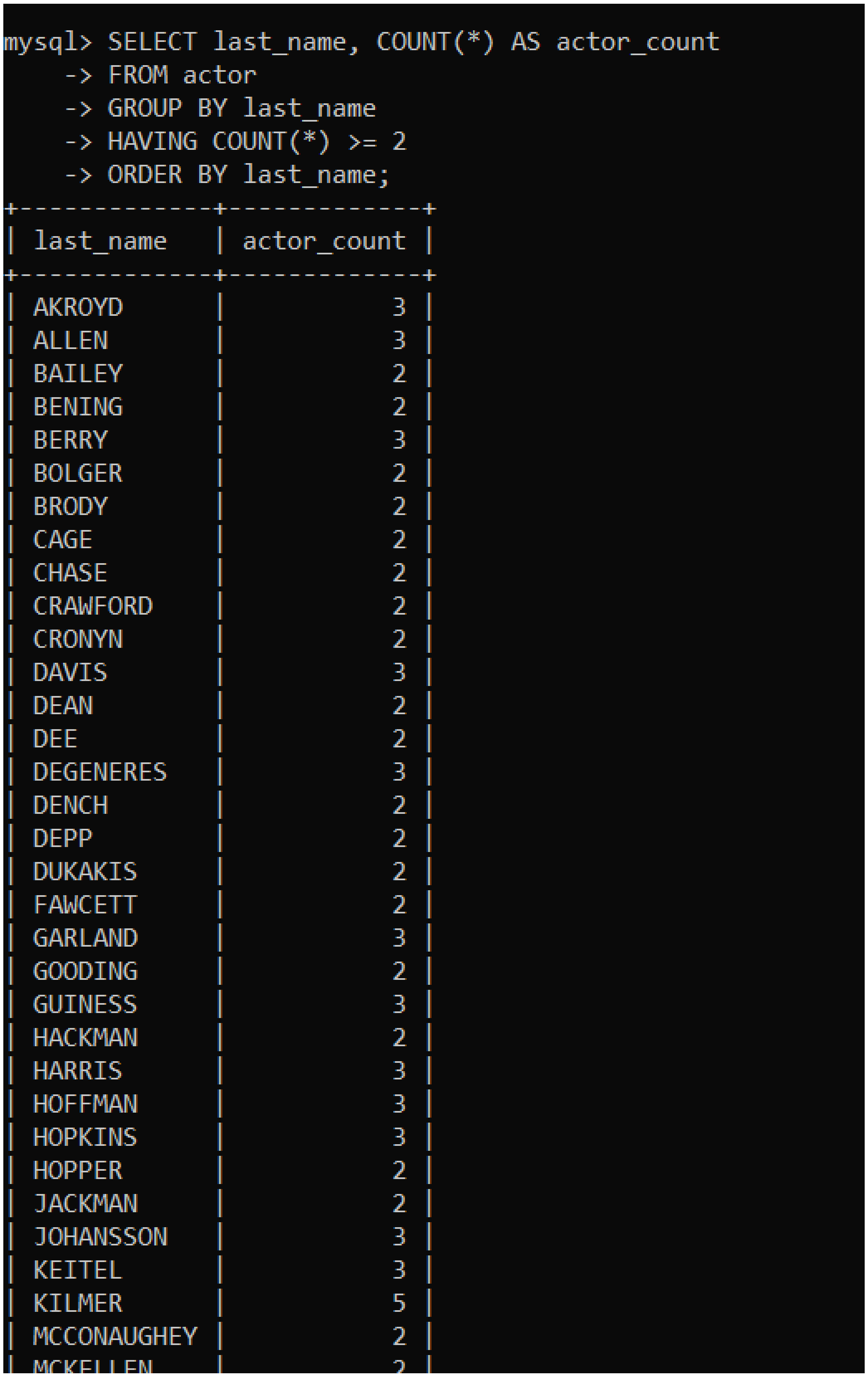
```
mysql> SELECT country_id, country
    -> FROM country
    -> WHERE country IN ('Afghanistan', 'Bangladesh', 'China');
+------------+-------------+
| country_id | country     |
+------------+-------------+
|          1 | Afghanistan |
|         12 | Bangladesh  |
|         23 | China       |
+------------+-------------+
3 rows in set (0.31 sec)
```

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



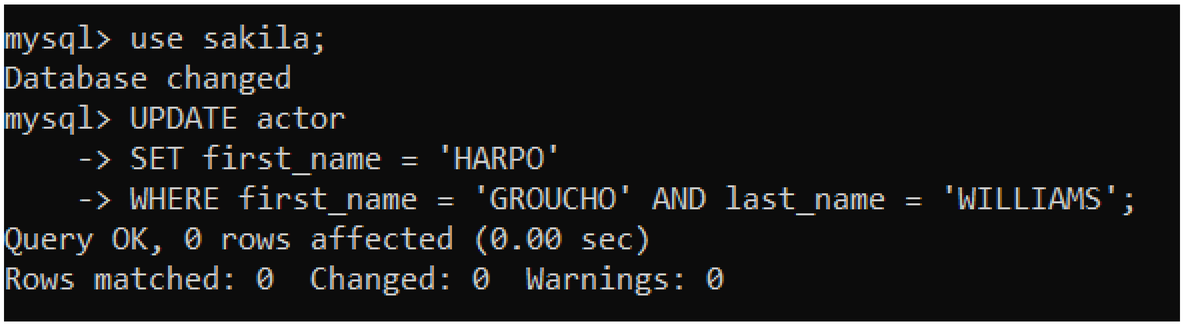
```
mysql> SELECT last_name, COUNT(*) AS actor_count
    -> FROM actor
    -> GROUP BY last_name
    -> ORDER BY last_name;
+--------------+-------------+
| last_name    | actor_count |
+--------------+-------------+
| AKROYD       |           3 |
| ALLEN        |           3 |
| ASTAIRE      |           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 |
| CARREY       |           1 |
| CHAPLIN      |           1 |
| CHASE        |           2 |
| CLOSE        |           1 |
| COSTNER      |           1 |
| CRAWFORD     |           2 |
| CRONYN       |           2 |
| CROWE        |           1 |
| CRUISE       |           1 |
| CRUZ         |           1 |
| DAMON        |           1 |
| DAVIS        |           3 |
```

