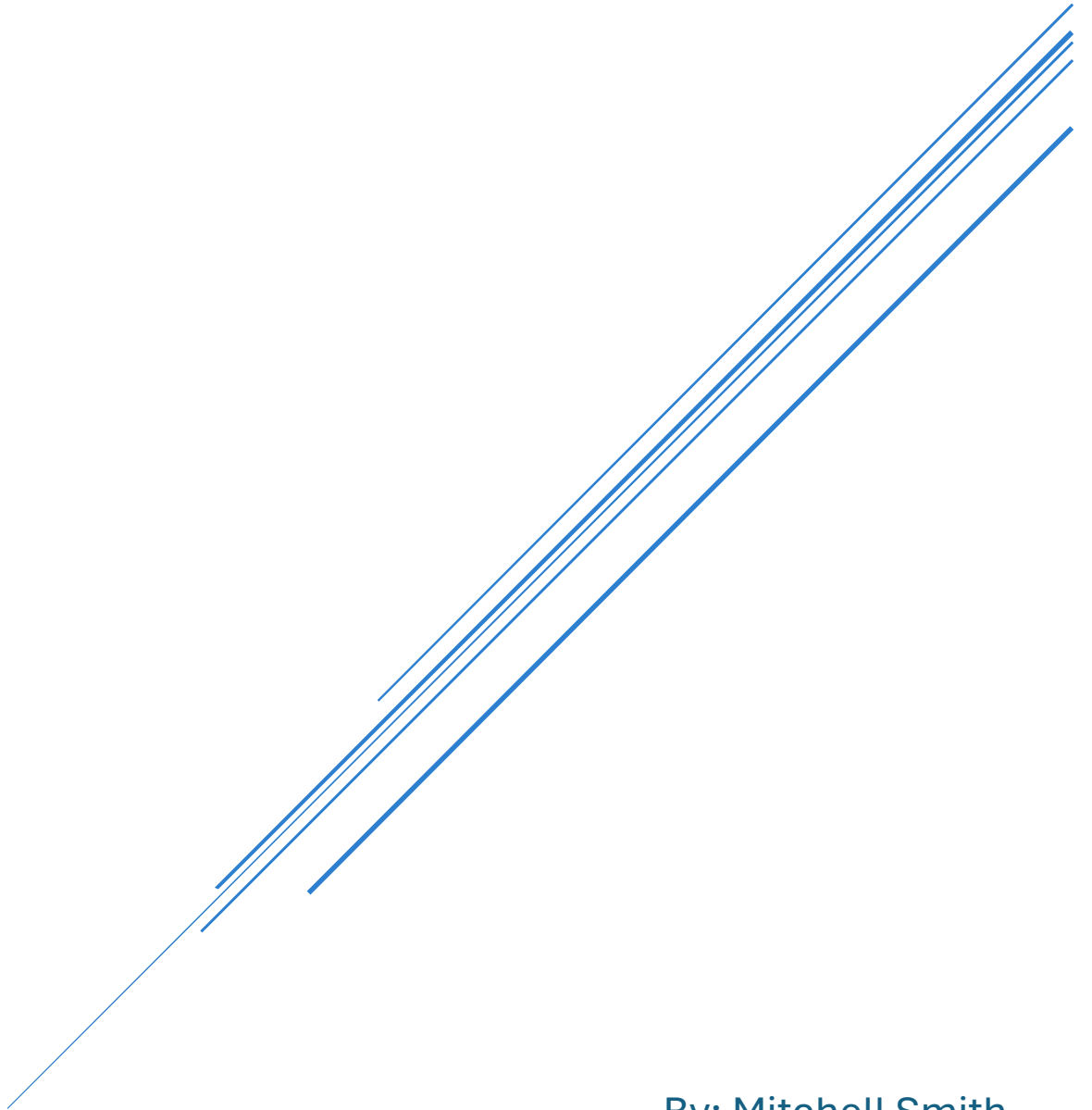


ASSIGNMENT #4

DBAS3035



By: Mitchell Smith
W0440925

Table of Contents

Introduction	2
Part A: SQL Queries.....	2
1. Create two queries, which are 1) using subquery and 2) using CTE. Two queries should provide the same result. Make sure to provide what your queries mean.	2
2. Run EXPLAIN ANALYZE on each query. Compare the result of running EXPLAIN ANALYZE and provide an explanation of why they are different and/or the same.	5
Part B: Recursive CTE Creation.....	8
1. Create the recursive CTE named managers so that it will show all the managers (manager's hierarchy) of the employee with ID =15. It is based on the dataset given in the class.....	8
Conclusion.....	9
References	9

Introduction

In this document, I will demonstrate the completion of all assignment requirements regarding Structured Query Language (SQL). I will accomplish this by using the “DVD-Rental” sample database from the PostgreSQL Tutorial website and creating queries with sub-queries and Common Table Expressions (CTE), running the ‘EXPLAIN ANALYZE’ commands on each query, creating a recursive CTE using the sample scripts from the ‘Recursive Query’ section of the PostgreSQL Tutorial website, and providing both my basic and intermediate Hacker Rank Certifications.

Part A: SQL Queries

1. Create two queries, which are 1) using subquery and 2) using CTE. Two queries should provide the same result. Make sure to provide what your queries mean.

Query #1 (Sub-Query)

Screenshot of Query:

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
1 SELECT DISTINCT c.first_name, c.last_name, f.title, f.rating, r.rental_date, f.rental_rate
2 FROM customer c
3 JOIN rental r ON c.customer_id = r.customer_id
4 JOIN inventory i ON r.inventory_id = i.inventory_id
5 JOIN film f ON i.film_id = f.film_id
6 WHERE f.film_id IN (
7     SELECT f.film_id
8     FROM film f
9     WHERE f.rental_rate BETWEEN 2.99 AND 4.99
10    AND f.rating = 'R'
11 )
12 AND r.rental_date BETWEEN '2005-07-01' AND '2005-07-31'
13 ORDER BY r.rental_date ASC;
```

The results are displayed in a table with the following columns: first_name, last_name, title, rating, rental_date, and rental_rate. The table contains 6 rows of data.

	first_name	last_name	title	rating	rental_date	rental_rate
1	Kirk	Stclair	Galaxy Sweethearts	R	2005-07-05 23:30:36	4.99
2	Max	Pitt	Kissing Dolls	R	2005-07-05 23:50:04	4.99
3	Marcus	Hidalgo	Head Stranger	R	2005-07-06 00:02:08	4.99
4	Leah	Curtis	Jeepers Wedding	R	2005-07-06 00:11:28	2.99
5	Ellen	Simpson	Devil Desire	R	2005-07-06 00:15:06	4.99
6	Tonya	Chapman	Bear Graceland	R	2005-07-06 00:18:29	2.99

Total rows: 724 of 724 Query complete 00:00:00.459

Screenshot of Result Set:

Data Output Messages Notifications						
	first_name character varying (45)	last_name character varying (45)	title character varying (255)	rating mpaa_rating	rental_date timestamp without time zone	rental_rate numeric (4,2)
1	Kirk	Stclair	Galaxy Sweethearts	R	2005-07-05 23:30:36	4.99
2	Max	Pitt	Kissing Dolls	R	2005-07-05 23:50:04	4.99
3	Marcus	Hidalgo	Head Stranger	R	2005-07-06 00:02:08	4.99
4	Leah	Curtis	Jeepers Wedding	R	2005-07-06 00:11:28	2.99
5	Ellen	Simpson	Devil Desire	R	2005-07-06 00:15:06	4.99
6	Tonya	Chapman	Bear Graceland	R	2005-07-06 00:18:29	2.99
7	Francisco	Skidmore	Jet Neighbors	R	2005-07-06 00:27:41	4.99
8	Monica	Hicks	Quest Mussolini	R	2005-07-06 00:48:55	2.99
9	Vicki	Fields	Home Pity	R	2005-07-06 01:13:27	4.99
10	Zachary	Hite	Tuxedo Mile	R	2005-07-06 01:36:53	2.99
11	Leroy	Bustamante	Streetcar Intentions	R	2005-07-06 02:16:17	4.99

Explanation of Query:

In this query, I selected distinct columns pertaining to the names of customers, the movies they rented, the rating of the movies, when they rented the movies, and how much they rented the movies for with the purpose of determining which customers rented rated 'R' movies that are affordable and were rented specifically during the month of July 2005.

