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)
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. *Use aggregation and CTE statement*

**ANSWER:**

Eleanor Hunt

Query to return all customers and their count of rentals:

SELECT COUNT(\*), CONCAT(first\_name,' ',last\_name) AS full\_name

FROM customer

JOIN rental ON customer.customer\_id=rental.customer\_id

GROUP BY customer.customer\_id

ORDER BY COUNT(\*) DESC

;

Query to return only the customer with the most rentals:

WITH customer\_rentals AS (

SELECT COUNT(\*) AS rentals, customer.customer\_id AS ci

FROM customer

JOIN rental ON customer.customer\_id=rental.customer\_id

GROUP BY customer.customer\_id

)

SELECT rentals, CONCAT(first\_name,' ',last\_name) AS full\_name

FROM customer\_rentals

JOIN customer ON customer\_rentals.ci=customer.customer\_id

WHERE rentals = (SELECT MAX(rentals) FROM customer\_rentals)

;

1. Did that customer rent any movie more than once? (provide the query to validate)

**ANSWER:**

No

Query:

WITH customer\_rentals AS (

SELECT COUNT(\*) AS rentals, customer.customer\_id AS ci

FROM customer

JOIN rental ON customer.customer\_id=rental.customer\_id

GROUP BY customer.customer\_id

), max\_rentals AS (

SELECT rentals,ci

FROM customer\_rentals

WHERE rentals = (SELECT MAX(rentals) FROM customer\_rentals)

)

SELECT COUNT(\*), film.film\_id, film.title

FROM customer

JOIN rental ON customer.customer\_id=rental.customer\_id

JOIN inventory ON rental.inventory\_id=inventory.inventory\_id

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

WHERE customer.customer\_id IN (SELECT ci FROM max\_rentals)

GROUP BY film.film\_id

HAVING COUNT(\*)>1

;

1. What is the customer’s favorite movie category? Return the name and the total number of films they have rented in that category.

**ANSWER:**

Sci-fi

Query:

WITH customer\_rentals AS (

SELECT COUNT(\*) AS rentals, customer.customer\_id AS ci

FROM customer

JOIN rental ON customer.customer\_id=rental.customer\_id

GROUP BY customer.customer\_id

), max\_rentals AS (

SELECT rentals,ci

FROM customer\_rentals

WHERE rentals = (SELECT MAX(rentals) FROM customer\_rentals)

), categories AS (

SELECT customer.customer\_id, category.category\_id, category.name

FROM customer

JOIN rental ON customer.customer\_id=rental.customer\_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

)

SELECT COUNT(\*), categories.name

FROM categories

WHERE categories.customer\_id IN (SELECT ci FROM max\_rentals)

GROUP BY categories.name

ORDER BY COUNT(\*) DESC

LIMIT 1

;

A screenshot of a computer

AI-generated content may be incorrect.

1. Write a trigger to delete a customer if they become inactive.

**ANSWER:**

CREATE OR REPLACE FUNCTION delete\_inactive()

RETURNS trigger

AS

$BODY$

BEGIN

IF (NEW.activebool=FALSE)

THEN DELETE FROM customer WHERE customer.customer\_id=New.customer\_id ;

END IF;

RETURN NULL; -- because this is an AFTER event trigger

END;

$BODY$

LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER inactive\_customer

AFTER UPDATE ON customer

FOR EACH ROW

EXECUTE FUNCTION delete\_inactive();

1. 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 a 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.

*What is denormalization?*

Denormalization is the process of adding redundancy into one or more tables in a normalized database. The goal of denormalization is to improve query performance, thus allowing business users to access the data for which they are searching faster. This is achieved by reducing the number of joins the user must perform on the tables in the schema. In a normalized schema, data is organized in multiple tables without repeating attributes throughout the different tables. While this reduces storage space, eliminates data duplication, and improves update process, the normalized schema is more complex with multiple tables. The user has to search these tables, which can become a daunting task depending the number of tables needed to search and the size of each table. To reduce the effort required by the user, the schema can be denormalized by repeating some attributes across multiple tables, so that it is likely the user will not have to access as many tables in order to find the right information. All in all, data denormalization allows for faster and simpler queries, improved read performance, and less effort by the user, all of which enables business users to access data more efficiently.

*Textbook ch 1*

<https://www.geeksforgeeks.org/denormalization-in-databases/>

<https://www.ibm.com/think/topics/data-redundancy>

A view, or virtual table, in SQL is a table that is derived from one or more tables in the database and does not always exist physically with the database system. The advantages and disadvantages of views essentially break down into the increase in efficiency for users and the challenges in updating and maintaining views, respectively. Views make searching for data simpler for business users. A view may be derived from multiple tables, meaning that when a view is created for the user, complex querying steps, like multiple joins and aggregation, are already completed. The user is not required to have knowledge of or perform these querying steps, and because the view is created prior to the user attempting to access the data, the time for the user to access the data is reduced because the view has been previously computed. Views can be tailored to specific users without affecting the base tables from which the view is derived. Multiple views can be created for specific users depending on needs, relevance, and access privileges. Ambiguous or cryptic column names can be renamed in views so that the user can better understand and interpret the data they are receiving. Views may also be used to limit access to data to ensure data privacy. Since views can be specifically tailored for users, sensitive data can be omitted while the user still quickly receives the data they need and for which they have clearance.

Implementing views so that they improve query performance while ensuring the view is accurate and consistent with its base tables yields a few challenges. The two strategies to implement views, query modification and view materialization, both have disadvantages. Query modification is inefficient for views defined by complex queries that take considerable computational time to compute. Its inefficiencies are exacerbated by multiple queries on the view in a short time. View materialization may resolve this by physically storing a view while it is consistently being queried over a period of time; however, this requires the use of additional storage space, and update strategies must be implemented to ensure the data in the view is consistent with the base tables. The lazy update strategy, which updates the view once it is queried, may slow down query execution. The periodic update strategy, which involves updating the view at set times, may allow the querying of inaccurate data in between updates. Finally, UPDATE, INSERT, and DELETE commands often do not work on views. If a view consists of multiple joined tables or utilizes grouping and aggregation functions, it is not updatable.

*Textbook chapter 7*

<https://www.datacamp.com/tutorial/views-in-sql>

<https://www.geeksforgeeks.org/sql-views/>

NoSQL, or non-relational database, technologies offer many advantages over SQL. NoSQL has high scalability, meaning that increases in data and traffic can be handled with a scale-out architecture in which data is stored and processed over a large number of computers. For this reason, NoSQL technologies are efficient for growing systems that have lots of users. Additionally, NoSQL technologies store data in simple forms that are sometimes easier to understand than the data models created in relational databases. Due to this simplicity, schemas and data are easily updated without requiring significant changes to the database. Finally, NoSQL technologies offer better performance with big data. Non-relational databases make large amounts of data available to the user quickly.

While NoSQL makes big data available fast, it does so at the expense of consistency. NoSQL has limited ACID support and sacrifices consistency to offer quick availability and partition tolerance (the ability for the system to continue working even after a communication break between nodes). Another disadvantage to NoSQL technologies are their lack of support for complex queries. Compared to SQL, NoSQL does not have the same JOIN or advanced query operations, which can slow down users in complex data analysis.