To submit you can make

1. A google doc and click share and share public - then submit that google doc
2. Better option: make a github repo, put the sql files for pagilla in it. Make a README.md file and put all the queries and questions in there formated nicely
   1. and give instructions on how to populate the database.
      1. To create this database
         1. Open up psql in the directory of these files and run these commands
         2. \i …
         3. \i …
         4. \i ...

* 1a. You need a list of all the actors’ first name and last name

SELECT first\_name, last\_name

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

SELECT CONCAT (UPPER(first\_name), ' ', UPPER(last\_name))

AS Actor\_Name

FROM actor;

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

SELECT actor\_id, first\_name, last\_name

FROM actor

WHERE first\_name='JOE';

* 2b. Find all actors whose last name contain the letters GEN. Make this case insensitive

SELECT \*

FROM actor

WHERE last\_name iLIKE '%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. Make this case insensitive.

SELECT \*

FROM actor

WHERE last\_name iLIKE '%LI%'

ORDER BY last\_name ASC, first\_name ASC;

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

SELECT country\_id, country

FROM country

WHERE country IN ('Afghanistan', 'Bangladesh', 'China');

* 3a. Add a middle\_name column to the table actor. Specify the appropriate column type

ALTER TABLE actor

ADD middle\_name varchar(45);

* 3b. You realize that some of these actors have tremendously long last names. Change the data type of the middle\_name column to something that can hold more than varchar.
* 3c. Now write a query that would remove the middle\_name column.

ALTER TABLE actor

DROP COLUMN middle\_name;

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

SELECT last\_name, count(last\_name)

FROM actor

GROUP BY last\_name;

* 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

SELECT last\_name, count(\*)

FROM actor

GROUP BY last\_name

HAVING count(\*) > 1;

* 4c. Oh, no! The actor HARPO WILLIAMS was accidentally entered in the actor table as GROUCHO WILLIAMS. Write a query to fix the record.

SELECT REPLACE(first\_name,'GROUCHO','HARPO')

FROM actor

WHERE first\_name='GROUCHO' AND 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.

Otherwise, change the first name to MUCHO GROUCHO, as that is exactly what the actor will be with the grievous error.

BE CAREFUL NOT TO CHANGE THE FIRST NAME OF EVERY ACTOR TO MUCHO GROUCHO

(Hint: update the record using a unique identifier.)

* 5a.
  + What’s the difference between a left join and a right join?

Left join returns all the rows from the left table, even if there are no matches from the right table.

Right join returns all the rows from the right table, even if there are no matches from the left table.

* + What about an inner join and an outer join?

Inner join returns the rows where there’s a match in both tables.

Outer join returns all rows from both tables, whether there is a match or not.

* + When would you use rank?

RANK gives ranking to your data based on your row, but it skipped rank if there are ties. It provides the same ranking to “ties”.

* + What about dense\_rank?

Similar to RANK, it ranks your data but in consecutive order. It doesn’t skipped rank.

* + When would you use a subquery in a select?

Subquery is used when you know how to search for a value using a SELECT statement, but don’t know the exact value in the database.

* + When would you use a right join?

A right join is similar to a left join except you are referencing the table on the “right”. You want to return all data from the right table, but only certain data from the left table that intersect the data on the right table.

* + When would you use an inner join over an outer join?

Use an inner join when you want to find and return matching data from both tables.

* + What’s the difference between a left outer and a left join?

There isn’t a difference really.

* + When would you use a group by?

Group by is used to arrange identical data into groups.

* + Describe how you would do data reformatting

You can use the FORMAT() function.

SELECT FORMAT (column\_name, format)

FROM table\_name;

* + When would you use a with clause?

The WITH clause is used to combine subqueries.

* + bonus: When would you use a self join?

When you want to compare values in a column with other values in the same column in the same table.

You’re answering each of these questions. Please include examples and answer them in plain english.

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

SELECT staff.first\_name, staff.last\_name, address.address

FROM staff

LEFT JOIN address ON staff.address\_id=address.address\_id;

* 6b. Use a JOIN to display the total amount rung up by each staff member in January of 2007. Use tables staff and payment.

SELECT staff.first\_name, staff.last\_name, payment.amount

FROM staff

LEFT JOIN payment ON staff.staff\_id=payment.staff\_id

WHERE payment\_date iLIKE '2007-01%';

You’ll have to google for this one, we didn’t cover it explicitly in class.

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

SELECT film.title, COUNT(film\_actor.actor\_id)

FROM film

INNER JOIN film\_actor ON film.film\_id=film\_actor.film\_id

GROUP BY film.title

ORDER BY film.title ASC;

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

SELECT film.title, count(inventory.store\_id)

FROM film

INNER JOIN inventory ON film.film\_id=inventory.film\_id

WHERE title iLIKE 'Hunchback Impossible'

GROUP BY film.title;

* 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:

SELECT customer.last\_name, customer.first\_name, SUM(payment.amount)

FROM customer

INNER JOIN payment ON customer.customer\_id=payment.customer\_id

GROUP BY customer.last\_name, customer.first\_name

ORDER BY customer.last\_name ASC;

* 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. display the titles of movies starting with the letters K and Q whose language is English.

SELECT film.title, film.language\_id, language.name

FROM film

INNER JOIN language ON film.language\_id=language.language\_id

WHERE title iLIKE 'K%' or title iLIKE 'Q%'

GROUP BY film.title, film.language\_id, language.name

HAVING film.language\_id='1';

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

SELECT sub.title, actor.first\_name, actor.last\_name

FROM (

SELECT film.title, film\_actor.actor\_id

FROM film

INNER JOIN film\_actor ON film.film\_id=film\_actor.film\_id

WHERE film.title iLIKE 'Alone Trip'

) sub

INNER JOIN actor ON sub.actor\_id=actor.actor\_id;

* 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

* 7d. Sales have been lagging among young families, and you wish to target all family movies for a promotion. Identify all movies categorized as a family film.

SELECT film.title

FROM film

INNER JOIN film\_category ON film\_category.film\_id=film.film\_id

WHERE category\_id='8';

Now we mentioned family film, but there is no family film category. There’s a category that resembles that. In the real world nothing will be exact.

* 7e. Display the most frequently rented movies in descending order.
* 7f. Write a query to display how much business, in dollars, each store brought in.

SELECT staff.store\_id, SUM(payment.amount)

FROM staff

INNER JOIN payment ON staff.staff\_id=payment.staff\_id

LEFT JOIN store ON store.store\_id=staff.store\_id

GROUP BY staff.store\_id;

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

SELECT store.store\_id, city.city, country.country

FROM store

INNER JOIN address ON store.address\_id=address.address\_id

LEFT JOIN city ON address.city\_id=city.city\_id

LEFT JOIN country ON city.country\_id=country.country\_id;

* 7h. List the top five genres in gross revenue in descending order.
* 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.
* 8b. How would you display the view that you created in 8a?
* 8c. You find that you no longer need the view top\_five\_genres. Write a query to delete it.