

TABLE SAKILA

1a. Display the first and last names of all actors from the table actor.

```
1 select first_name, last_name
2 from actor;
```

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

```
1 select upper(concat(first_name, ' ', last_name)) as Actor_Name
2 from actor;
```

2a. You need to find the ID number, first name, and last name of an actor, of whom you know only the first name, "Joe." What is one query would you use to obtain this information?

```
1 • SELECT actor_id, first_name, last_name
2 FROM actor
3 WHERE lower(first_name)=lower('Joe');
```

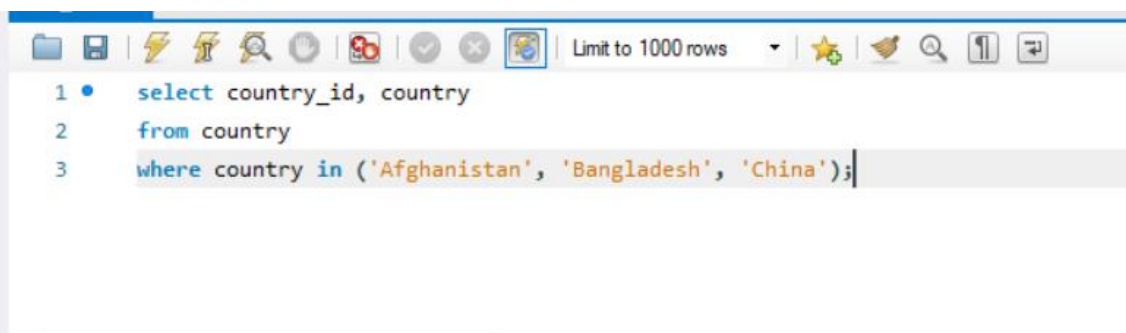
2b. Find all actors whose last name contain the letters GEN:

```
1 • SELECT actor_id, first_name, last_name
2 FROM actor
3 WHERE upper(last_name) LIKE '%GEN%';
```

2c. Find all actors whose last names contain the letters LI. This time, order the rows by last name and first name, in that order:

```
1 • SELECT actor_id,first_name,last_name
2 FROM actor
3 WHERE upper(last_name) LIKE '%LI%'
4 ORDER BY last_name,first_name;
```

2d. Using IN, display the country_id and country columns of the following countries: Afghanistan, Bangladesh, and China:



```
1 • select country_id, country
2 from country
3 where country in ('Afghanistan', 'Bangladesh', 'China');
```

3a. You want to keep a description of each actor. You don't think you will be performing queries on a description, so create a column in the table actor named description and use the data type BLOB (Make sure to research the type BLOB, as the difference between it and VARCHAR are significant).



```
1 • ALTER TABLE sakila.actor
2 ADD description BLOB;
3
4 • select *
5 from actor;
```

3b. Very quickly you realize that entering descriptions for each actor is too much effort. Delete the description column.

```
1 • ALTER TABLE sakila.actor
2 DROP description ;
3
```

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

```
1 • SELECT last_name,count(last_name) as count
2 from actor
3 GROUP BY last_name
4 ORDER BY count DESC,last_name
5
```

4b. 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

```
1 • SELECT last_name,count(*) as count_last_name
2 from actor
3 GROUP BY last_name
4 HAVING count_last_name>1
5 ORDER BY count_last_name DESC,last_name
6
```

4c. The actor HARPO WILLIAMS was accidentally entered in the actor table as GROUCHO WILLIAMS. Write a query to fix the record.

```

1 • SELECT *
2   from actor
3   where upper(first_name) = 'GROUCHO' AND upper(last_name)='WILLIAMS';
4
5 • UPDATE actor
6   SET first_name = 'HARPO', last_name='WILLIAMS'
7   WHERE upper(first_name) = 'GROUCHO' AND upper(last_name)='WILLIAMS';
8
9 • SELECT *
10  from actor
11  where upper(first_name) = 'HARPO' AND upper(last_name)='WILLIAMS';

```

4d. Perhaps we were too hasty in changing GROUCHO to HARPO. It turns out that GROUCHO was the correct name after all! In a single query, if the first name of the actor is currently HARPO, change it to GROUCHO.

```

1 • UPDATE actor
2   SET first_name = 'GROUCHO', last_name='WILLIAMS'
3   WHERE upper(first_name) = 'HARPO' AND upper(last_name)='WILLIAMS';
4
5 • SELECT *
6   FROM actor
7   WHERE upper(first_name)='GROUCHO' AND upper(last_name)='WILLIAMS';

```

5a. You cannot locate the schema of the address table. Which query would you use to re-create it?