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



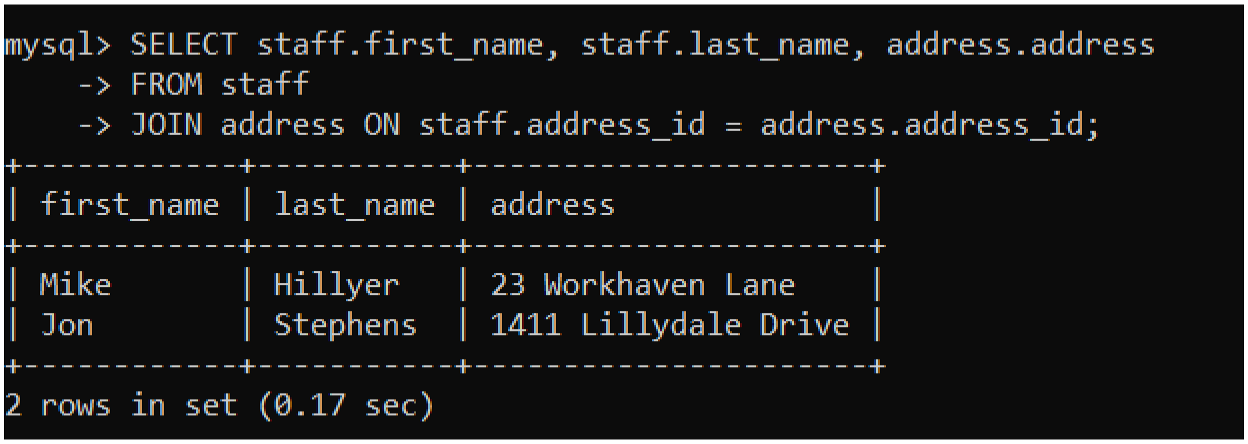
```
mysql> SELECT last_name, COUNT(*) AS actor_count
    -> FROM actor
    -> GROUP BY last_name
    -> HAVING COUNT(*) >= 2
    -> ORDER BY last_name;
+-------------+-------------+
| last_name   | actor_count |
+-------------+-------------+
| AKROYD      |           3 |
| ALLEN       |           3 |
| BAILEY      |           2 |
| BENING      |           2 |
| BERRY       |           3 |
| BOLGER      |           2 |
| BRODY       |           2 |
| CAGE        |           2 |
| CHASE       |           2 |
| CRAWFORD    |           2 |
| CRONYN      |           2 |
| DAVIS       |           3 |
| DEAN        |           2 |
| DEE         |           2 |
| DEGENERES   |           3 |
| DENCH       |           2 |
| DEPP        |           2 |
| DUKAKIS     |           2 |
| FAWCETT     |           2 |
| GARLAND     |           3 |
| GOODING     |           2 |
| GUINESS     |           3 |
| HACKMAN     |           2 |
| HARRIS      |           3 |
| HOFFMAN     |           3 |
| HOPKINS     |           3 |
| HOPPER      |           2 |
| JACKMAN     |           2 |
| JOHANSSON   |           3 |
| KEITEL      |           3 |
| KILMER      |           5 |
| MCCONAUGHEY |           2 |
| MCKELLEN    |           2 |
```

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



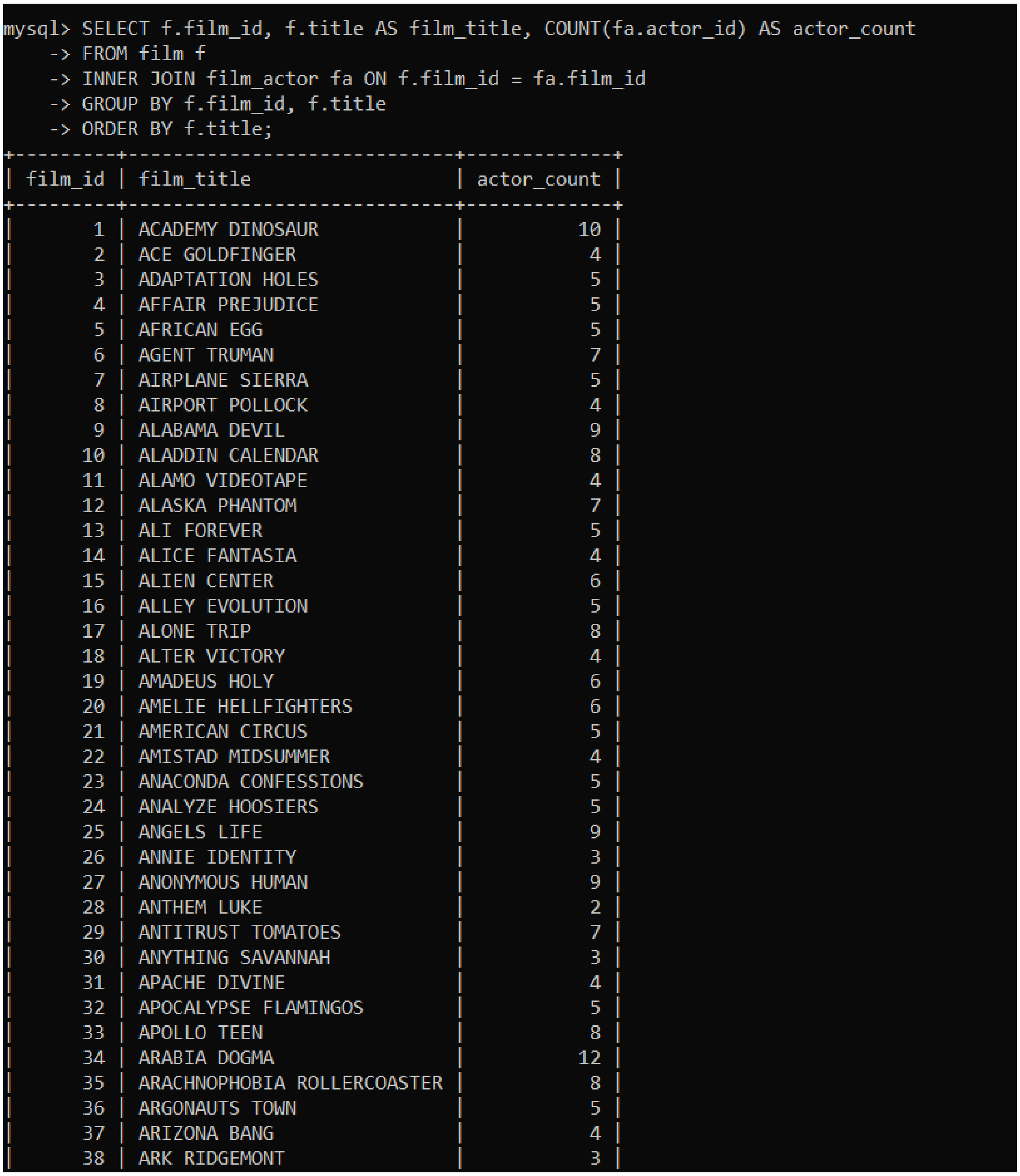
```
mysql> use sakila;
Database changed
mysql> UPDATE actor
    -> SET first_name = 'HARPO'
    -> WHERE first_name = 'GROUCHO' AND last_name = 'WILLIAMS';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

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:



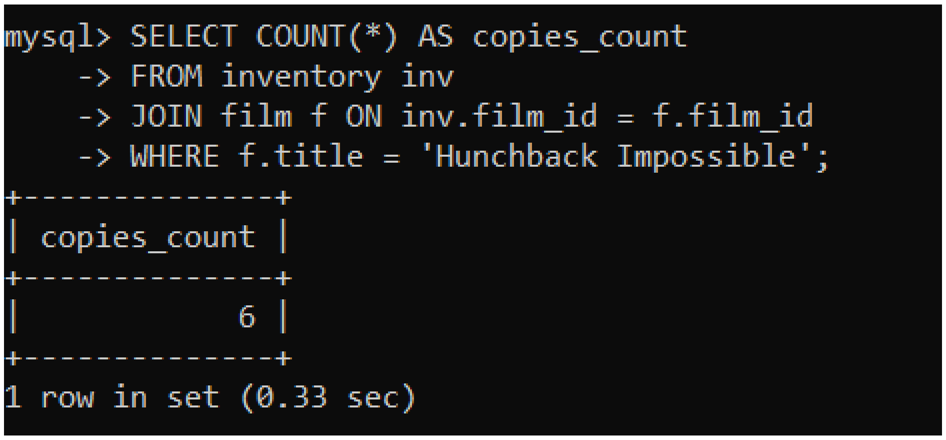
```
mysql> SELECT staff.first_name, staff.last_name, address.address
    -> FROM staff
    -> JOIN address ON staff.address_id = address.address_id;
+------------+-----------+----------------------+
| first_name | last_name | address              |
+------------+-----------+----------------------+
| Mike       | Hillyer   | 23 Workhaven Lane    |
| Jon        | Stephens  | 1411 Lillydale Drive |
+------------+-----------+----------------------+
2 rows in set (0.17 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.



