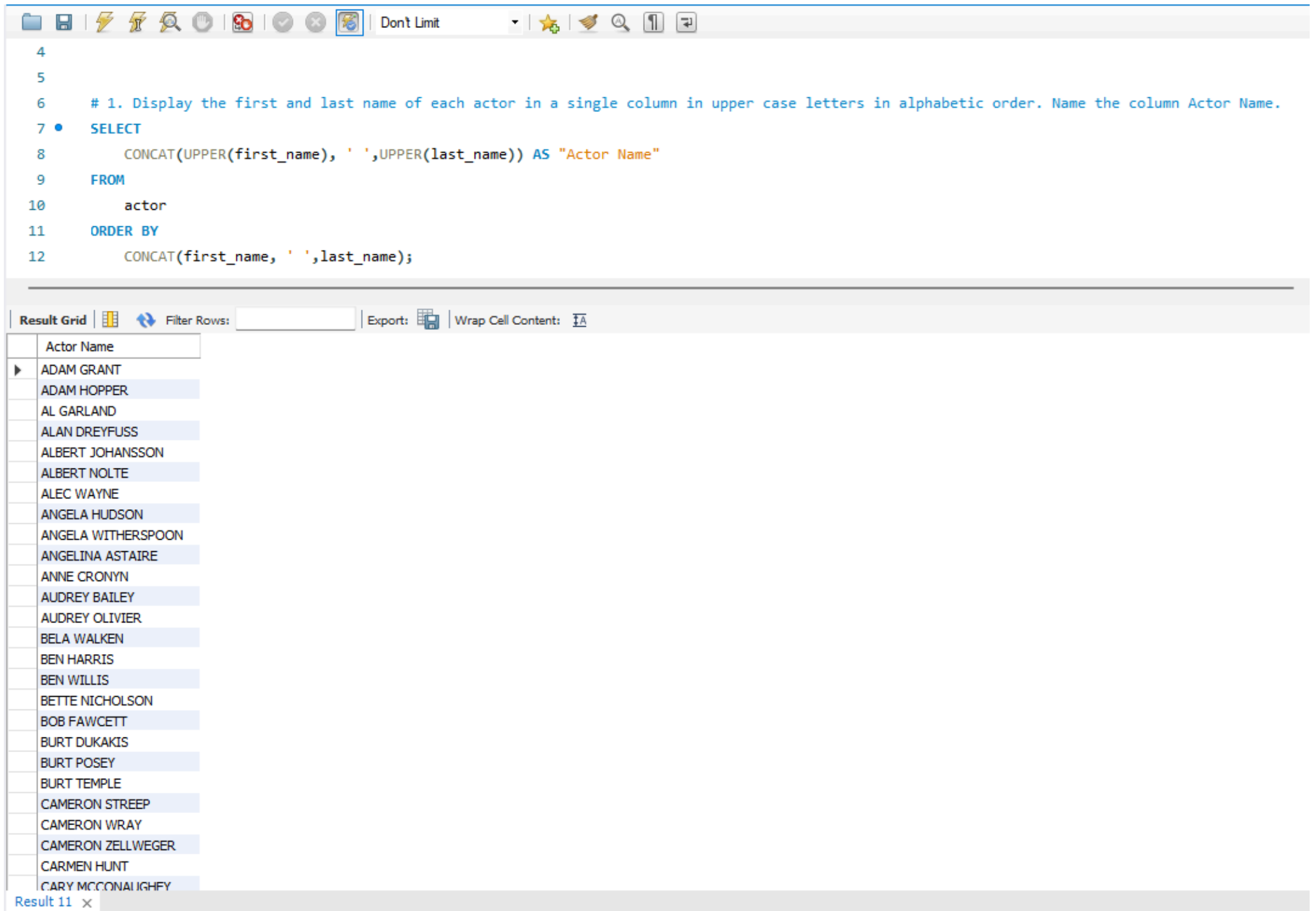


1. Display the first and last name of each actor in a single column in uppercase letters in alphabetic order. Name the column Actor Name.