I did this by initially selecting the relevant columns that I wanted to display in the output, giving them aliases, and then using the 'customer' table to join the 'rental', 'inventory', and 'film' tables together by using similar columns between the tables that establish relationships between them and applying the correct aliases to each table as well.

Next, I used a 'WHERE' and 'IN' clause with the 'film_id' from the 'film' table to insert a subquery within the original query that worked together to filter the 'film_id' column to display only rated 'R' movies that were rented by customers in the output. Additionally, I added an 'AND' clause within the subquery to add an additional filter that displays only the rated 'R' movies that were rented for a price of between \$2.99 and \$4.99.

After I implemented the closing bracket to complete the subquery, I then added an extra filter at the end of my code using another 'AND' clause (since I already used a 'WHERE' clause earlier on in the outer query) to find the affordable, rated 'R' movies that were rented between July 1st, 2005, and July 31st, 2005. At the end of my query, I also added an 'ORDER BY' clause to place the 'rental_date' column from the 'rental' table in ascending order.

Query #2 (CTE)

Screenshot of Query:

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
18 WITH Affordable_Rated_R_Movies AS (  
19     SELECT DISTINCT c.first_name, c.last_name, f.title, f.rating, r.rental_date, f.rental_rate  
20     FROM customer c  
21     JOIN rental r ON c.customer_id = r.customer_id  
22     JOIN inventory i ON r.inventory_id = i.inventory_id  
23     JOIN film f ON i.film_id = f.film_id  
24     WHERE f.film_id IN (  
25         SELECT f.film_id  
26         FROM film f  
27         WHERE f.rental_rate BETWEEN 2.99 AND 4.99  
28         AND f.rating = 'R'  
29     )  
30     AND r.rental_date BETWEEN '2005-07-01' AND '2005-07-31'  
31 )  
32 SELECT * FROM Affordable_Rated_R_Movies  
33 ORDER BY rental_date ASC;
```

Below the query, the 'Data Output' tab shows the results of the query. The results are displayed in a table with the following columns: first_name, last_name, title, rating, rental_date, and rental_rate. The results are ordered by rental_date ASC.

	first_name character varying (45)	last_name character varying (45)	title character varying (255)	rating mpaa_rating	rental_date timestamp without time zone	rental_rate numeric (4,2)
1	Kirk	Stclair	Galaxy Sweethearts	R	2005-07-05 23:30:36	4.99
2	Max	Pitt	Kissing Dolls	R	2005-07-05 23:50:04	4.99
3	Marcus	Hidalgo	Head Stranger	R	2005-07-06 00:02:08	4.99