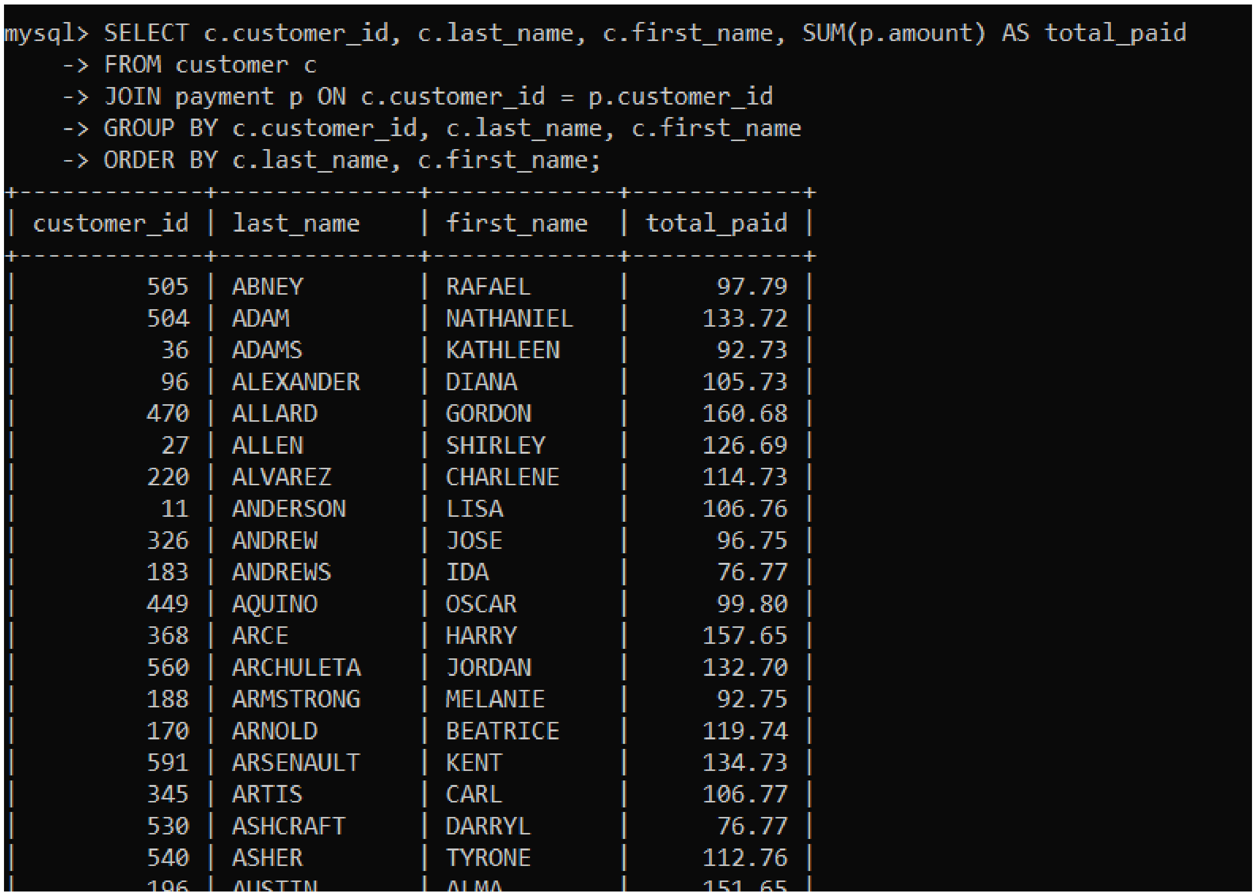
```
mysql> SELECT f.film_id, f.title AS film_title, COUNT(fa.actor_id) AS actor_count
    -> FROM film f
    -> INNER JOIN film_actor fa ON f.film_id = fa.film_id
    -> GROUP BY f.film_id, f.title
    -> ORDER BY f.title;
+---------+-----------------------------+-------------+
| film_id | film_title                  | actor_count |
+---------+-----------------------------+-------------+
|       1 | ACADEMY DINOSAUR            |          10 |
|       2 | ACE GOLDFINGER              |           4 |
|       3 | ADAPTATION HOLES            |           5 |
|       4 | AFFAIR PREJUDICE            |           5 |
|       5 | AFRICAN EGG                 |           5 |
|       6 | AGENT TRUMAN                |           7 |
|       7 | AIRPLANE SIERRA             |           5 |
|       8 | AIRPORT POLLOCK             |           4 |
|       9 | ALABAMA DEVIL               |           9 |
|      10 | ALADDIN CALENDAR            |           8 |
|      11 | ALAMO VIDEOTAPE             |           4 |
|      12 | ALASKA PHANTOM              |           7 |
|      13 | ALI FOREVER                 |           5 |
|      14 | ALICE FANTASIA              |           4 |
|      15 | ALIEN CENTER                |           6 |
|      16 | ALLEY EVOLUTION             |           5 |
|      17 | ALONE TRIP                  |           8 |
|      18 | ALTER VICTORY               |           4 |
|      19 | AMADEUS HOLY                |           6 |
|      20 | AMELIE HELLFIGHTERS         |           6 |
|      21 | AMERICAN CIRCUS             |           5 |
|      22 | AMISTAD MIDSUMMER           |           4 |
|      23 | ANACONDA CONFESSIONS        |           5 |
|      24 | ANALYZE HOOSIERS            |           5 |
|      25 | ANGELS LIFE                 |           9 |
|      26 | ANNIE IDENTITY              |           3 |
|      27 | ANONYMOUS HUMAN             |           9 |
|      28 | ANTHEM LUKE                 |           2 |
|      29 | ANTITRUST TOMATOES          |           7 |
|      30 | ANYTHING SAVANNAH           |           3 |
|      31 | APACHE DIVINE               |           4 |
|      32 | APOCALYPSE FLAMINGOS        |           5 |
|      33 | APOLLO TEEN                 |           8 |
|      34 | ARABIA DOGMA                |          12 |
|      35 | ARACHNOPHOBIA ROLLERCOASTER |           8 |
|      36 | ARGONAUTS TOWN              |           5 |
|      37 | ARIZONA BANG                |           4 |
|      38 | ARK RIDGEMONT               |           3 |
```

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



```
mysql> SELECT COUNT(*) AS copies_count
    -> FROM inventory inv
    -> JOIN film f ON inv.film_id = f.film_id
    -> WHERE f.title = 'Hunchback Impossible';
+--------------+
| copies_count |
+--------------+
|            6 |
+--------------+
1 row in set (0.33 sec)
```