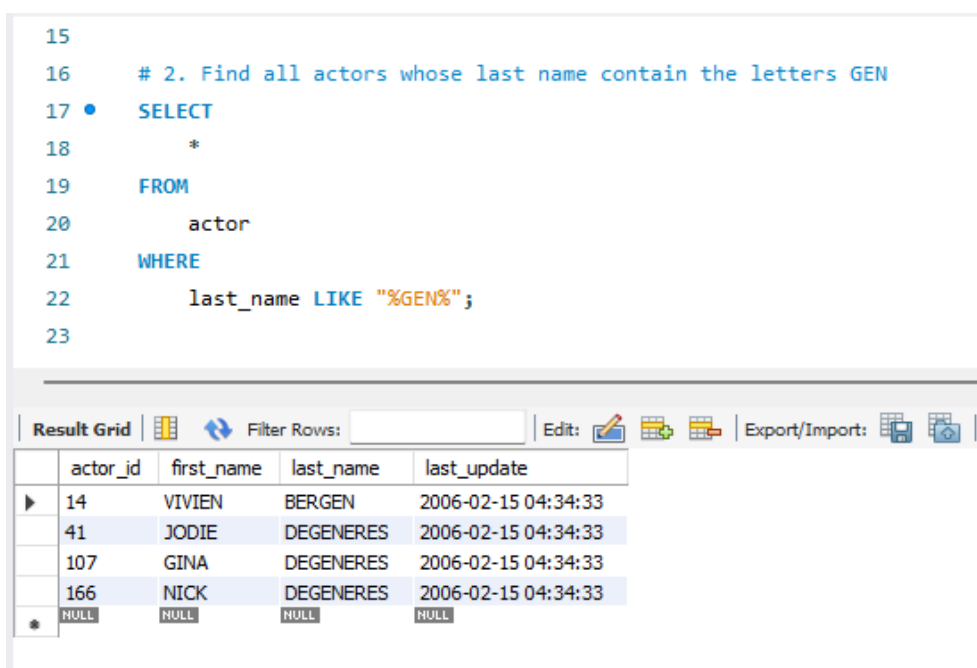
The screenshot shows a MySQL IDE window with a SQL query editor and a result grid. The query is as follows:

```
4  
5  
6 # 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.  
7 • SELECT  
8     CONCAT(UPPER(first_name), ' ',UPPER(last_name)) AS "Actor Name"  
9 FROM  
10    actor  
11 ORDER BY  
12    CONCAT(first_name, ' ',last_name);
```

The result grid displays the following data:

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 MCCONAIGHFY

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



The screenshot shows a MySQL IDE window with a SQL query editor and a result grid. The query is as follows:

```
15  
16 # 2. Find all actors whose last name contain the letters GEN  
17 • SELECT  
18     *  
19 FROM  
20    actor  
21 WHERE  
22    last_name LIKE "%GEN%";  
23
```

The result grid displays the following data:

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
*	NULL	NULL	NULL

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

```
24
25 # 3. Using IN, display the country_id and country columns of the following countries: Afghanistan, Bangladesh, and China:
26 SELECT
27     *
28 FROM
29     country
30 WHERE
31     country IN ("Afghanistan","Bangladesh","China");
32
```

Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
country_id	country	last_update		
1	Afghanistan	2006-02-15 04:44:00		
12	Bangladesh	2006-02-15 04:44:00		
23	China	2006-02-15 04:44:00		
NULL	NULL	NULL		