Screenshot of Result Set:

	first_name character varying (45)	last_name character varying (45)	title character varying (255)	rating mpaa_rating	rental_date timestamp without time zone	rental_rate numeric (4,2)
1	Kirk	Stclair	Galaxy Sweethearts	R	2005-07-05 23:30:36	4.99
2	Max	Pitt	Kissing Dolls	R	2005-07-05 23:50:04	4.99
3	Marcus	Hidalgo	Head Stranger	R	2005-07-06 00:02:08	4.99
4	Leah	Curtis	Jeepers Wedding	R	2005-07-06 00:11:28	2.99
5	Ellen	Simpson	Devil Desire	R	2005-07-06 00:15:06	4.99
6	Tonya	Chapman	Bear Graceland	R	2005-07-06 00:18:29	2.99
7	Francisco	Skidmore	Jet Neighbors	R	2005-07-06 00:27:41	4.99
8	Monica	Hicks	Quest Mussolini	R	2005-07-06 00:48:55	2.99
9	Vicki	Fields	Home Pity	R	2005-07-06 01:13:27	4.99
10	Zachary	Hite	Tuxedo Mile	R	2005-07-06 01:36:53	2.99
11	Leroy	Bustamante	Streetcar Intentions	R	2005-07-06 02:16:17	4.99

Explanation of Query:

In this query, I created a Common Table Expression (CTE) that consisted of the contents of my original query that I created in the “Query #1 (Sub-Query)” section, where I selected distinct columns pertaining to the names of customers, the movies they rented, the rating of the movies, when they rented the movies, and how much they rented the movies for with the purpose of determining which customers rented rated ‘R’ movies that

are affordable and were rented specifically during the month of July 2005. The goal of this CTE was to basically create an alias that references my entire query with a specific name that I can use when writing other queries or to quickly generate the table of results from my original query.

I did this by using the ‘WITH’ clause to create a CTE, and then naming it ‘Affordable_Rated_R_Movies’ and opening a set of parentheses, where I enclosed all essential parts of my original query that generated the table of results inside of it.

After I enclosed my original query within the parentheses to define the CTE, I then selected all the contents from the table of results of my original query using the ‘Affordable_Rated_R_Movies’ alias that I defined when creating the CTE. Additionally, I used the ‘ORDER BY’ clause to order the result set in the same way that I ordered the results of the original query. As a result, the table of results of my original query quickly generated.

2. Run EXPLAIN ANALYZE on each query. Compare the result of running EXPLAIN ANALYZE and provide an explanation of why they are different and/or the same.

Explain Analyze Query Screenshots

Query #1 (Sub-Query) Screenshot A:

Data Output Messages Explain × Notifications				
Graphical Analysis Statistics				
#	Node	Rows Actual	Loops	
1.	→ Unique (rows=724 loops=1)		724	1
2.	→ Sort (rows=724 loops=1)		724	1
3.	→ Hash Inner Join (rows=724 loops=1) Hash Cond: (i.film_id = f.film_id)		724	1
4.	→ Hash Inner Join (rows=724 loops=1) Hash Cond: (r.customer_id = c.customer_id)		724	1
5.	→ Hash Inner Join (rows=724 loops=1) Hash Cond: (r.inventory_id = i.inventory_id)		724	1
6.	→ Index Only Scan using idx_unq_rental_rental_date_inventory_id_customer_id on rental as r (...) Index Cond: ((rental_date >= '2005-07-01 00:00:00'::timestamp without time zone) AND (rental_date	6030		1
7.	→ Hash (rows=566 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 33 kB		566	1
8.	→ Hash Inner Join (rows=566 loops=1) Hash Cond: (i.film_id = f_1.film_id)		566	1
9.	→ Seq Scan on inventory as i (rows=4581 loops=1)		4581	1
10.	→ Hash (rows=125 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 13 kB		125	1

Query #1 (Sub-Query) Screenshot B:

10.	→ Hash (rows=125 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 13 kB	125	1
11.	→ Seq Scan on film as f_1 (rows=125 loops=1) Filter: ((rental_rate >= 2.99) AND (rental_rate <= 4.99) AND (rating = 'R'::mpaa_rating)) Rows Removed by Filter: 875	125	1
12.	→ Hash (rows=599 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 39 kB	599	1
13.	→ Seq Scan on customer as c (rows=599 loops=1)	599	1
14.	→ Hash (rows=1000 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 72 kB	1000	1
15.	→ Seq Scan on film as f (rows=1000 loops=1)	1000	1

