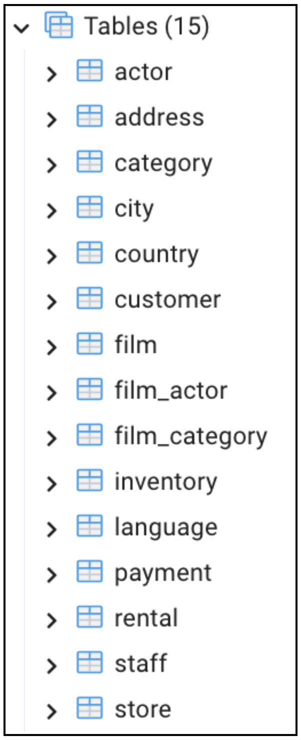
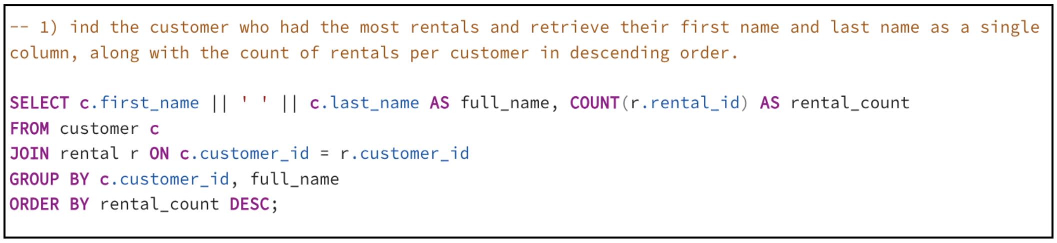
Lab Assignment 4

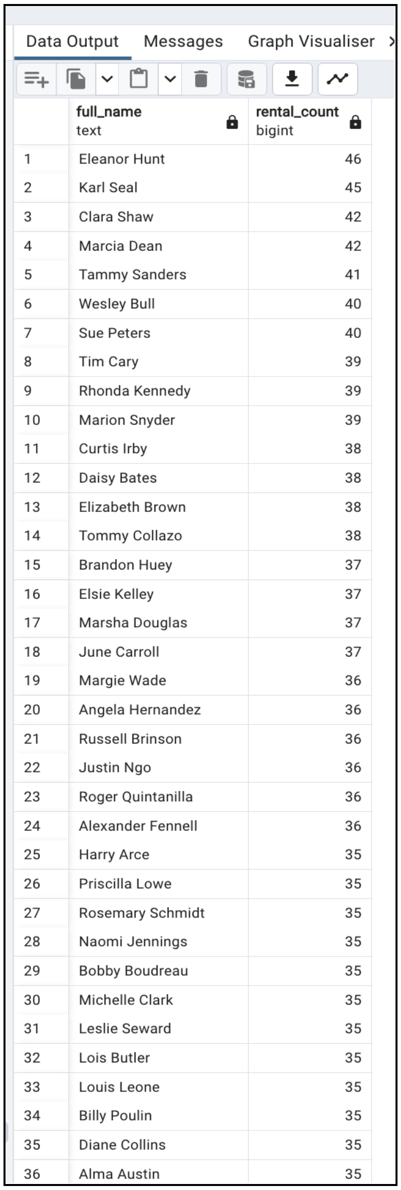
1. **Create the DVDRental database in Postgres by adding a new database and “restoring” the DB using the dvdrental.tar file. (All tables will exist in the Public schema)**


Tables (15)
- actor
- address
- category
- city
- country
- customer
- film
- film_actor
- film_category
- inventory
- language
- payment
- rental
- staff
- store

2. **Which customer had the most rentals? Return the first name and last name as a single column and the count of rentals per customer in descending order. (Table is truncated)**
   a. Eleanor Hunt - 46 Rentals