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

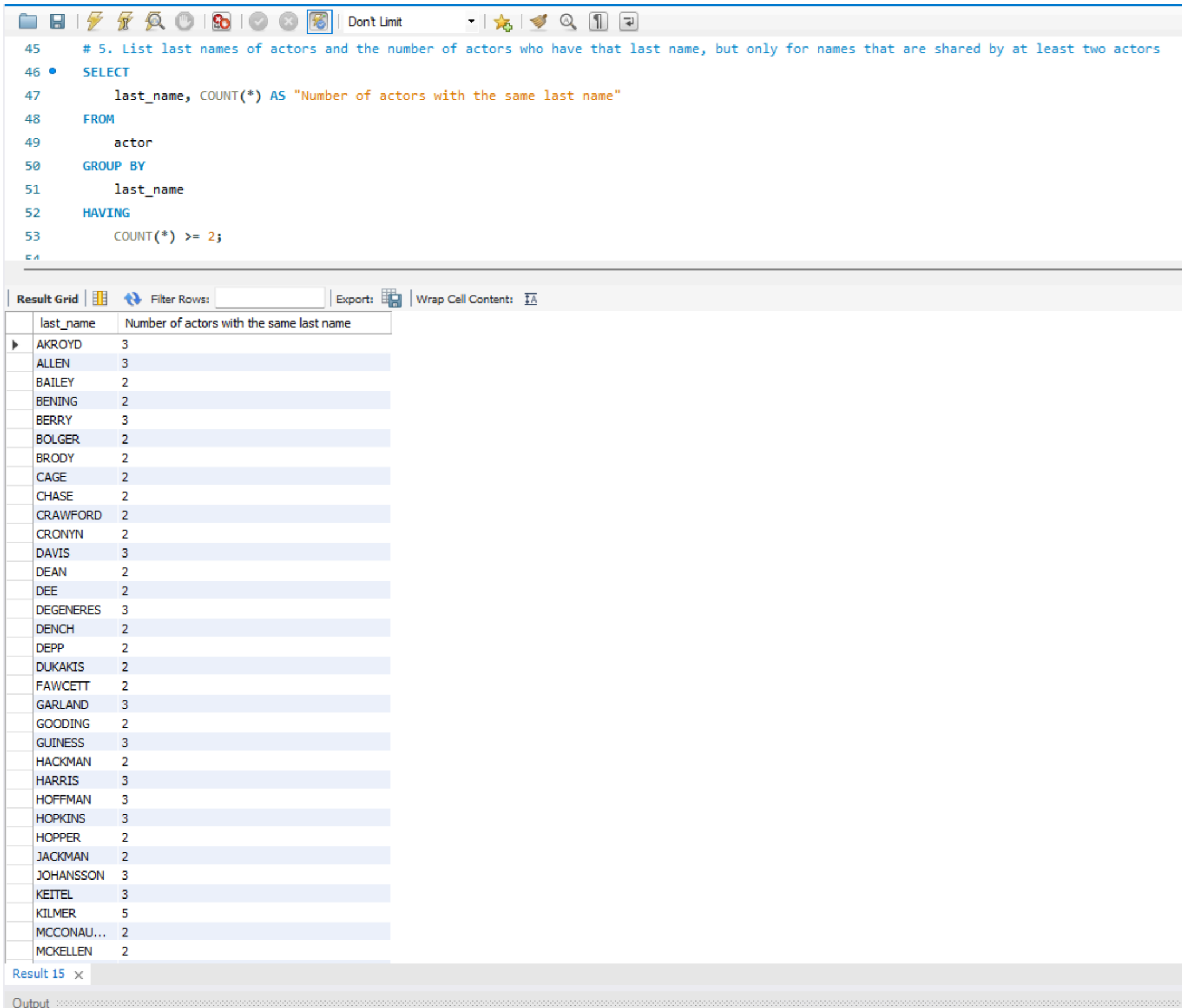
```
34
35 # 4. List the last names of actors, as well as how many actors have that last name.
36 • SELECT
37     last_name, COUNT(*) AS "Number of actors with the same last name"
38 FROM
39     actor
40 GROUP BY
41     last_name;
42
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
last_name	Number of actors with the same last name		
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		
DAY-LEWIS	1		
DEAN	2		

Result 14 ×

Output

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



The screenshot shows a MySQL IDE interface. The top toolbar includes icons for file operations, execution, and navigation. The SQL editor contains the following query:

```
45 # 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
46 • SELECT
47     last_name, COUNT(*) AS "Number of actors with the same last name"
48 FROM
49     actor
50 GROUP BY
51     last_name
52 HAVING
53     COUNT(*) >= 2;
```

Below the editor, the 'Result Grid' tab is active, displaying the query results in a table. The table has two columns: 'last_name' and 'Number of actors with the same last name'. The results are as follows:

last_name	Number of actors with the same last name
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
GUINNESS	3
HACKMAN	2
HARRIS	3
HOFFMAN	3
HOPKINS	3
HOPPER	2
JACKMAN	2
JOHANSSON	3
KETTEL	3
KILMER	5
MCCONAU...	2
MCKELLEN	2

At the bottom, there is a 'Result 15' tab and an 'Output' section.

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

Before Update:

170	MENA	HOPPER	2006-02-15 04:34:33
171	OLYMPIA	PFEIFFER	2006-02-15 04:34:33
172	GROUCHO	WILLIAMS	2024-08-08 13:21:31
173	ALAN	DREYFUSS	2006-02-15 04:34:33
174	MICHAEL	BENING	2006-02-15 04:34:33
175	WILLIAM	HACKMAN	2006-02-15 04:34:33