Query #2 (CTE) Screenshot A:

Data Output Messages Explain X Notifications			
Graphical Analysis Statistics			
#	Node	Actual	Loops
1.	→ Sort (rows=724 loops=1)	724	1
2.	→ Aggregate (rows=724 loops=1) Buckets: Batches: Memory Usage: 169 kB	724	1
3.	→ Hash Inner Join (rows=724 loops=1) Hash Cond: (i.film_id = f.film_id)	724	1
4.	→ Hash Inner Join (rows=724 loops=1) Hash Cond: (r.customer_id = c.customer_id)	724	1
5.	→ Hash Inner Join (rows=724 loops=1) Hash Cond: (r.inventory_id = i.inventory_id)	724	1
6.	→ Index Only Scan using idx_unq_rental_rental_date_inventory_id_customer_id on rental as r (...) Index Cond: ((rental_date >= '2005-07-01 00:00:00'::timestamp without time zone) AND (rental_date	6030	1
7.	→ Hash (rows=566 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 33 kB	566	1
8.	→ Hash Inner Join (rows=566 loops=1) Hash Cond: (i.film_id = f_1.film_id)	566	1
9.	→ Seq Scan on inventory as i (rows=4581 loops=1)	4581	1
10.	→ Hash (rows=125 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 13 kB	125	1

Query #2 (CTE) Screenshot B:

10.	→ Hash (rows=125 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 13 kB	125	1
11.	→ Seq Scan on film as f_1 (rows=125 loops=1) Filter: ((rental_rate >= 2.99) AND (rental_rate <= 4.99) AND (rating = 'R'::mpaa_rating)) Rows Removed by Filter: 875	125	1
12.	→ Hash (rows=599 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 39 kB	599	1
13.	→ Seq Scan on customer as c (rows=599 loops=1)	599	1
14.	→ Hash (rows=1000 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 72 kB	1000	1
15.	→ Seq Scan on film as f (rows=1000 loops=1)	1000	1

Execution Plan Comparison

Observation #1: The first difference that I noticed when running the 'EXPLAIN ANALYZE' command was that query #1 took 0.967 seconds to execute, while query #2 took only 0.174 seconds to execute. This is happening because when a CTE is create in a database, it is stored in the databases memory, which allows the CTE to perform the quickest and most efficient operation to produce the result set of the query.

Observation #2: The second difference that I noticed between the two queries was that query #1 began with a unique operation, while query #2 began with an aggregate operation.

This happens because in my original query that contained the sub-query (query #1), I began by using a 'SELECT' statement with the 'DISTINCT' clause, which forced the execution plan to begin with a unique operation to eliminate duplicate rows. I think that the query that I created with a CTE (query #2) began with an aggregate function because the database optimizer chose aggregate operations to be the most efficient way of producing the result set of the table within the CTE.

Observation #3: The two queries both seem to use hash inner join conditions. This is happening because both queries are essentially performing the same operations and have the same overall structure, which includes selecting the same columns, joining the same tables on the same columns, and using the same conditions / filters.

Observation #4: Both queries contain the same number of steps as one another. The two queries both seem to use hash inner join conditions. This is happening because both queries are essentially performing the same operations and have the same overall structure, which includes selecting the same columns, joining the same tables on the same columns, and using the same conditions / filters.

Observation #5: Both queries perform the same sequential scans as one another. The two queries both seem to use hash inner join conditions. This is happening because both queries are essentially performing the same operations and have the same overall structure, which includes selecting the same columns, joining the same tables on the same columns, and using the same conditions / filters.

Observation #6: Both queries perform the same hash operations. The two queries both seem to use hash inner join conditions. This is happening because both queries are essentially performing the same operations and have the same overall structure, which includes selecting the same columns, joining the same tables on the same columns, and using the same conditions / filters.