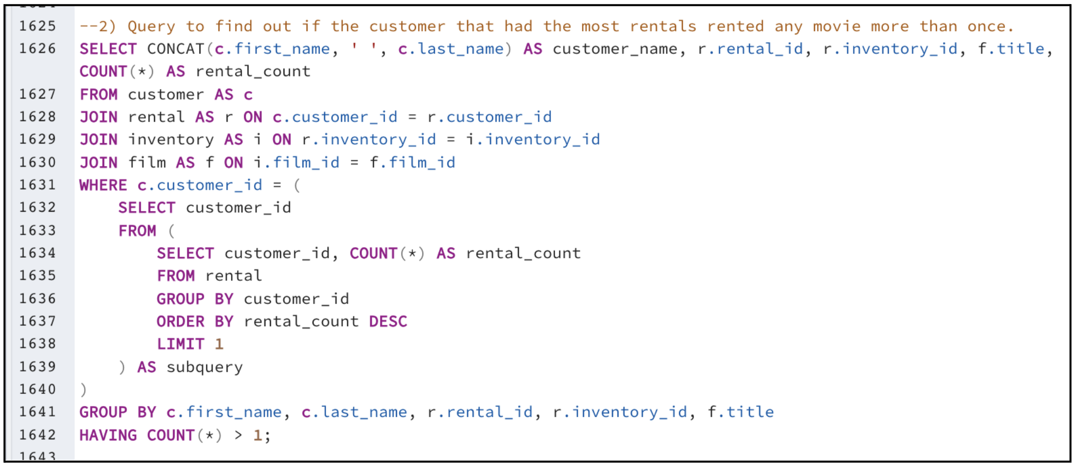

```
-- 1) ind the customer who had the most rentals and retrieve their first name and last name as a single
column, along with the count of rentals per customer in descending order.

SELECT c.first_name || ' ' || c.last_name AS full_name, COUNT(r.rental_id) AS rental_count
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
GROUP BY c.customer_id, full_name
ORDER BY rental_count DESC;
```



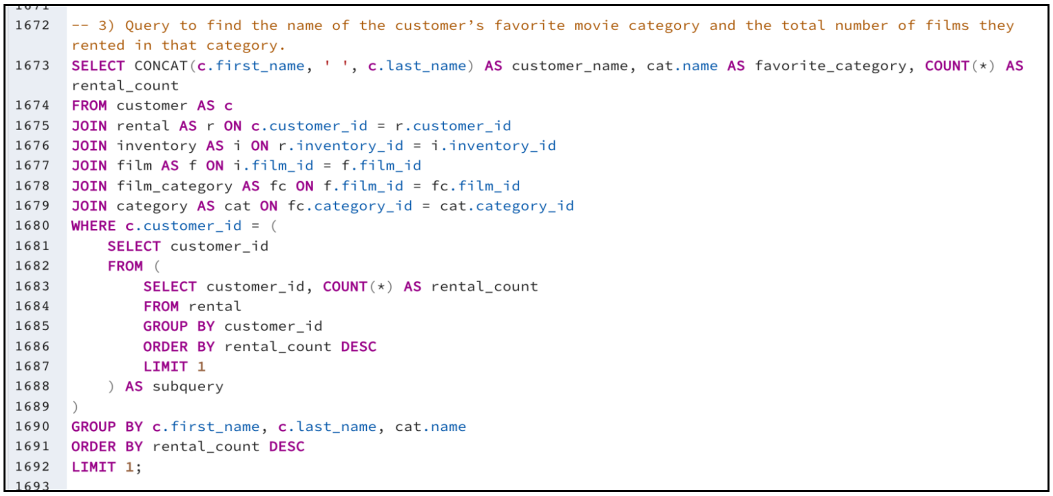
	full_name text	rental_count bigint
1	Eleanor Hunt	46
2	Karl Seal	45
3	Clara Shaw	42
4	Marcia Dean	42
5	Tammy Sanders	41
6	Wesley Bull	40
7	Sue Peters	40
8	Tim Cary	39
9	Rhonda Kennedy	39
10	Marion Snyder	39
11	Curtis Irby	38
12	Daisy Bates	38
13	Elizabeth Brown	38
14	Tommy Collazo	38
15	Brandon Huey	37
16	Elsie Kelley	37
17	Marsha Douglas	37
18	June Carroll	37
19	Margie Wade	36
20	Angela Hernandez	36
21	Russell Brinson	36
22	Justin Ngo	36
23	Roger Quintanilla	36
24	Alexander Fennell	36
25	Harry Arce	35
26	Priscilla Lowe	35
27	Rosemary Schmidt	35
28	Naomi Jennings	35
29	Bobby Boudreau	35
30	Michelle Clark	35
31	Leslie Seward	35
32	Lois Butler	35
33	Louis Leone	35
34	Billy Poulin	35
35	Diane Collins	35
36	Alma Austin	35

