✦ Member-only story

DATA SCIENCE

How to Use Recursive CTEs in SQL: All You Need To Know in 2024

Recursive CTE In SQL References Itself To Easily Analyze Hierarchical Data. Here's How It Works!

Suraj Gurav · Follow

Published in Learning SQL · 8 min read · Jan 24, 2024

Image created by Author using Canva AI Image Generator

Common Table Expressions aka CTEs are one of the most powerful and widely used tools in SQL!

CTEs help you to simplify complex queries and to write readable, maintainable and easy-to-understand SQL queries.

You can create a temporary table from the result of a query using CTE. In fact, you can actually break down complex sub-queries into simpler CTEs.

You can read more about CTEs in one of my previous story.

5 Advanced SQL Concepts You Should Know In 2022
Master these time-saving, advanced SQL queries today.
towardsdatascience.com

However, what makes CTEs even more powerful is, its ability to analyse hierarchical data. CTEs support recursion i.e. a CTE can refer to itself to analyze data involving relationships between different entities.

Let me take an example of nesting dolls below to simplify the scenario —

Photo by Julia Kadel on Unsplash

> *Nesting dolls are souvenirs you'll get in Prague where each doll fits inside the doll bigger than itself.*

When you want to find out exactly how many dolls are there in the set, you need to look into the same set repeatedly until you find the smallest doll. Finding the smallest doll, i.e. a doll which has no smaller doll inside, is the stop condition for counting the number of dolls in a set.

Recursive CTEs follow exactly the same logic and **help you explore the dataset which has multiple layers, one below another**. With each iteration of recursive CTE, you go one level deep into the data structure until you met a specific stop condition.

In reality, you can see such type of layered data in social media networks (such as your friend circle on Facebook), chatting threads on slack, and a company's employee tree (from top management to individual employees). Recursive CTE can be super-useful while dealing with such data.

Some Basics of a Recursive CTE

Any recursive CTE contains two main parts —

1. **Anchor part** — It is the main query or you can say an initial query. So it is a starting point that can be referenced in the recursive part.
2. **Recursive part** — It is the second part of the recursive CTE which refers to the Anchor part and therefore executes iteratively as long as it satisfies some condition. So, the result of one iteration serves as input for the next iteration.

It is absolutely fine if you don't understand the above two basics in the first go!

I'll explain it with examples as you explore the following two use cases where recursive CTE can be used.

Working with Hierarchical Data

Hierarchical data, as its name suggests, contains data structured in the tree form or parent-child relationships.

You can commonly observe such type of data in organizations such as manager-employee or in file systems such as folder — subfolder structure.

So an item in the hierarchy is a 'parent' of all other items beneath it.

Recursive CTEs can be extremely useful to get insights from such type of data. They help you scan the entire hierarchical structure by referencing a CTE to itself.

An example would be best to understand this concept. Let's take an extremely common example of organizational hierarchy.

Suppose you have an `employee` table as shown below and would like to get a comma-separated list of all employees such that each list represents a path from manager to employee.

EmployeeID	EmployeeName	ManagerID
1	John	NULL
2	Jane	1
3	Bob	1
4	Alice	2
5	Charlie	2
6	David	3
7	Eva	3

Sample data | Image by Author

You can re-create this input data using the following query.

Please note that `analyticswithsuraj` is a schema name in MySQL Workbench. You can replace it with your schema.

```
DROP TABLE IF EXISTS analyticswithsuraj.employee;
CREATE TABLE analyticswithsuraj.employee (
    EmployeeID VARCHAR(10),
    EmployeeName VARCHAR(50),
    ManagerID VARCHAR(10)
);

-- Insert sample data
INSERT INTO analyticswithsuraj.employee VALUES (1, 'John', NULL);
INSERT INTO analyticswithsuraj.employee VALUES (2, 'Jane', 1);
INSERT INTO analyticswithsuraj.employee VALUES (3, 'Bob', 1);
INSERT INTO analyticswithsuraj.employee VALUES (4, 'Alice', 2);
INSERT INTO analyticswithsuraj.employee VALUES (5, 'Charlie', 2);
INSERT INTO analyticswithsuraj.employee VALUES (6, 'David', 3);
INSERT INTO analyticswithsuraj.employee VALUES (7, 'Eva', 3);
```

