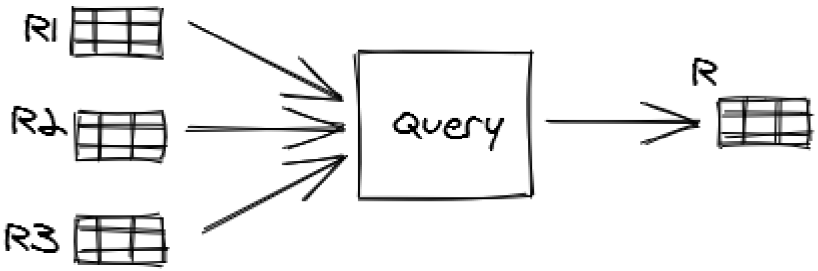
Published in The Startup

Denis Lukichev Follow
Nov 22, 2020 · 5 min read · Listen

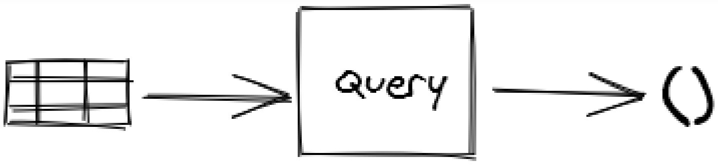
Recursion in SQL Explained Visually

Recursion in SQL? But why? Oh, there are many uses for that. It's common to store hierarchical data in SQL and recursive queries are a convenient way to extract information from such graphs. Organizational structure, application menu structure, a set of tasks with sub-tasks in the project, links between web pages, breakdown of an equipment module into parts and sub-parts are examples of the hierarchical data. The post will not go into great details of those many use cases rather look at two toy examples to understand the concept - the simplest possible case of recursion on numbers and querying data from the family tree.

Let's think about queries as a function. In a sense that a function takes an input and produces an output. Queries operate on relations or one could say tables. We will denote those as Rn. Here is a picture of a query. It takes three relations R1, R2, R3 and produces an output R. Simple enough.



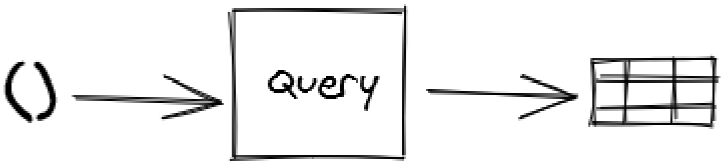
Caption: A picture representation of how a query works.



A visual representation of a query taking something and producing nothing.

SQL example: `SELECT <something> FROM R1 WHERE 1 = 2`

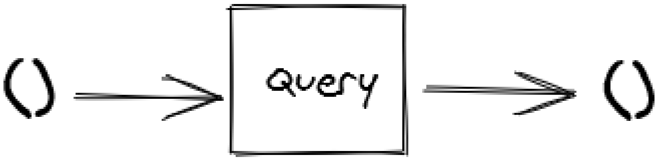
Take nothing and produce something:



A visual representation of a query taking nothing and producing something.

SQL example: `SELECT 1`

Or take nothing and produce nothing



A visual representation of a query taking nothing and producing nothing.

```
SELECT 1 WHERE 1 = 2
```

Recursion is achieved by `WITH` statement, in SQL jargon called Common Table Expression (CTE). It allows to name the result and reference it within other queries sometime later.

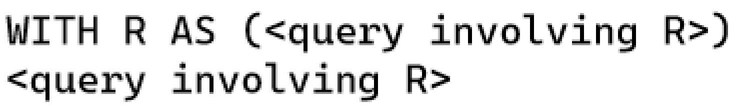
Naming the result and referencing it within other queries.

Here is a sample.

```
WITH R AS (SELECT 1 AS n)
SELECT n + 1 FROM R
```

Query `(SELECT 1 AS n)` now have a name — `R`. We refer to that name in `SELECT n + 1 FROM R`. Here `R` is a single row, single column table containing number 1. The result of the whole expression is number 2.

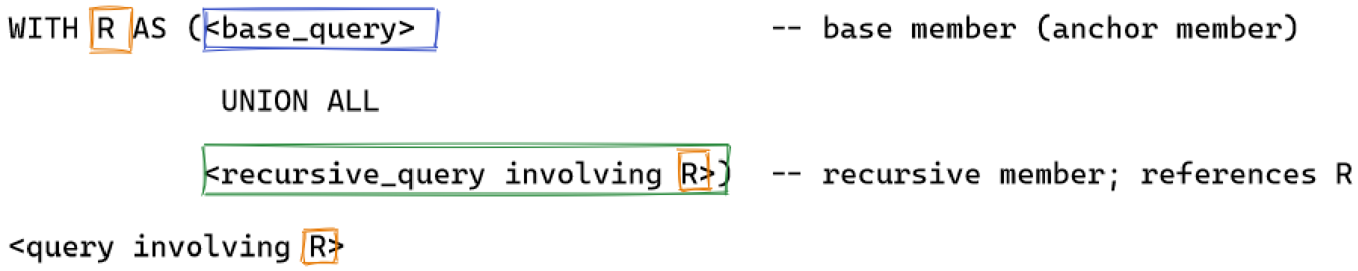
The recursive version of `WITH` statement references to itself while computing output.



```
WITH R AS (<query involving R>)
<query involving R>
```

Using the recursive with statement.