3. **Did that customer rent any movie more than once? (provide the query to validate):** No



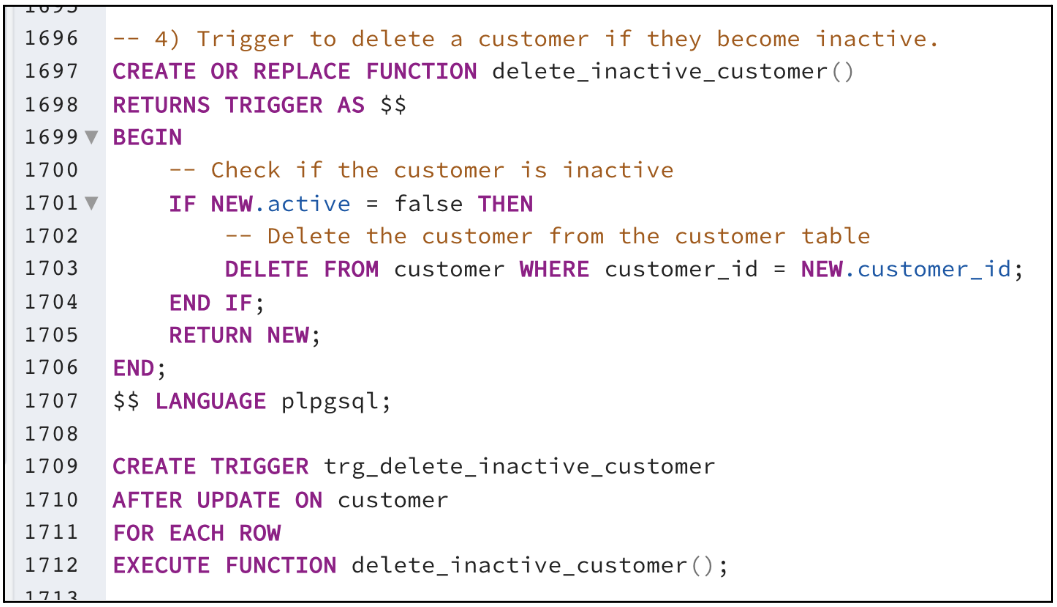
```sql
--2) Query to find out if the customer that had the most rentals rented any movie more than once.
SELECT CONCAT(c.first_name, ' ', c.last_name) AS customer_name, r.rental_id, r.inventory_id, f.title, COUNT(*) AS rental_count
FROM customer AS c
JOIN rental AS r ON c.customer_id = r.customer_id
JOIN inventory AS i ON r.inventory_id = i.inventory_id
JOIN film AS f ON i.film_id = f.film_id
WHERE c.customer_id = (
    SELECT customer_id
    FROM (
        SELECT customer_id, COUNT(*) AS rental_count
        FROM rental
        GROUP BY customer_id
        ORDER BY rental_count DESC
        LIMIT 1
    ) AS subquery
)
GROUP BY c.first_name, c.last_name, r.rental_id, r.inventory_id, f.title
HAVING COUNT(*) > 1;
```

4. **What is the customer’s favorite movie category? Return the name and the total number of films they have rented in that category.** Favorite Category: Sci Fi Number of films rented: 7



```sql
-- 3) Query to find the name of the customer’s favorite movie category and the total number of films they rented in that category.
SELECT CONCAT(c.first_name, ' ', c.last_name) AS customer_name, cat.name AS favorite_category, COUNT(*) AS rental_count
FROM customer AS c
JOIN rental AS r ON c.customer_id = r.customer_id
JOIN inventory AS i ON r.inventory_id = i.inventory_id
JOIN film AS f ON i.film_id = f.film_id
JOIN film_category AS fc ON f.film_id = fc.film_id
JOIN category AS cat ON fc.category_id = cat.category_id
WHERE c.customer_id = (
    SELECT customer_id
    FROM (
        SELECT customer_id, COUNT(*) AS rental_count
        FROM rental
        GROUP BY customer_id
        ORDER BY rental_count DESC
        LIMIT 1
    ) AS subquery
)
GROUP BY c.first_name, c.last_name, cat.name
ORDER BY rental_count DESC
LIMIT 1;
```