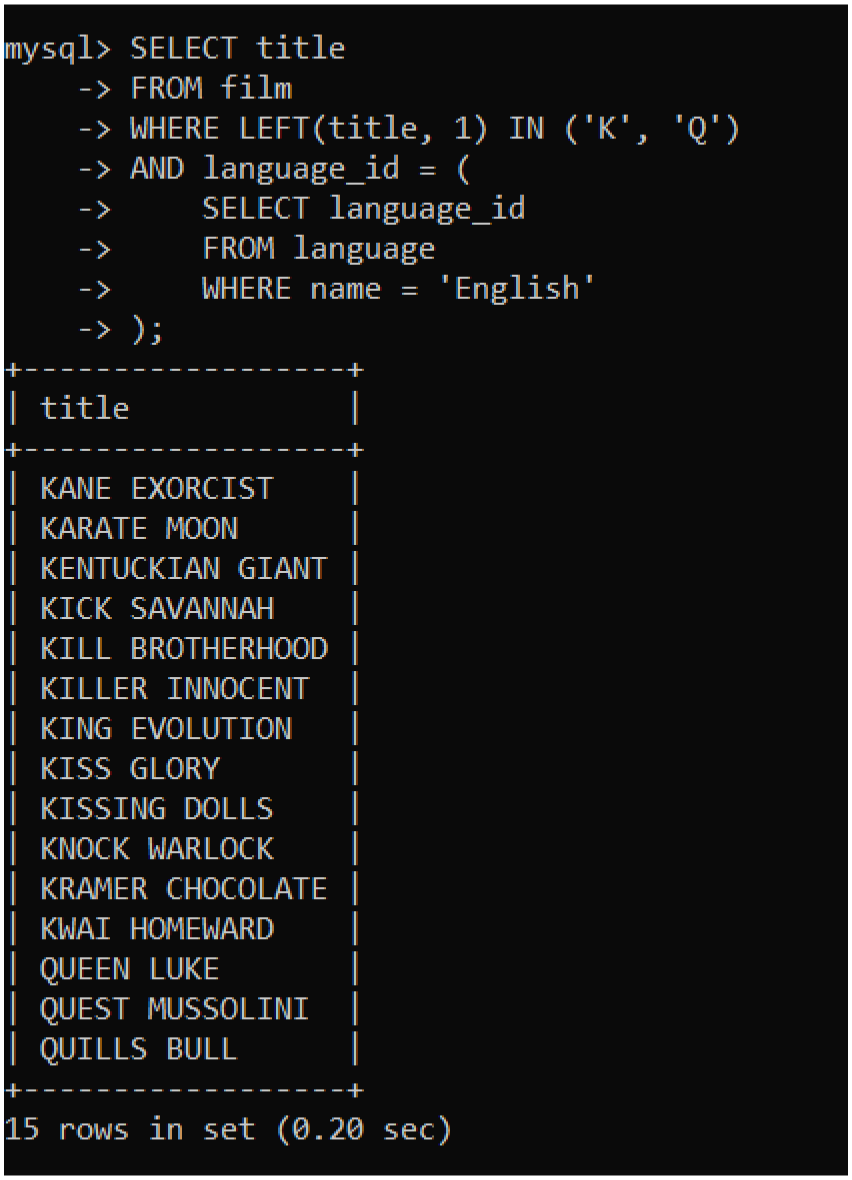
10. Using the tables payment and customer and the JOIN command, list the total paid by each customer. List the customers alphabetically by last name



```
mysql> SELECT c.customer_id, c.last_name, c.first_name, SUM(p.amount) AS total_paid
    -> FROM customer c
    -> JOIN payment p ON c.customer_id = p.customer_id
    -> GROUP BY c.customer_id, c.last_name, c.first_name
    -> ORDER BY c.last_name, c.first_name;
+-------------+--------------+-------------+------------+
| customer_id | last_name    | first_name  | total_paid |
+-------------+--------------+-------------+------------+
|         505 | ABNEY        | RAFAEL      |      97.79 |
|         504 | ADAM         | NATHANIEL   |     133.72 |
|          36 | ADAMS        | KATHLEEN    |      92.73 |
|          96 | ALEXANDER    | DIANA       |     105.73 |
|         470 | ALLARD       | GORDON      |     160.68 |
|          27 | ALLEN        | SHIRLEY     |     126.69 |
|         220 | ALVAREZ      | CHARLENE    |     114.73 |
|          11 | ANDERSON     | LISA        |     106.76 |
|         326 | ANDREW       | JOSE        |      96.75 |
|         183 | ANDREWS      | IDA         |      76.77 |
|         449 | AQUINO       | OSCAR       |      99.80 |
|         368 | ARCE         | HARRY       |     157.65 |
|         560 | ARCHULETA    | JORDAN      |     132.70 |
|         188 | ARMSTRONG    | MELANIE     |      92.75 |
|         170 | ARNOLD       | BEATRICE    |     119.74 |
|         591 | ARSENAULT    | KENT        |     134.73 |
|         345 | ARTIS        | CARL        |     106.77 |
|         530 | ASHCRAFT     | DARRYL      |      76.77 |
|         540 | ASHER        | TYRONE      |     112.76 |
|         196 | AUSTIN       | ALMA        |     151.65 |
```

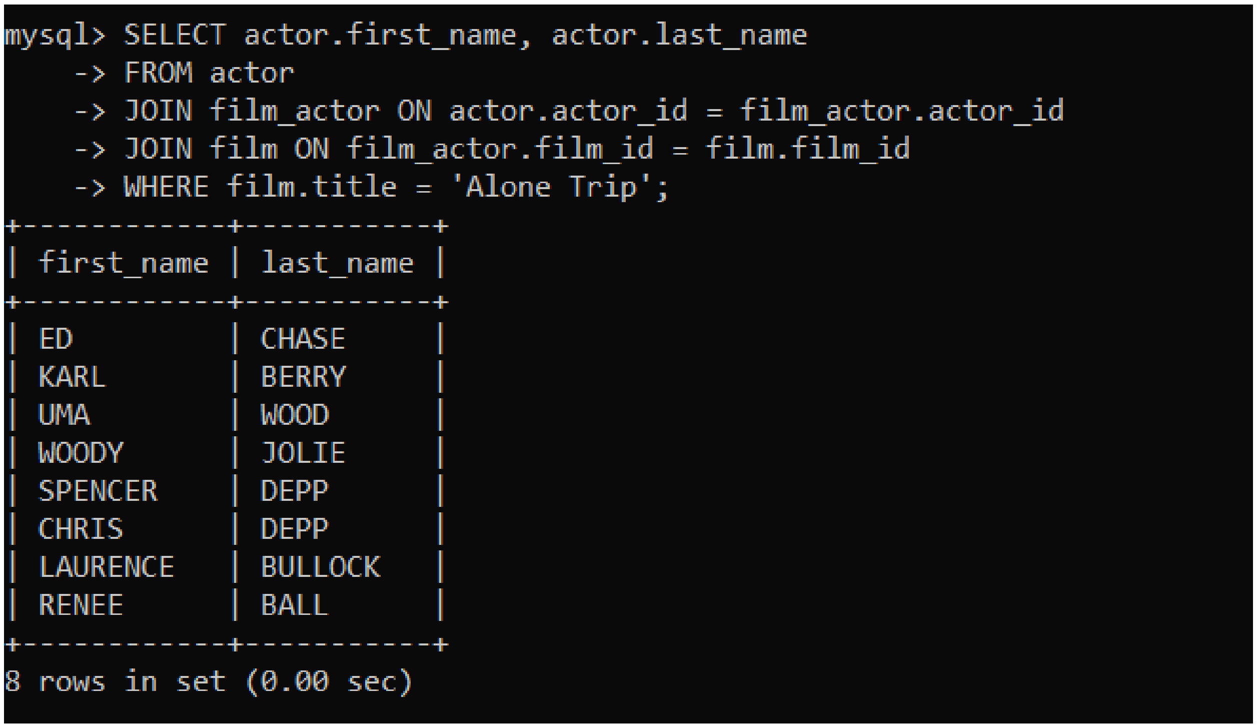
11. The music of Queen and Kris Kristofferson have seen an unlikely resurgence. As an unintended consequence, films starting with the letters `K` and `Q` have also soared in popularity. Use subqueries to display the titles of movies starting with

the letters `K` and `Q` whose language is English.