The screenshot shows a database client interface. In the SQL editor, the command 'SHOW CREATE TABLE address;' is entered. Below the editor, there are tabs for 'Result Grid', 'Table', and 'Create Table'.

```

1 CREATE TABLE `address` (
2   `address_id` smallint unsigned NOT NULL AUTO_INCREMENT,
3   `address` varchar(50) NOT NULL,
4   `address2` varchar(50) DEFAULT NULL,
5   `district` varchar(20) NOT NULL,
6   `city_id` smallint unsigned NOT NULL,
7   `postal_code` varchar(10) DEFAULT NULL,
8   `phone` varchar(20) NOT NULL,
9   `location` geometry NOT NULL,
10  `last_update` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
11  PRIMARY KEY (`address_id`),
12  KEY `idx_fk_city_id` (`city_id`),
13  SPATIAL KEY `idx_location` (`location`),
14  CONSTRAINT `fk_address_city` FOREIGN KEY (`city_id`) REFERENCES `city` (`city_id`) ON DELETE RESTRICT
    ON UPDATE CASCADE
15 ) ENGINE=InnoDB AUTO_INCREMENT=606 DEFAULT CHARSET=utf8

```

6a. Use JOIN to display the first and last names, as well as the address, of each staff member. Use the tables staff and address:

```

1 • SELECT stf.first_name,stf.last_name,ads.district,ads.location,ads.address
2 FROM staff stf
3 LEFT JOIN address ads
4 ON stf.address_id=ads.address_id
5

```

6b. Use JOIN to display the total amount rung up by each staff member in August of 2005. Use tables staff and payment.

```

SELECT std.staff_id,std.first_name,std.last_name,SUM(pay.amount) as total_amount,pay.payment_date
FROM staff std
LEFT JOIN payment pay
ON std.staff_id=pay.staff_id
WHERE month(pay.payment_date)=8 AND year(pay.payment_date)=2005
GROUP BY std.staff_id
ORDER BY total_amount DESC

```

6c. List each film and the number of actors who are listed for that film. Use tables film_actor and film. Use inner join.

```

1 • SELECT film.film_id,film.title,count(fact.actor_id) as num_actor
2 FROM film_actor fact
3 INNER JOIN film
4 ON fact.film_id=film.film_id
5 GROUP BY film.film_id
6 ORDER BY num_actor DESC,film.title desc

```

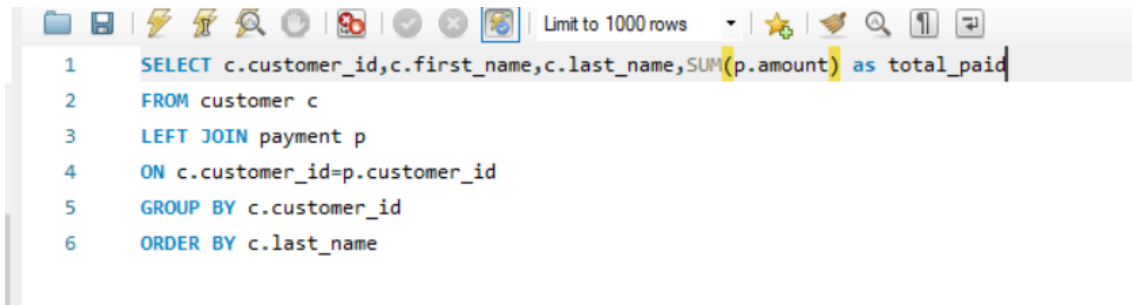
6d. How many copies of the film Hunchback Impossible exist in the inventory system?

```

1 • SELECT *
2 FROM film
3 WHERE title='Hunchback Impossible';
4
5 • SELECT film.title,count(*) as Num_copies
6 FROM inventory
7 JOIN film
8 ON inventory.film_id=film.film_id
9 WHERE upper(film.title)='Hunchback Impossible'

```

6e. Using the tables payment and customer and the JOIN command, list the total paid by each customer. List the customers alphabetically by last name:

A screenshot of a SQL query editor interface. The top toolbar includes icons for file operations, a search icon, and a 'Limit to 1000 rows' dropdown. The query text is as follows:

```
1 SELECT c.customer_id,c.first_name,c.last_name,SUM(p.amount) as total_paid
2 FROM customer c
3 LEFT JOIN payment p
4 ON c.customer_id=p.customer_id
5 GROUP BY c.customer_id
6 ORDER BY c.last_name
```

7a. 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.