So getting back to the solution...Let me show you a complete solution and then you can go through its explanation following it.

```
WITH RECURSIVE RecursiveCTE AS (
    SELECT
        EmployeeID,
        EmployeeName,
        ManagerID,
        EmployeeName AS Path -- Initial path is just the employee's ID
    FROM
        analyticswithsuraj.employee
    WHERE
        ManagerID IS NULL -- Anchor part

    UNION ALL
```

```
    SELECT
        e.EmployeeID,
        e.EmployeeName,
        e.ManagerID,
        concat_ws(',', rc.Path, e.EmployeeName) AS Path
    FROM
        analyticswithsuraj.employee e
    JOIN
        RecursiveCTE rc ON e.ManagerID = rc.EmployeeID -- Recursive part
)
```

Note that in MySQL Workbench you need to write the keyword RECURSIVE before the CTE name to write a recursive CTE.

As you read in the basics of recursive CTE, your solution must contain two parts.

The Anchor part, which is the starting point which can be referenced in the recursive query will be a simple select statement as below.

```
    SELECT
        EmployeeID,
        EmployeeName,
        ManagerID,
        EmployeeName AS Path -- Initial path is just the employee's ID
    FROM
        analyticswithsuraj.employee
    WHERE
        ManagerID IS NULL -- Anchor part
```

Here, you will create an additional column `Path` to store the entire hierarchy of an employee.

This Anchor query will always contain only those employees who do not have any manager above them i.e. when ManagerID is NULL.

The second part of recursive CTE is the recursive part which selects the employee name and updates the value in the `Path` column to create a comma separated path as shown below.

```
    UNION ALL

    SELECT
        e.EmployeeID,
        e.EmployeeName,
        e.ManagerID,
        concat_ws(', ', rc.Path, e.EmployeeName) AS Path
    FROM
        analyticswithsuraj.employee e
    JOIN
        RecursiveCTE rc ON e.ManagerID = rc.EmployeeID -- Recursive part
```

As you see this recursive part queries data from the original employee table and joins it with the CTE within which it is written.

Whereas UNION ALL combines the results of the anchor and the recursive query.

Ultimately, you can query this recursive CTE with a simple SELECT statement.

```
SELECT
    EmployeeID,
    EmployeeName,
    ManagerID,
    Path
FROM
    RecursiveCTE;
```

EmployeeID	EmployeeName	ManagerID	Path
1	John	NULL	John
2	Jane	1	John, Jane
3	Bob	1	John, Bob
4	Alice	2	John, Jane, Alice
5	Charlie	2	John, Jane, Charlie
6	David	3	John, Bob, David
7	Eva	3	John, Bob, Eva

Employee hierarchy using recursive CTE | Image by Author

As a result, for each record, you can see a complete hierarchy of the employee. Here it shows that *John* is the ultimate manager who has no one above him, and *Jane* and *Bob* are his direct reporters.

In the input table, on 4th record you can see that *Jane* is the manager of *Alice* and *John* is the manager of *Jane*, so the entire hierarchy for *Alice* is '*John, Jane, Alice*'.

However, the data can be more complex than a simple hierarchy, such as a Facebook friends list or any social networking data. Let's see how recursive CTE can be used there.

Working with Network Data

Network data, as its name suggests is about a network of things, entities, and people. As you can see in the below picture, you can visualize network data as nodes (blocks) and edges (lines) connecting the nodes.