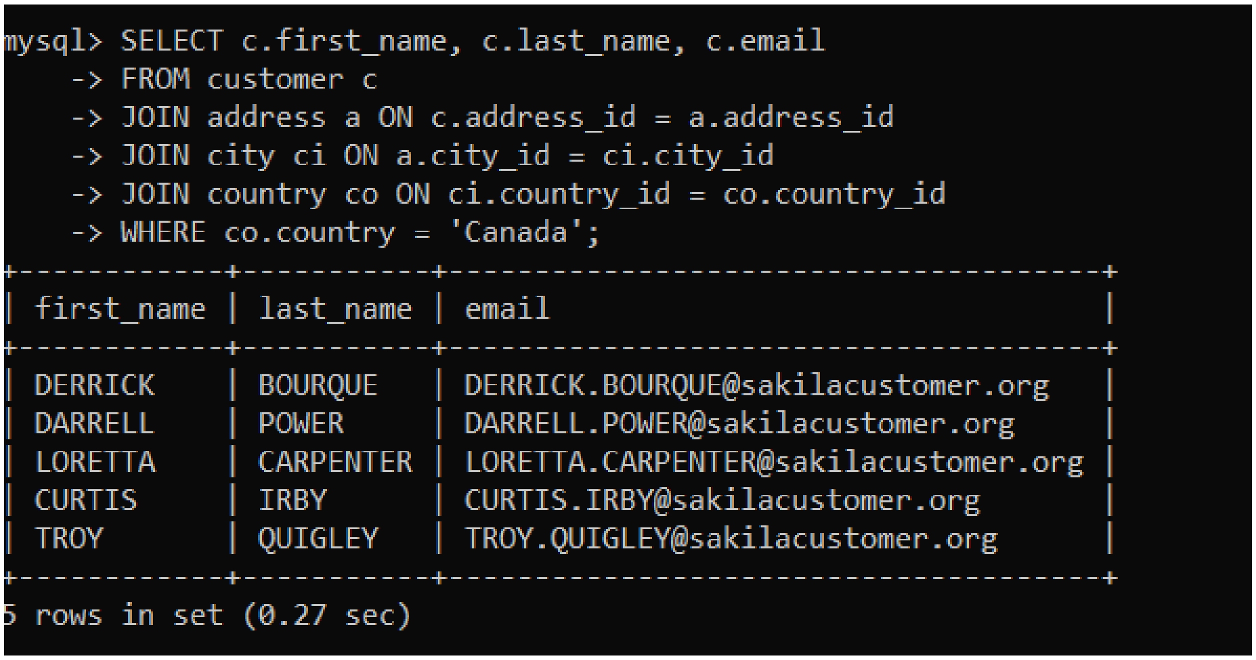
```
mysql> SELECT title
    -> FROM film
    -> WHERE LEFT(title, 1) IN ('K', 'Q')
    -> AND language_id = (
    ->     SELECT language_id
    ->     FROM language
    ->     WHERE name = 'English'
    -> );
+-------------------+
| title             |
+-------------------+
| KANE EXORCIST     |
| KARATE 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 (0.20 sec)
```

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



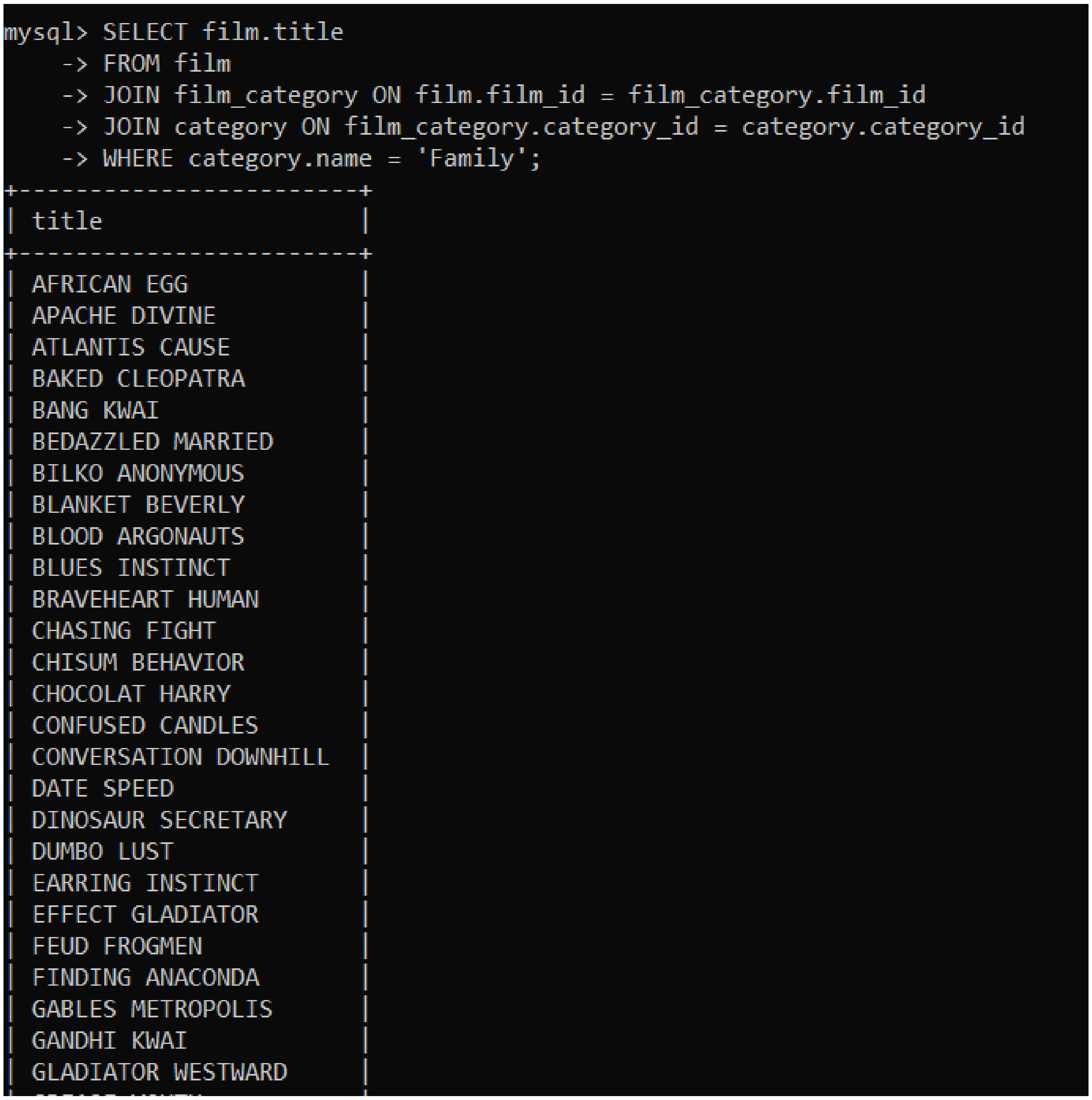
```
mysql> SELECT actor.first_name, actor.last_name
    -> FROM actor
    -> JOIN film_actor ON actor.actor_id = film_actor.actor_id
    -> JOIN film ON film_actor.film_id = film.film_id
    -> WHERE film.title = 'Alone Trip';
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| 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.



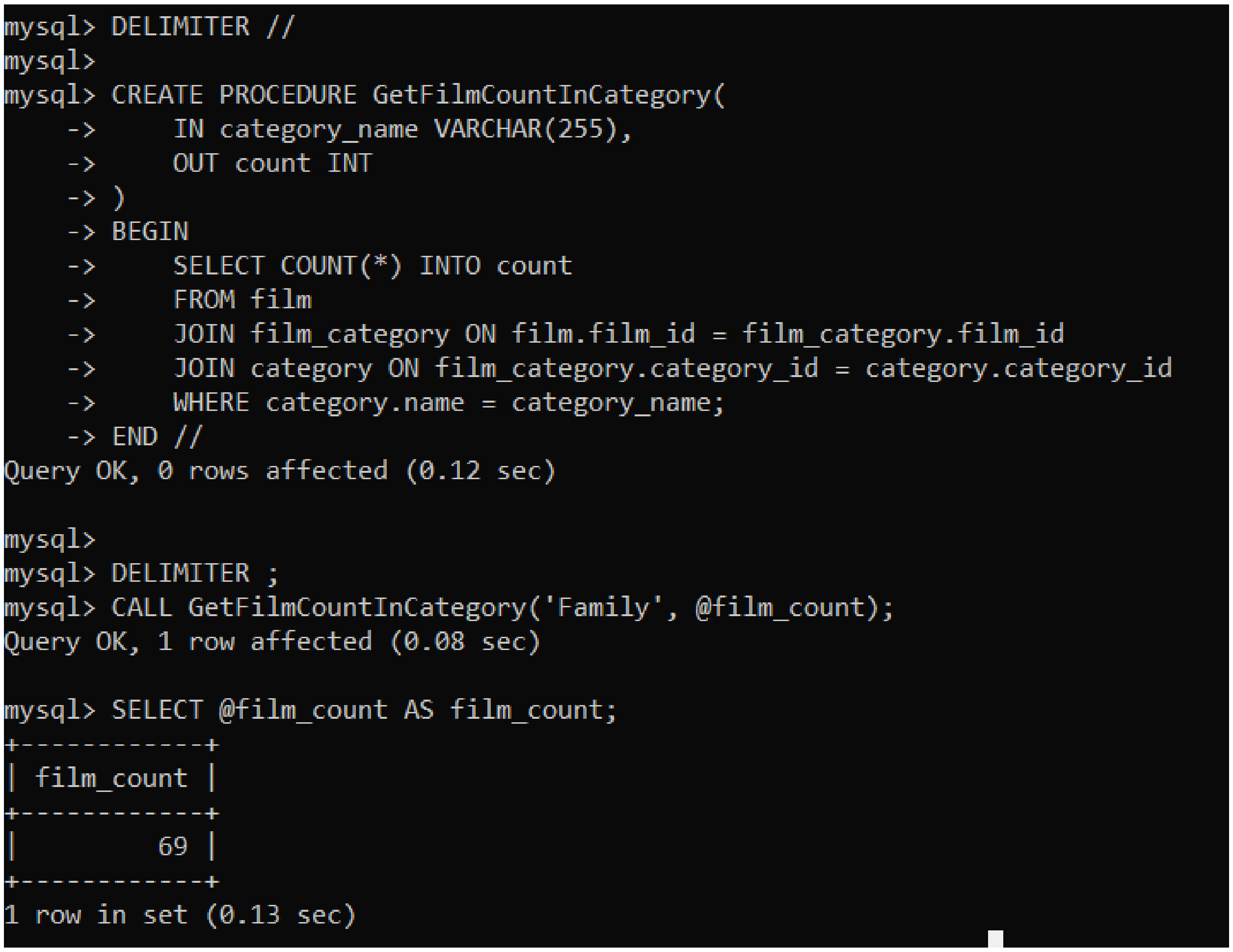
```
mysql> SELECT c.first_name, c.last_name, c.email
    -> FROM customer c
    -> JOIN address a ON c.address_id = a.address_id
    -> JOIN city ci ON a.city_id = ci.city_id
    -> JOIN country co ON ci.country_id = co.country_id
    -> WHERE co.country = 'Canada';
+------------+-----------+-------------------------------------+
| first_name | last_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 (0.27 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.