```

56
57 # 6. The actor HARPO WILLIAMS was accidentally entered in the actor table as GROUCHO WILLIAMS. Write a query to fix the record.
58 • UPDATE
59     actor
60 SET
61     first_name = "HARPO"
62 WHERE
63     first_name = "GROUCHO" AND last_name = "WILLIAMS";
64
65

```

Output			
Action Output			
#	Time	Action	Message
✓ 1	13:28:30	UPDATE actor SET first_name = "HARPO" WHERE first_name = "GROUCHO" AND last_name = "WILLIAMS"	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

After Update:

169	KENNETH	HOFFMAN	2006-02-15 04:34:33
170	MENA	HOPPER	2006-02-15 04:34:33
171	OLYMPIA	PFEIFFER	2006-02-15 04:34:33
172	HARPO	WILLIAMS	2024-08-08 13:28:30
173	ALAN	DREYFUSS	2006-02-15 04:34:33
174	MICHAEL	BENING	2006-02-15 04:34:33
175	WILLIAM	HACKMAN	2006-02-15 04:34:33
176	TOM	CHASE	2006-02-15 04:34:33

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:

SCHEMAS

Filter objects

sakila

Tables

actor

address

Columns

address_id

address

address2

district

city_id

postal_code

phone

location

last_update

Indexes

Foreign Keys

Triggers

category

city

country

customer

film

film_actor

film_category

film_text

inventory

language

payment

rental

staff

Columns

staff_id

first_name

Administration

Schemas

Information

66

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

68 • SELECT

69 s.first_name,

70 s.last_name,

71 a.address

72 FROM

73 staff s

74 JOIN

75 address a ON s.address_id = a.address_id ;

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

first_name	last_name	address
Mike	Hillyer	23 Workhaven Lane
Jon	Stephens	1411 Lillydale Drive

Result 16 x

Output

Action Output

#	Time	Action	Message
✓ 1	13:28:30	UPDATE actor SET first_name = "HARPO" WHERE first_name = "GROUCHO" AND last_name = "WILLIAMS"	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
✓ 2	14:12:08	SELECT s.first_name, s.last_name, a.address FROM staff s JOIN address a ON s.address_id = a.address_id	2 row(s) returned

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

```

78
79 # 8. List each film and the number of actors who are listed for that film. Use tables film_actor and film. Use inner join.
80 • SELECT
81     f.title AS "Film Title",
82     COUNT(*) AS "Number of actors listed"
83 FROM
84     film f
85 INNER JOIN
86     film_actor fa ON f.film_id = fa.film_id
87 GROUP BY
88     f.title;
89

```

Result Grid		
Filter Rows: <input type="text"/>		
Export:		
Wrap Cell Content:		
	Film Title	Number of actors listed
▶	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
	ALLEY EVOLUTION	5
	ALONE TRIP	8
	ALTER VICTORY	4
	AMADEUS HOLY	6
	AMELIE HELLFIGHTERS	6
	AMERICAN CIRCUS	5
	AMISTAD MIDSUMMER	4
	ANACONDA CONFES...	5

Result 19 x

Output

Action Output

#	Time	Action	Message
✓ 1	13:28:30	UPDATE actor SET first_name = "HARPO" WHERE first_name = "GROUCHO" AND last_name = "WILLIAMS"	1 row(s) affected Rows ma
✓ 2	14:12:08	SELECT s.first_name, s.last_name, a.address FROM staff s JOIN address a ON s.address_id = a.address_id	2 row(s) returned
✓ 3	14:30:51	SELECT f.title AS film_title, COUNT(*) AS actor_count FROM film f INNER JOIN film_actor fa ON f.film_id = fa.film_id GROUP BY f.title OR...	997 row(s) returned
✓ 4	14:32:37	SELECT f.title AS "Film Title", COUNT(*) AS "Number of actors listed" FROM film f INNER JOIN film_actor fa ON f.film_id = fa.film_id GROUP...	997 row(s) returned
✓ 5	14:55:13	SELECT f.title AS "Film Title", COUNT(*) AS "Number of actors listed" FROM film f INNER JOIN film_actor fa ON f.film_id = fa.film_id GROUP...	997 row(s) returned

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

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the 'sakila' database structure, including tables like 'country', 'customer', 'film', 'inventory', and 'payment'. The 'film' table is selected, and its columns are visible. The main editor shows the following SQL query:

```

91
92 # 9. How many copies of the film Hunchback Impossible exist in the inventory system?
93 • SELECT
94     COUNT(*) AS "Number of Hunchback Impossible copies in the inventory"
95 FROM
96     inventory i
97 JOIN
98     film f ON i.film_id = f.film_id
99 WHERE
100    f.title="HUNCHBACK IMPOSSIBLE";
101
102

```

The 'Result Grid' at the bottom shows the query result:

Number of Hunchback Impossible copies in the inventory
6

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

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the 'sakila' database structure. The 'payment' table is selected, and its columns are visible. The main editor shows the following SQL query:

```

101
102
103
104 # 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
105 • SELECT
106     c.first_name,
107     c.last_name,
108     SUM(p.amount) AS "Total paid by each customer"
109 FROM
110     customer c
111 JOIN
112     payment p ON c.customer_id = p.customer_id
113 GROUP BY
114     c.first_name, c.last_name ;
115
116

```

The 'Result Grid' at the bottom shows the query result, listing customers alphabetically by last name and their total payment:

first_name	last_name	Total paid by each customer
MARY	SMITH	118.68
PATRICIA	JOHNSON	128.73
LINDA	WILLIAMS	135.74
BARBARA	JONES	81.78
ELIZABETH	BROWN	144.62
JENNIFER	DAVIS	93.72
MARIA	MILLER	151.67
SUSAN	WILSON	92.76
MARGARET	MOORE	89.77
DOROTHY	TAYLOR	99.75
LISA	ANDERSON	106.76
NANCY	THOMAS	103.72
KAREN	JACKSON	131.73
BETTY	WHITE	117.72
HELEN	HARRIS	134.68
SANDRA	MARTIN	118.72
DONNA	THOMPSON	98.79
CAROL	GARCIA	91.78
RUTH	MARTINEZ	125.76
SHARON	ROBINSON	115.70

The 'Output' pane at the bottom shows the execution log, indicating that the query was executed successfully and returned 599 rows.

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.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the 'film' table structure, with 'language_id' highlighted. Below it, the 'Administration' tab shows the 'Schemas' section, and the 'Information' tab shows the 'Column: language_id' definition: 'language_id tinyint UN'. The main editor displays a SQL query:

```
117
118
119 # 11. The music of Queen and Kris Kristofferson have seen an unlikely resurgence. As an unintended consequence,
120 # films starting with the letters K and Q have also soared in popularity.
121 # Use subqueries to display the titles of movies starting with the letters K and Q whose language is English.
122 • SELECT
123     f.title
124 FROM
125     film f
126 WHERE
127     (f.title LIKE "K%" OR f.title LIKE "Q%") AND f.language_id = ( SELECT l.language_id FROM language l WHERE name = "English");
128
129
130
131
132
```

The 'Result Grid' at the bottom shows the following titles:

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

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

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the 'film_actor' table structure, with 'actor_id' highlighted. Below it, the 'Administration' tab shows the 'Schemas' section, and the 'Information' tab shows the 'Column: actor_id' definition: 'actor_id tinyint UN'. The main editor displays a SQL query:

```
128
129
130
131
132
133 # 12. Use subqueries to display all actors who appear in the film Alone Trip.
134 • SELECT
135     CONCAT(first_name, ' ',last_name) AS "Actors in the film Alone Trip."
136 FROM
137     actor
138 WHERE
139     actor_id IN (SELECT actor_id FROM film_actor WHERE film_id = ( SELECT film_id FROM film WHERE title = "ALONE TRIP"));
140
141
142
```

The 'Result Grid' at the bottom shows the following actors:

Actors in the film Alone Trip.
ED CHASE
KARL BERRY
UMA WOOD
WOODY JOLIE
SPENCER DEPP
CHRIS DEPP
LAURENCE BULLOCK
RENEE BALL

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.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the database structure, including tables like 'address', 'city', and 'customer'. The 'customer' table is selected, showing its columns: 'country_id', 'country', and 'last_update'. Below this, the 'Definition' section shows the 'country_id' column as a small integer, unsigned, auto-incremented primary key.

The main editor displays a SQL query (lines 138-158) that retrieves the first name, last name, and email of all Canadian customers using joins:

```
138
139
140
141  # 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.
142  # Use joins to retrieve this information.
143  • SELECT
144      c.first_name,
145      c.last_name,
146      c.email
147  FROM
148      customer c
149  JOIN
150      address a ON c.address_id = a.address_id
151  JOIN
152      city ci ON a.city_id = ci.city_id
153  JOIN
154      country co ON ci.country_id = co.country_id
155  WHERE
156      co.country = "Canada";
157
158
```

Below the query, the 'Result Grid' shows 5 rows of data:

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

At the bottom, the 'Output' pane shows the execution details for 'Result 3':

#	Time	Action	Message
3	21:37:58	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...	5 row(s) returned

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

SCHEMAS
Don't Limit

Filter objects

- ▼ sakila
 - Tables
 - actor
 - address
 - category
 - Columns
 - category_id
 - name
 - last_update
 - Indexes
 - Foreign Keys
 - Triggers
 - city
 - country
 - customer
 - film
 - Columns
 - film_id
 - title
 - description
 - release_year
 - language_id
 - original_language
 - rental_duration
 - rental_rate
 - length
 - replacement_cost
 - rating
 - special_features
 - last_update
 - Indexes
 - Foreign Keys

```

160 # 14. Sales have been lagging among young families, and you wish to target all family movies for a promotion. Identify all movies categorized as family films.
161 • SELECT
162     f.title AS "Family Category Films"
163 FROM
164     film f
165 JOIN
166     film_category fc ON f.film_id = fc.film_id
167 JOIN
168     category c ON fc.category_id = c.category_id
169 WHERE
170     c.name = "Family";

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Family Category Films
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
FELD FROGMEN
FINDING ANACONDA
GABLES METROPOLIS
GANDHI KWAI
GLADIATOR WESTWARD
GREASE YOUTH
HALF OUTFIELD
HOCUS FRIDA
HOMICIDE PEACH
HOUSE DYNAMITE
HUNTING MUSKETEERS
INDIAN LOVE
INDIAN LOVE

Column: **name**

Collation: utf8mb4_0900_ai_ci

Definition:

```
name varchar(25)
```

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

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'sakila' database structure, including tables like 'film_category' and 'film'. The main editor shows the following SQL code:

```
172
173
174 # 15. Create a Stored procedure to get the count of films in the input category (IN category_name, OUT count)
175 DELIMITER //
176 • CREATE PROCEDURE FilmCountBy(IN category_name VARCHAR(50), OUT count INT)
177 BEGIN
178     SELECT
179         COUNT(f.film_id)
180     INTO
181         count
182     FROM
183         film f
184     JOIN
185         film_category fc ON f.film_id = fc.film_id # Join film_category table as fc with film table based on the film_id.
186     JOIN
187         category c ON fc.category_id = c.category_id # Join category table as c with film_category table based on category_id
188     WHERE
189         c.name = category_name;
190 END //
191 DELIMITER ;
192
193 • CALL FilmCountBy("Action",@count); # call procedure
194 • SELECT @count AS "Action Films Count"; # show result
195
196
```

Below the SQL editor, the 'Result Grid' shows the output of the execution:

Action Films Count
64

At the bottom left, a message states: 'Unable to retrieve node description.'

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

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'sakila' database with its tables and columns. The 'Columns' pane for the 'film' table is visible, showing columns like 'film_id', 'title', 'description', etc. The main editor shows a SQL query to find the most frequently rented movies. The 'Result Grid' pane shows the results of the query, listing movie titles and their rental frequencies in descending order.