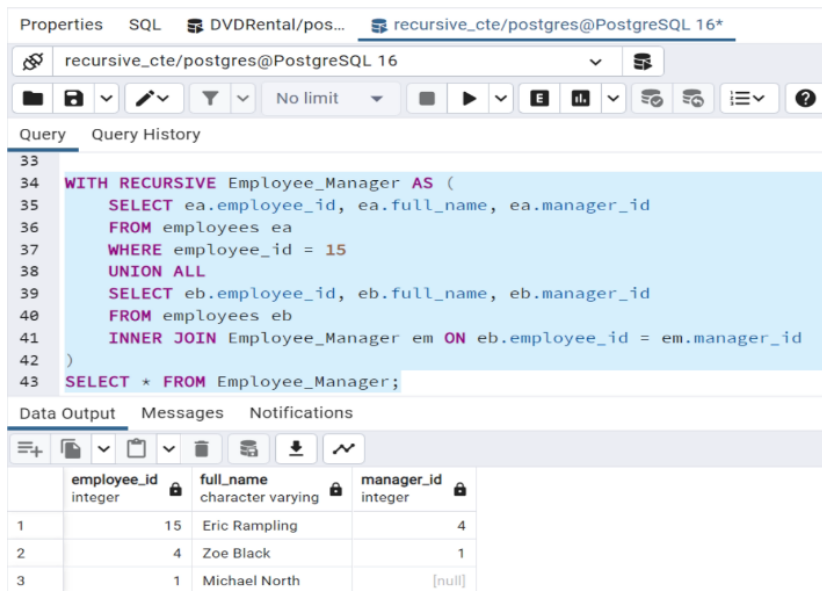
Observation #7: Both queries perform the same index scan on the rental table. The two queries both seem to use hash inner join conditions. This is happening because both queries are essentially performing the same operations and have the same overall structure, which includes selecting the same columns, joining the same tables on the same columns, and using the same conditions / filters.

Part B: Recursive CTE Creation

1. Create the recursive CTE named managers so that it will show all the managers (manager's hierarchy) of the employee with ID =15. It is based on the dataset given in the class.

Query (Recursive CTE)

Screenshot of Query & Result Set:



The screenshot shows a PostgreSQL query editor window. The query is as follows:

```
33
34 WITH RECURSIVE Employee_Manager AS (
35     SELECT ea.employee_id, ea.full_name, ea.manager_id
36     FROM employees ea
37     WHERE employee_id = 15
38     UNION ALL
39     SELECT eb.employee_id, eb.full_name, eb.manager_id
40     FROM employees eb
41     INNER JOIN Employee_Manager em ON eb.employee_id = em.manager_id
42 )
43 SELECT * FROM Employee_Manager;
```

The result set is displayed below the query, showing three rows of data:

	employee_id integer	full_name character varying	manager_id integer
1	15	Eric Rampling	4
2	4	Zoe Black	1
3	1	Michael North	[null]

Explanation of Query:

In this query, I began by using a 'WITH' clause to create a Common Table Expression (CTE) and added the additional 'RECURSIVE' clause to define this CTE as a recursive CTE, which will allow the CTE to return all rows that relate to one another in a hierarchical structure. I then defined the name of the CTE 'Employee_Manager'.

After creating the CTE and defining it as a recursive CTE, I opened a set of parentheses and selected all columns that I wanted to display in the output, used a 'FROM' clause to define the table that the columns are contained in (while also defining aliases), and I used a 'WHERE' clause to define the specific employee id that I wanted to initially retrieve and use to create the hierarchical structure. The first part of the recursive CTE is known as the anchor.

Next, I used the 'UNION ALL' clauses to unify the anchor member output and the recursive member output that I'm about to create so that the recursive part of the query can generate the sub-info that is related to the anchor part of the query.