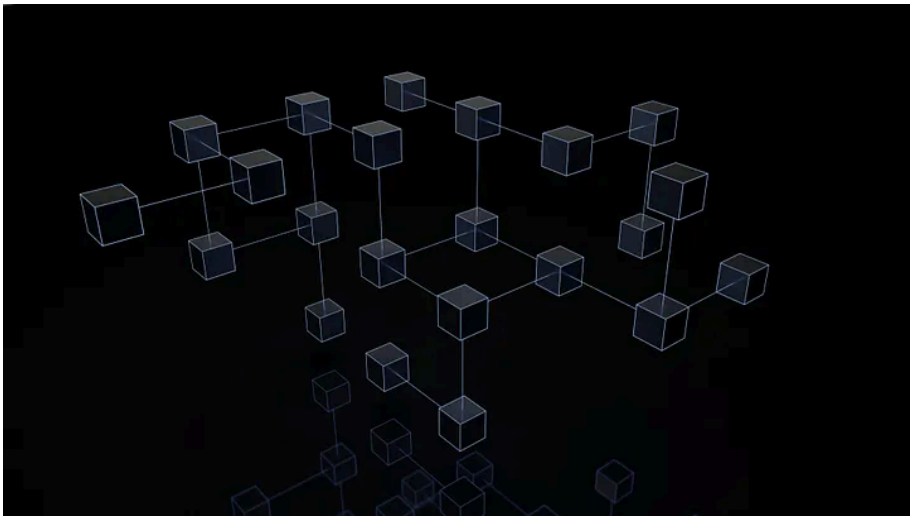


Photo by Shubham Dhage on Unsplash

A node essentially represents a user or an entity, and an Edge represents a relationship between them.

In simple terms, if there is an Edge (line) between two nodes, then the two nodes are directly connected to each other.

> *A classic example of network data is Facebook, where you are a 'friend' with me, I am a 'friend' with someone else. So we are i.e. you, me and someone else are nodes whereas our 'friendship' is the edge.*

Suppose you are analyzing a Facebook network of 11 friends, as shown below in the table `network_connections`. And you want to know all the people who are in the *Paolo's* network i.e. people who are *Paolo's* friends and friends of his friends.

source_node	target_node
Paolo	David
Paolo	Anna
David	Mark
Anna	Peter
Samar	Patrik
Mark	Vivan
Patrik	Maya
Julia	Robert

Sample Facebook network data | Image by Author

You can create this table with the following query.

```
DROP TABLE IF EXISTS alldata.network_connections;
CREATE TABLE alldata.network_connections (
    source_node VARCHAR(50),
    target_node VARCHAR(50)
);

INSERT INTO alldata.network_connections (source_node, target_node) VALUES
('Paolo', 'David'),
('Paolo', 'Anna'),
('David', 'Mark'),
('Anna', 'Peter'),
('Samar', 'Patrik'),
('Mark', 'Vivan'),
('Patrik', 'Maya'),
('Julia', 'Robert');
```

Please note that `alldata` is a schema name in MySQL Workbench. You can replace it with your schema.

As I already explained the anchor part and recursive part in the previous example of hierarchical data, let's directly go to the solution here.

```
WITH RECURSIVE NetworkCTE AS (
    -- Anchor part to select the starting node
    SELECT source_node,
           target_node
    FROM network_connections
    WHERE source_node = 'Paolo' -- Change this to get someone else's network
```

```
    UNION ALL

    -- Recursive part to select connected nodes
    SELECT nc.source_node,
           nc.target_node
    FROM network_connections nc
    JOIN NetworkCTE n ON nc.source_node = n.target_node
)

SELECT * FROM NetworkCTE;
```

As the code explains, the anchor part first pulls all the records, where *Paolo* is source node i.e. all the direct friends of *Paolo*. Whereas the recursive part gets all the friends of friends of *Paolo*.

So, ultimately you can see *Paolo's* network as following.

source_node	target_node
Paolo	David
Paolo	Anna
David	Mark
Anna	Peter
Mark	Vivan

Social network analysis | Image by Author

Some of the other use-cases of network data analysis can be in the supply chain industry to analyze dependencies between different entities or into financial transactions analysis to track movement of funds through different accounts.

If you come across some more such examples, where recursive CTE can be used efficiently, don't forget to mention it in comments.

I hope you found this article useful and informative!