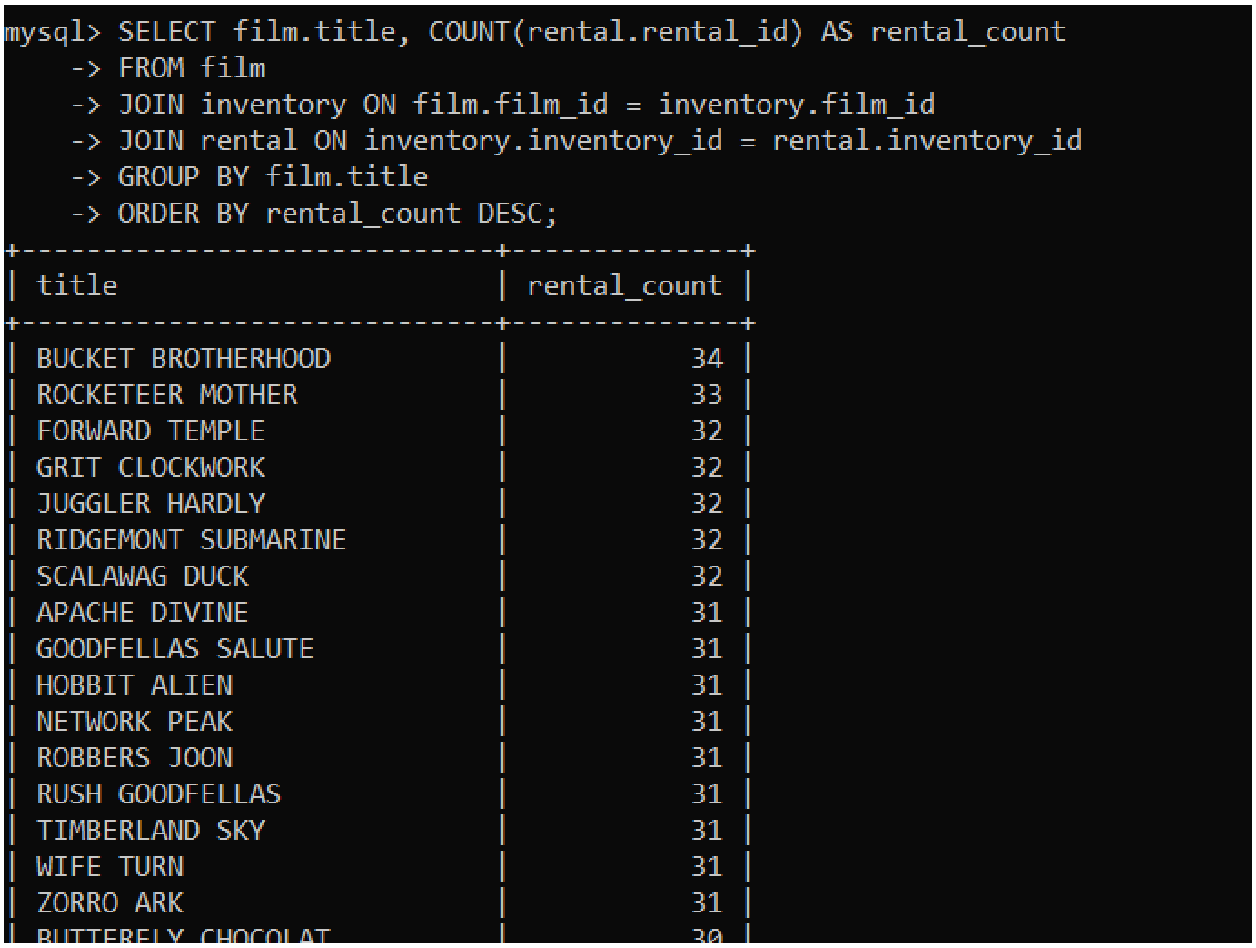
```
mysql> SELECT film.title
    -> FROM film
    -> JOIN film_category ON film.film_id = film_category.film_id
    -> JOIN category ON film_category.category_id = category.category_id
    -> WHERE category.name = 'Family';
+-------------------------+
| title                   |
+-------------------------+
| AFRICAN EGG             |
| APACHE DIVINE           |
| ATLANTIS CAUSE          |
| BAKED CLEOPATRA         |
| BANG KWAI               |
| BEDAZZLED MARRIED       |
| BILKO ANONYMOUS         |
| BLANKET BEVERLY         |
| BLOOD ARGONAUTS         |
| BLUES INSTINCT          |
| BRAVEHEART HUMAN        |
| CHASING FIGHT           |
| CHISUM BEHAVIOR         |
| CHOCOLAT HARRY          |
| CONFUSED CANDLES        |
| CONVERSATION DOWNHILL   |
| DATE SPEED              |
| DINOSAUR SECRETARY      |
| DUMBO LUST              |
| EARRING INSTINCT        |
| EFFECT GLADIATOR        |
| FEUD FROGMEN            |
| FINDING ANACONDA        |
| GABLES METROPOLIS       |
| GANDHI KWAI             |
| GLADIATOR WESTWARD      |
```