A screenshot of a SQL query editor showing a query to find movie titles starting with 'K' or 'Q' in English. The query text is as follows:

```
SELECT title
FROM film
WHERE (title LIKE 'Q%' or title LIKE 'K%')
AND language_id in(SELECT language_id FROM language WHERE upper(name)='ENGLISH')
```

7b. Use subqueries to display all actors who appear in the film Alone Trip.

```

1 • SELECT first_name,last_name
2   FROM actor
3   WHERE actor_id in
4   (SELECT actor_id
5    FROM film_actor
6    JOIN film
7    ON film_actor.film_id=film.film_id
8    WHERE lower(film.title)=lower("Alone Trip"));
9
10 • SELECT concat(first_name,' ',last_name) as Actor_Name
11   FROM actor
12   WHERE actor_id IN
13   (SELECT actor_id
14    FROM film_actor
15    WHERE film_id IN
16    (SELECT film_id
17     FROM film
18     WHERE lower(title)=lower("Alone Trip")))

```

7c. 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.

```

• SELECT concat(first_name,' ',last_name) as Customer_Name,email
  FROM customer
  WHERE address_id IN
  (SELECT address_id
   FROM address
   WHERE city_id IN
   (SELECT city_id
    FROM city
    WHERE country_id IN
    (SELECT country_id
     FROM country
     WHERE lower(country)=lower('CANADA'))))

• SELECT concat(c.first_name,' ',c.last_name) as Customer_Name,c.email
  FROM customer c
  JOIN address ad ON ad.address_id=c.address_id
  JOIN city ci ON ad.city_id= ci.city_id
  JOIN country co ON ci.country_id=co.country_id
  WHERE lower(co.country)=lower('Canada')

```

7d. 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.

```
SELECT f.title as TITLE,f.language_id as LANGUAGE,c.name as CATEGORY,f.description as DESCRIPTION
FROM film f
JOIN film_category fc
ON fc.film_id=f.film_id
JOIN category c
ON c.category_id=fc.category_id
WHERE lower(c.name)=lower('family')
```

7e. Display the most frequently rented movies in descending order.

```
• SELECT f.film_id,f.title,COUNT(*) as times_rented
FROM film f
JOIN inventory i ON f.film_id=i.film_id
JOIN rental r ON r.inventory_id=i.inventory_id
GROUP BY f.title
ORDER BY times_rented DESC
```

7f. Write a query to display how much business, in dollars, each store brought in.

```
SELECT s.store_id,SUM(p.amount) as Total_amount
FROM store s
JOIN customer c ON c.store_id=s.store_id
JOIN payment p ON p.customer_id=c.customer_id
GROUP BY s.store_id
;
```


7g. Write a query to display for each store its store ID, city, and country.

```
SELECT s.store_id,SUM(p.amount) as Total_amount,ci.city,coun.country
FROM store s
JOIN customer c ON c.store_id=s.store_id
JOIN payment p ON p.customer_id=c.customer_id
JOIN address a ON a.address_id=s.address_id
JOIN city ci ON ci.city_id=a.city_id
JOIN country coun ON coun.country_id=ci.country_id
GROUP BY s.store_id
;
```

7h. List the top five genres in gross revenue in descending order. (Hint: you may need to use the following tables: category, film_category, inventory, payment, and rental.)

```
1 • SELECT c.name,SUM(p.amount) as total_amount
2 FROM category c
3 JOIN film_category fc ON fc.category_id=c.category_id
4 JOIN film f ON f.film_id=fc.film_id
5 JOIN inventory i ON i.film_id=f.film_id
6 JOIN rental r ON i.inventory_id=r.inventory_id
7 JOIN payment p ON p.rental_id=r.rental_id
8 GROUP BY c.name
9 ORDER BY total_amount desc
10 LIMIT 5
```

8a. In your new role as an executive, you would like to have an easy way of viewing the Top five genres by gross revenue. Use the solution from the problem above to create a view. If you haven't solved 7h, you can substitute another query to create a view.

- ```
create view top_five_genres as
SELECT c.name,SUM(p.amount) as total_amount
FROM category c
JOIN film_category fc ON fc.category_id=c.category_id
JOIN film f ON f.film_id=fc.film_id
JOIN inventory i ON i.film_id=f.film_id
JOIN rental r ON i.inventory_id=r.inventory_id
JOIN payment p ON p.rental_id=r.rental_id
GROUP BY c.name
ORDER BY total_amount desc
LIMIT 5;
```
- ```
select * from top_five_genres;
```

8b. How would you display the view that you created in 8a?

```
select * from top_five_genres;
```

8c. You find that you no longer need the view top_five_genres. Write a query to delete it.

```
drop view top_five_genres
```