5. **Write a trigger to delete a customer if they become inactive.**



```sql
-- 4) Trigger to delete a customer if they become inactive.
CREATE OR REPLACE FUNCTION delete_inactive_customer()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if the customer is inactive
    IF NEW.active = false THEN
        -- Delete the customer from the customer table
        DELETE FROM customer WHERE customer_id = NEW.customer_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_delete_inactive_customer
AFTER UPDATE ON customer
FOR EACH ROW
EXECUTE FUNCTION delete_inactive_customer();
```

Writing Assignment:

How can denormalization aid business users answer questions on the database more efficiently? Would it be more advantageous to create a VIEW or to use NoSQL technology? List pros and cons of both approaches. Your answer should be a minimum of four paragraphs explaining: how denormalization can help and a comparative analysis for VIEWs vs NoSQL. You will need to perform research online or at the library to answer this question, be sure to cite at LEAST 3 pros and 3 cons for each approach. Cite all sources for your answer.

Denormalization can help business users answer questions on a database more efficiently by improving query performance and simplifying data retrieval because it adds redundant data to tables, which reduces the need for complex joins and allows faster data access. This can be particularly beneficial for business users who need to retrieve data quickly and efficiently to make informed decisions.

One way to leverage denormalization is to create a "**VIEW**" in the database, which is a virtual table based on the result of a query. This process allows business users to retrieve precomputed denormalized data without directly modifying the underlying tables.

<u>Pros of using **VIEWs**:</u>

1. Simplified data access: **VIEWs** provide simplified and consolidated data views, which makes it easier for business users to query and retrieve information they need.
2. Improved performance: **VIEWs** can be optimized for specific queries, which can result in faster data retrieval compared to performing complex joins on normalized tables.
3. Data security: **VIEWs** can be used to restrict access to sensitive data by controlling the columns and rows that are exposed to users.

<u>Cons of using **VIEWs**:</u>

1. Increased storage requirements: **VIEWs** store redundant data, which increases the storage requirements of the database.
2. Data redundancy: **VIEWs** introduce redundancy, which could possibly lead to inconsistencies in the data, especially if the underlying tables are updated independently.
3. Maintenance complexity: As the underlying tables change, **VIEWs** may need to be updated to reflect the changes, this can introduce additional problems and maintenance overhead.

Using **NoSQL** technology would be another option. **NoSQL** databases can handle large volumes of unstructured or semi-structured data and provide high scalability and performance. They often use denormalization as a fundamental principle to optimize data retrieval.

<u>Pros of using **NoSQL** technology:</u>

1. Scalability: **NoSQL** databases are designed to scale horizontally, which can allow for better handling of large amounts of data and high traffic loads.
2. Flexibility: **NoSQL** databases offer flexible schema designs, which allows for easy revamping to changing business requirements.
3. Performance: **NoSQL** databases provide faster data retrieval as compared to traditional relational databases, especially for read-heavy workloads.

Cons of using **NoSQL** technology:

1. Lack of standardization: **NoSQL** databases often lack a standardized query language, which could make it more challenging for business users to write complex queries.
2. Limited transaction support: **NoSQL** databases may have limited support for transactions, which can be disadvantageous for applications that require strict data consistency.
3. Learning curve: **NoSQL** databases require a different mindset and skill set compared to traditional relational databases, which may require additional training and resources for business users.

Whether or not to create a **VIEW** in a relational database or use **NoSQL** technology is going to depend on a variety of factors, including the specific requirements of the application, nature of the data, data model, scalability needs.and any trade-offs the user is willing to make. It is important for the user to carefully evaluate the advantages and trade-offs of each option in the context of their application before making a decision.

Sources:

1. What is denormalization and how does it work? TechTarget
   https://www.techtarget.com/searchdatamanagement/definition/denormalization

2. Denormalization in Databases - GeeksforGeeks
   https://www.geeksforgeeks.org/denormalization-in-databases/

3. Data Denormalization: Pros, Cons & Techniques for Denormalizing Data | Splunk
   https://www.splunk.com/en_us/blog/learn/data-denormalization.html

4. The Advantages of Views in SQL to Organize Complex Queries
   https://www.headway.io/blog/advantages-of-sql-views

5. Advantages and Disadvantages of views in Sql Server
   https://www.c-sharpcorner.com/blogs/advantages-and-disadvantages-of-views-in-sql-server1

6. Benefits of Using Views – https://edtechbooks.org/learning_mysql/benefits_of_using_vi