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



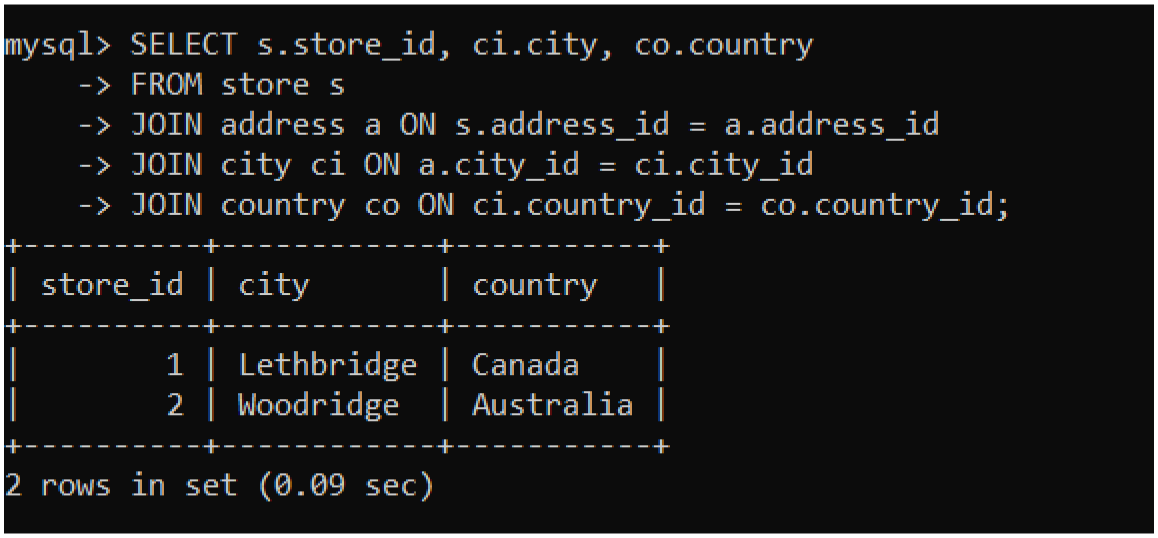
```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE GetFilmCountInCategory(
    ->     IN category_name VARCHAR(255),
    ->     OUT count INT
    -> )
    -> BEGIN
    ->     SELECT COUNT(*) INTO count
    ->     FROM film
    ->     JOIN film_category ON film.film_id = film_category.film_id
    ->     JOIN category ON film_category.category_id = category.category_id
    ->     WHERE category.name = category_name;
    -> END //
Query OK, 0 rows affected (0.12 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL GetFilmCountInCategory('Family', @film_count);
Query OK, 1 row affected (0.08 sec)

mysql> SELECT @film_count AS film_count;
+------------+
| film_count |
+------------+
|         69 |
+------------+
1 row in set (0.13 sec)
```

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



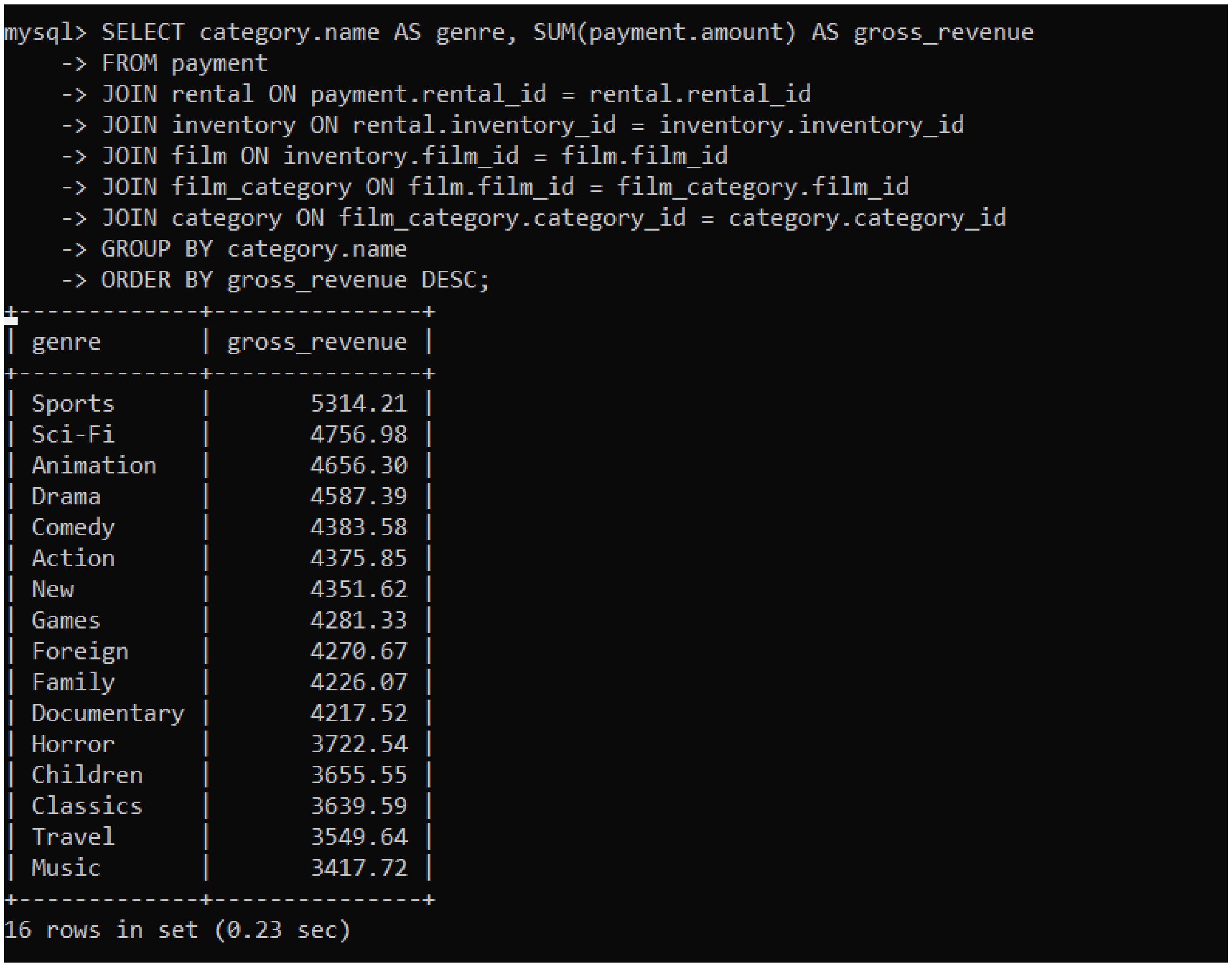
```
mysql> SELECT film.title, COUNT(rental.rental_id) AS rental_count
    -> FROM film
    -> JOIN inventory ON film.film_id = inventory.film_id
    -> JOIN rental ON inventory.inventory_id = rental.inventory_id
    -> GROUP BY film.title
    -> ORDER BY rental_count DESC;
+------------------------------+--------------+
| title                        | 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 |
| TIMBERLAND SKY               |           31 |
| WIFE TURN                    |           31 |
| ZORRO ARK                    |           31 |
| BUTTERFLY CHOCOLAT           |           30 |
```