The Common Table Expressions (CTEs) are widely used in SQL and recursive CTEs are special case. In this quick read, you explored how to work on hierarchical as well as network data using recursive CTE.

Here you learned with real-world examples, how recursive CTEs can be game-changer in SQL in specific scenarios. If you have any other insights for the same, feel free to share it in comments!

Your insights matter!

💡 Be sure to **Follow Me** and **Sign-up to my Email list** to never miss another article on data science, SQL, Python and on job search tricks.

Thank you for reading!

Data | Data Science | Programming | Sql | Editors Pick

More from the list: "SQL"

Curated by @reenum

SQL Fundamentals
Advanced SQL Techniques for Beginners
✦ · 2 min read · Jan 27, 2024

SQL Fundam... in DevOps...
Mastering SQL Data Cleaning
✦ · 2 min read · Jan 27, 2024

SQL Fundam... in DevOps...
From Basic to Intermediate SQL: 12...
✦ · 6 min read · Dec 3, 2023

5 R You
✦

View list

Written by Suraj Gurav

Follow

3K Followers · Writer for Learning SQL

Analytics professional and writer. I write about Data Science, Python, SQL & interviews.
Join Medium today to get all my articles: https://tinyurl.com/3fehn8pw

More from Suraj Gurav and Learning SQL

Suraj Gurav in Towards Data Science

Python Pandas Tricks: 3 Best Methods To Join Datasets

Master Python merge, concat and join in your coffee time!

✦ · 7 min read · May 17, 2022

160 1

Sarang S. Babu in Learning SQL

12 Tips for Optimizing SQL Queries for Faster Performance

Ways to Optimize SQL Queries

8 min read · Mar 6, 2023

120 2

Zach Quinn in Learning SQL

How I Reduced My Query's Run Time From 30 Min. To 30 Sec. In 1...

The query optimization steps a senior data engineer took to reduce the process time of ...

✦ · 6 min read · Mar 13, 2024

413 16

Suraj Gurav in Towards Data Science

3 Easy Ways To Compare Two Pandas DataFrames

Quickly learn how to find the common and uncommon rows between the two pandas...

✦ · 6 min read · Aug 8, 2023

248 2

See all from Suraj Gurav | See all from Learning SQL

Recommended from Medium

Jake Page

The guide to Git I never had.

Doctors have stethoscopes.

13 min read · Apr 11, 2024

2.1K 22

Douenergy in In the Pipeline

From Zero to dbt: How to Analyze and Build Data Models from...

Part 1: Analyze the 30GB json dataset with DuckDb and jq, then convert to Parquet to...

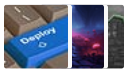
10 min read · Apr 11, 2024

198 2

Lists

General Coding Knowledge

20 stories · 1130 saves

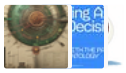


Predictive Modeling w/ Python

20 stories · 1113 saves

Coding & Development

11 stories · 572 saves



data science and AI

40 stories · 131 saves

Scott Teal in Snowflake

How to ETL from web APIs into an open data lakehouse with Python,...

Storing raw data from APIs is a common use case for a data lake, which is essentially...

13 min read · Apr 13, 2024

76 1

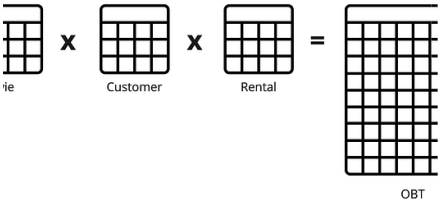
Kingsley Okoye

EXTRACT 200 TABLES FROM ANY SQL DATABASE WITH JUST 20...

Have you ever been in a situation where you were instructed to explore a database and y...

6 min read · Feb 25, 2024

31



Hubert Dulay

One Big Table (OBT) vs Star Schema

Two methodologies stand out when building an analytical data model: One Big Table (OB...

6 min read · Apr 3, 2024

12 1

Hugo Lu

Data Orchestration for Data Products

Open Source Tools, All-in-One Control Planes, & Impact on Data Product Velocity

6 min read · Apr 14, 2024

172 1

See more recommendations