SQL Query:

```
# 16. Display the most frequently rented movies in descending order.
SELECT
  f.title AS "The most frequently rented movies",
  COUNT(r.rental_id) AS frequency
FROM
  rental r
JOIN
  inventory i ON r.inventory_id = i.inventory_id
JOIN
  film f ON i.film_id = f.film_id
GROUP BY
  f.title
ORDER BY
  frequency DESC;
```

Result Grid:

The most frequently rented movies	frequency
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
CAT CONEHEADS	30
DOGMA FAMILY	30
ENGLISH BULWORTH	30
FROST HEAD	30
GRAFFITI LOVE	30
HARRY IDAHO	30
IDOLS SNATCHERS	30
MARRIED GO	30
MASSACRE USUAL	30
MUSCLE BRIGHT	30
PULP BEVERLY	30
PIGGY BACK SHAKESPEARE	30

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

The screenshot shows the MySQL Workbench interface. On the left is the 'Schemas' pane with a tree view of the 'sakila' database. The 'country' table is selected, showing its columns: country_id, country, and last_update. The main editor displays a SQL query to retrieve store information. Below the query editor is the 'Result Grid' showing two rows of data.

```
214
215 # 17. Write a query to display for each store its store ID, city, and country.
216 • SELECT
217     s.store_id AS "Store ID",
218     ci.city AS "City",
219     co.country AS "Country"
220 FROM
221     store s
222 JOIN
223     address a ON s.address_id = a.address_id
224 JOIN
225     city ci ON a.city_id = ci.city_id
226 JOIN
227     country co ON ci.country_id = co.country_id;
228
```

	Store ID	City	Country
▶	1	Lethbridge	Canada
	2	Woodridge	Australia

18. List the genres and its gross revenue.

SCHEMAS

Filter objects

sakila

Tables

actor

address

category

Columns

category_id

name

last_update

Indexes

Foreign Keys

Triggers

city

country

customer

film

film_actor

film_category

Columns

film_id

category_id

last_update

Indexes

Foreign Keys

Triggers

film_text

inventory

Columns

inventory_id

film_id

store_id

last_update

Indexes

Administration

Schemas

Information

Column: name

Collation: utf8mb4_0900_ai_ci

Definition:

name varchar(25)

231

18. List the genres and its gross revenue.

232

• SELECT

233

c.name AS "Genres",

234

SUM(pa.amount) AS "Gross revenue"

235

FROM

236

category c

237

JOIN

238

film_category fc ON c.category_id = fc.category_id

239

JOIN

240

inventory i ON fc.film_id = i.film_id

241

JOIN

242

rental r ON i.inventory_id = r.inventory_id

243

JOIN

244

payment pa ON r.rental_id = pa.rental_id

245

GROUP BY

246

c.name;

247

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

Genres	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

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

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the database structure, with the 'genres_gross_revenue' view selected under the 'Views' folder. The main editor shows the SQL script for creating the view and querying it. The 'Result Grid' at the bottom right displays the output of the query, showing the gross revenue for each genre.

```
249
250 # 19. Create a View for the above query
251 • CREATE VIEW genres_gross_revenue AS SELECT c.name AS "Genres", SUM(pa.amount) AS "Gross revenue"
252 FROM
253     category c
254 JOIN
255     film_category fc ON c.category_id = fc.category_id
256 JOIN
257     inventory i ON fc.film_id = i.film_id
258 JOIN
259     rental r ON i.inventory_id = r.inventory_id
260 JOIN
261     payment pa ON r.rental_id = pa.rental_id
262 GROUP BY
263     c.name;
264
265 • SELECT * FROM genres_gross_revenue; # Show view genres_gross_revenue
266
267
```

Result Grid

Genres	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

View: genres_gross_revenue

Columns:

Genres	varchar(25)
Gross revenue	decimal(27,2)

20. Select top 5 genres in gross revenue view.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays a tree view of the database structure, including tables like 'rental', 'staff', 'store', and views like 'genres_gross_revenue'. The 'genres_gross_revenue' view is selected, showing its columns: 'Genres' and 'Gross revenue'.

The main editor displays the following SQL query:

```
# 20. Select top 5 genres in gross revenue view.
SELECT
  *
FROM
  genres_gross_revenue
ORDER BY
  `Gross revenue` DESC
LIMIT 5;
```

Below the query editor, the 'Result Grid' shows the results of the query. The table has two columns: 'Genres' and 'Gross revenue'. The results are as follows:

Genres	Gross revenue
Sports	5314.21
Sci-Fi	4756.98
Animation	4656.30
Drama	4587.39
Comedy	4383.58

At the bottom, the 'Information' pane shows the definition of the 'Gross revenue' column:

Column: Gross revenue
Definition: decimal(27,2)