I then selected the same relevant columns that I previously selected in the anchor part of my query and used a 'FROM' clause to define the table that the columns are contained in once again while defining different aliases. Finally, I used a 'JOIN' and 'ON' clause to join the CTE with the employees table using the 'employee_id' and 'manager_id' columns since they correspond with one another. By using this join, it creates a loop of joining information that matches with one another in a hierarchical structure until it can not anymore.

Finally, after closing the parentheses of the recursive CTE, I used a 'SELECT *' command with the name that I defined for the recursive CTE to select all the contents from the CTE, which generated all information in a hierarchical structure based on the anchor part of the CTE.

Conclusion

In this document, I have successfully demonstrated the completion of all assignment requirements regarding Structured Query Language (SQL). I have accomplished this by using the "DVD-Rental" sample database from the PostgreSQL Tutorial website and creating queries with sub-queries and Common Table Expressions (CTE), running the 'EXPLAIN ANALYZE' commands on each query, creating a recursive CTE using the sample scripts from the 'Recursive Query' section of the PostgreSQL Tutorial website, and providing both my basic and intermediate Hacker Rank Certifications.

References

Aylward, A. (2019, October 2). *CTEs versus Subqueries*. Retrieved from Alissa in Techland: <https://www.alisa-in.tech/post/2019-10-02-ctes/>

Ayusharma0698. (2023, May 20). *CTE in SQL*. Retrieved from Geeks for Geeks: <https://www.geeksforgeeks.org/cte-in-sql/>

Babic, T. (2021, October 19). *What Is a Recursive CTE in SQL?* Retrieved from Learn SQL: <https://learnsql.com/blog/sql-recursive-cte/>

Bisso, I. L. (2022, January 19). *What Is a Common Table Expression (CTE) in SQL?* Retrieved from Learn SQL: <https://learnsql.com/blog/what-is-common-table-expression/>

Bladoszewski, K. (2020, May 22). *Subquery vs. CTE: A SQL Primer*. Retrieved from Learn SQL: <https://learnsql.com/blog/sql-subquery-cte-difference/>

ChatGPT. (n.d.). *ChatGPT*. Retrieved from ChatGPT: <https://openai.com/blog/chatgpt>

Geeks for Geeks. (2022, July 24). *SQL | Subquery*. Retrieved from Geeks for Geeks: <https://www.geeksforgeeks.org/sql-subquery/>

Hacker Rank. (n.d.). *Hacker Rank*. Retrieved from Hacker Rank: <https://www.hackerrank.com/>

Jamjala, J. (2023, March 30). *Key Notes on SQL CTE and Best Practices 2023*. Retrieved from Medium: <https://medium.com/@jagadeshjamjalarayanan/cte-what-why-how-c6cc98032422#:~:text=Optimize%20the%20Query%3A%20CTE%20can,the%20number%20of%20records%20returned.>

Lizee, A. (2020, October 21). *CTEs & Optimization*. Retrieved from Medium: <https://medium.com/alan/ctes-optimization-8c082efc47ef>

Mode Analytics, Inc. (n.d.). *Writing Subqueries in SQL*. Retrieved from Mode: <https://mode.com/sql-tutorial/sql-sub-queries>

PostgreSQL Tutorial. (n.d.). *PostgreSQL Recursive Query*. Retrieved from PostgreSQL Tutorial: <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-recursive-query/>

PostgreSQL Tutorial. (n.d.). *PostgreSQL Sample Database*. Retrieved from PostgreSQL Tutorial: <https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/>

Tech on the Net. (n.d.). *SQL: DISTINCT Clause*. Retrieved from Tech on the Net: <https://www.techonthenet.com/sql/distinct.php>

TechTFQ. (2021, September 5). *SQL WITH Clause | How to write SQL Queries using WITH Clause | SQL CTE (Common Table Expression)*. Retrieved from YouTube: <https://www.youtube.com/watch?v=QNfnuk-1YYY&t=21s>