Common Table Expressions (CTEs) and Recursive Queries - Task 8

RDBMS used : MySql

- A **Common Table Expression (CTE)** is an essential tool for simplifying complex queries and making them more readable.
- By defining **temporary result sets** (virtual table) that can be referenced multiple times, a CTE in SQL allows developers to break down complicated logic into manageable parts.
- CTEs help with **hierarchical data representation**, improve code reusability, and simplify maintenance.

SYNTAX:

```
WITH cte_name AS (
SELECT query
)
SELECT *
FROM cte_name;
```

➢ **NON-RECURSIVE CTE**

```
with AvgPriceByCat as (
        select category, avg(price) as avg_price
   from products
   group by category
)
select * from AvgPriceByCat;
```

category	avg_price
casual	950.0000
formal	3500.0000
semi-formal	3000.0000

```
with AvgPriceByCat as (
        select category, avg(price) as avg_price
   from products
   group by category
)
select p.prod_name, p.category, p.color, a.avg_price
from products p inner join AvgPriceByCat a
on p.category = a.category;
```

prod_name	category	color	avg_price
sneakers	casual	dark brown	950.0000
loafers	formal	deep black	3500.0000
boots	semi-formal	black	3000.0000
flip-flops	casual	blue	950.0000
boots	casual	black	950.0000
flip-flops	casual	pink	950.0000

- **RECURSIVE CTE**
  - A recursive CTE is one that references itself within that CTE.
  - The recursive **CTE** is useful when working with hierarchical data as the CTE continues to execute until the query returns the entire hierarchy.

  - Creating “employees” table and inserting values:

```
create table employees(
emp_id int not null,
emp_name varchar(20),
emp_role varchar(30),
manager_id int,
primary key(emp_id)
);

select * from employees;
```

emp_id	emp_name	emp_role	manager_id
101	Varun Malhotra	CEO	NULL
102	Ajay Kumar	Senior VP	101
103	Vijay Kumar	Senior VP	101
104	Gita Arora	Senior VP	101
201	Raghav	VP	102
202	Gopi	VP	103
301	Priya	Director	201
302	Philip	Director	201
303	Marcus	Director	202
401	Jaya	Senior Manager	301
402	Maya	Senior Manager	302
NULL	NULL	NULL	NULL

```
with recursive EmpHierarchy as (
select emp_id, emp_name, manager_id, 1 as level
from employees
where manager_id is null
union all
select e.emp_id, e.emp_name, e.manager_id, eh.level+1
from employees e
join EmpHierarchy eh on e.manager_id = eh.emp_id
) select * from EmpHierarchy;
```

emp_id	emp_name	manager_id	level
101	Varun Malhotra	NULL	1
102	Ajay Kumar	101	2
103	Vijay Kumar	101	2
104	Gita Arora	101	2
201	Raghav	102	3
202	Gopi	103	3
301	Priya	201	4
302	Philip	201	4
303	Marcus	202	4
401	Jaya	301	5
402	Maya	302	5