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



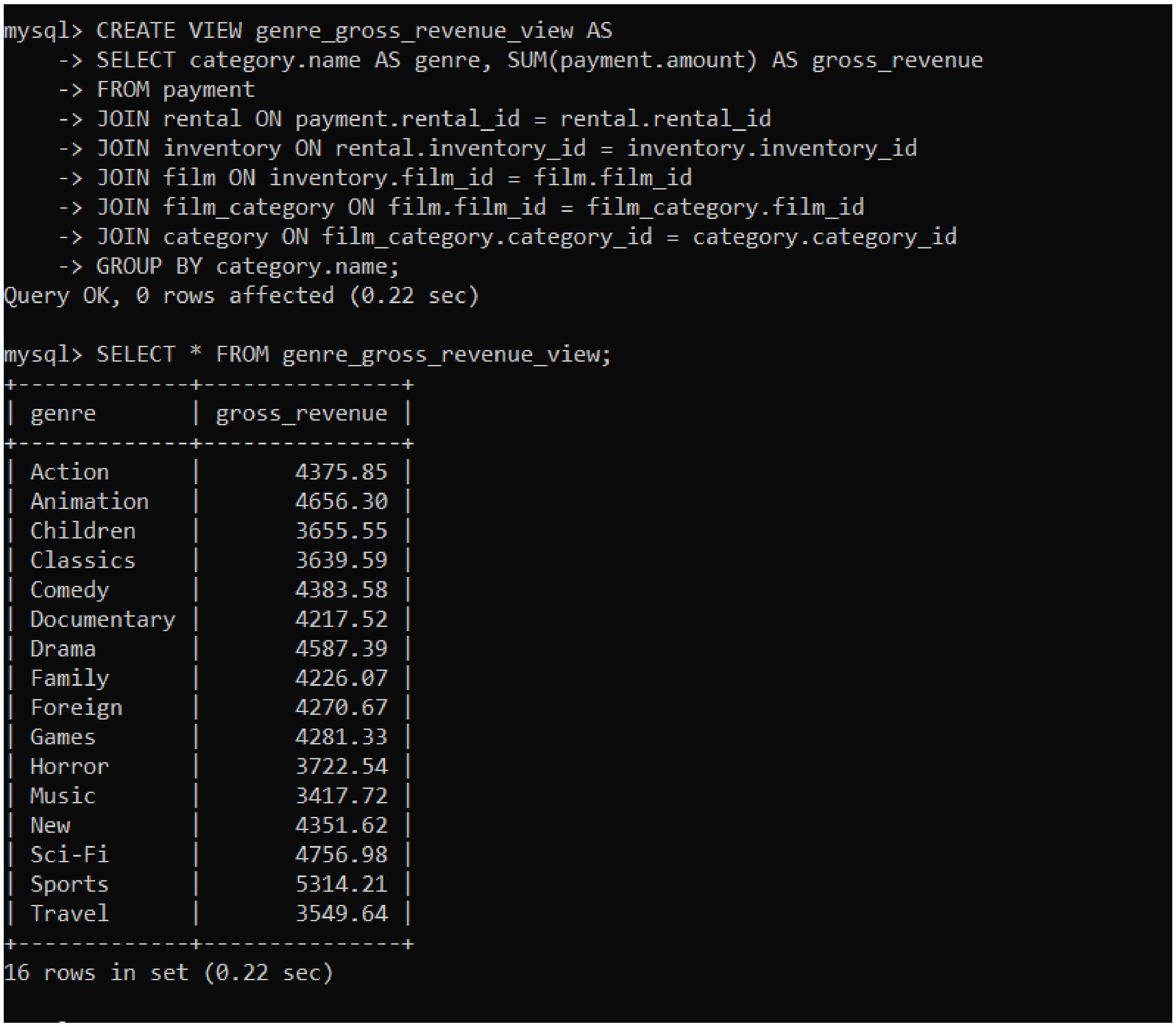
```
mysql> SELECT s.store_id, ci.city, co.country
    -> FROM store s
    -> JOIN address a ON s.address_id = a.address_id
    -> JOIN city ci ON a.city_id = ci.city_id
    -> JOIN country co ON ci.country_id = co.country_id;
+----------+------------+-----------+
| store_id | city       | country   |
+----------+------------+-----------+
|        1 | Lethbridge | Canada    |
|        2 | Woodridge  | Australia |
+----------+------------+-----------+
2 rows in set (0.09 sec)
```

18. List the genres and its gross revenue.



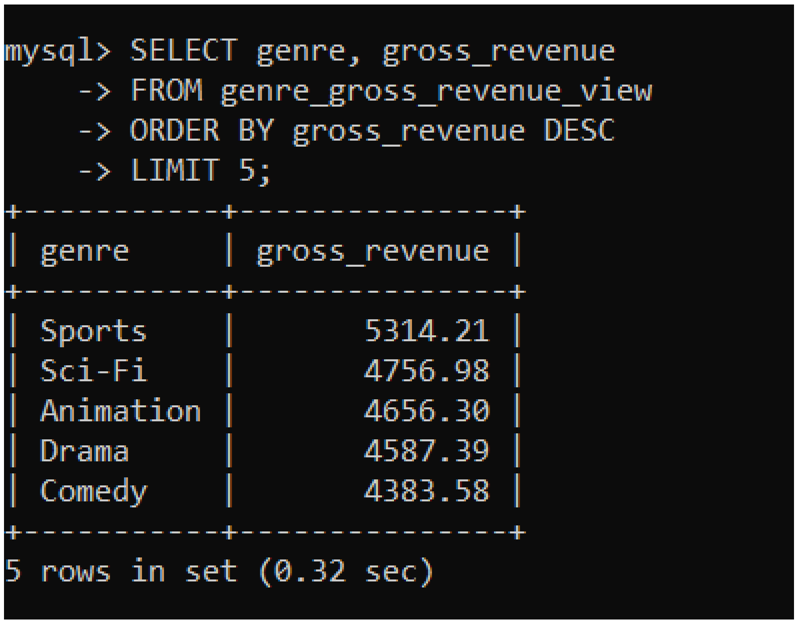
```
mysql> SELECT category.name AS genre, SUM(payment.amount) AS gross_revenue
    -> FROM payment
    -> JOIN rental ON payment.rental_id = rental.rental_id
    -> JOIN inventory ON rental.inventory_id = inventory.inventory_id
    -> JOIN film ON inventory.film_id = film.film_id
    -> JOIN film_category ON film.film_id = film_category.film_id
    -> JOIN category ON film_category.category_id = category.category_id
    -> GROUP BY category.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 (0.23 sec)
```

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



```
mysql> CREATE VIEW genre_gross_revenue_view AS
    -> SELECT category.name AS genre, SUM(payment.amount) AS gross_revenue
    -> FROM payment
    -> JOIN rental ON payment.rental_id = rental.rental_id
    -> JOIN inventory ON rental.inventory_id = inventory.inventory_id
    -> JOIN film ON inventory.film_id = film.film_id
    -> JOIN film_category ON film.film_id = film_category.film_id
    -> JOIN category ON film_category.category_id = category.category_id
    -> GROUP BY category.name;
Query OK, 0 rows affected (0.22 sec)

mysql> SELECT * FROM genre_gross_revenue_view;
+-------------+---------------+
| genre       | gross_revenue |
+-------------+---------------+
| Action      |       4375.85 |
| Animation   |       4656.30 |
| Children    |       3655.55 |
| Classics    |       3639.59 |
| Comedy      |       4383.58 |
| Documentary |       4217.52 |
| Drama       |       4587.39 |
| Family      |       4226.07 |
| Foreign     |       4270.67 |
| Games       |       4281.33 |
| Horror      |       3722.54 |
| Music       |       3417.72 |
| New         |       4351.62 |
| Sci-Fi      |       4756.98 |
| Sports      |       5314.21 |
| Travel      |       3549.64 |
+-------------+---------------+
16 rows in set (0.22 sec)
```

20. Select top 5 genres in gross revenue view.



```
mysql> SELECT genre, gross_revenue
    -> FROM genre_gross_revenue_view
    -> ORDER BY gross_revenue DESC
    -> 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 (0.32 sec)
```