For the recursion to work we need to start with something and decide when the recursion should stop. To achieve this, usually recursive `with` statement has following form.

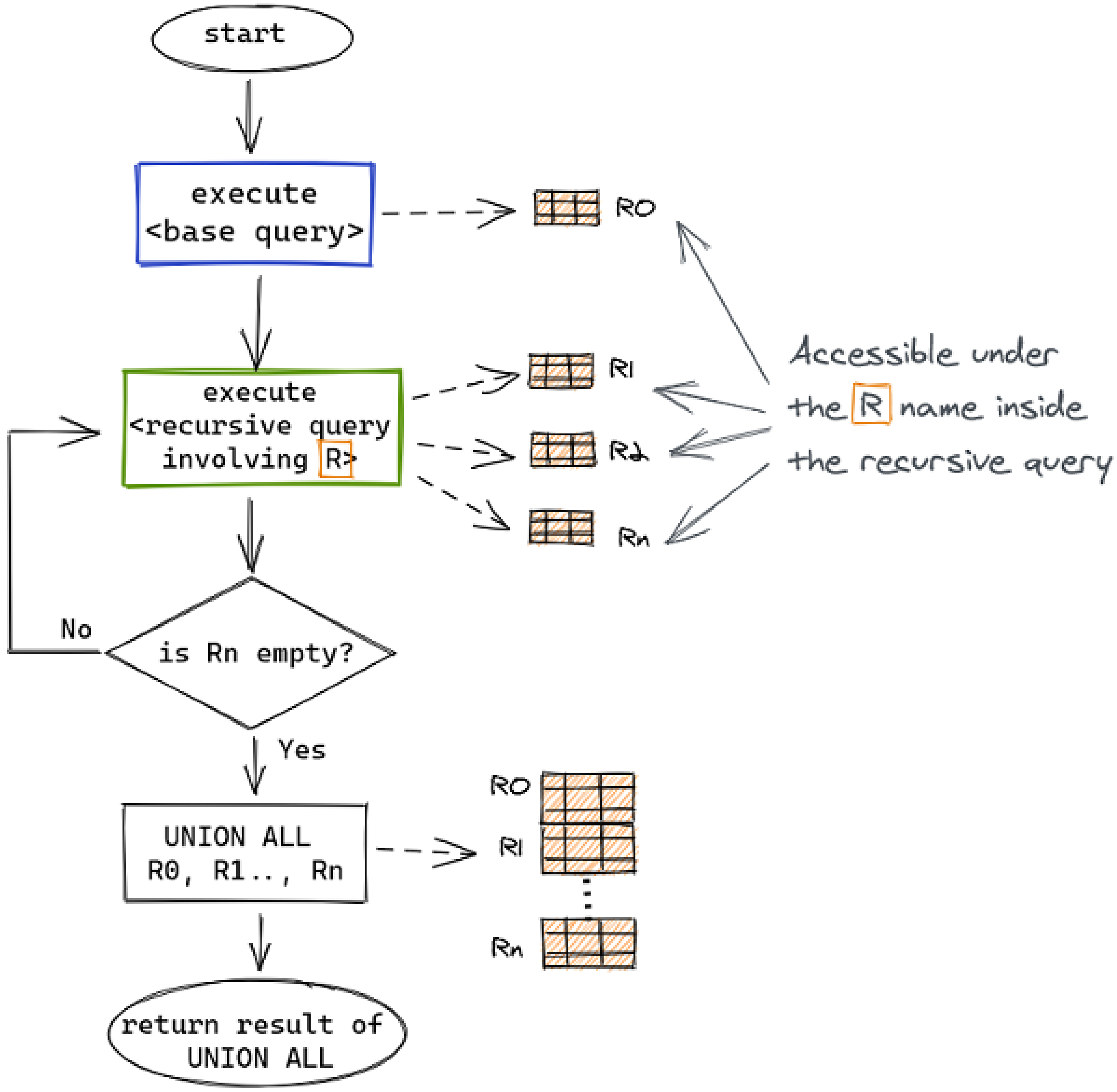


```
WITH R AS (<base_query>                     -- base member (anchor member)

            UNION ALL

           <recursive_query involving R>)   -- recursive member; references R

<query involving R>
```

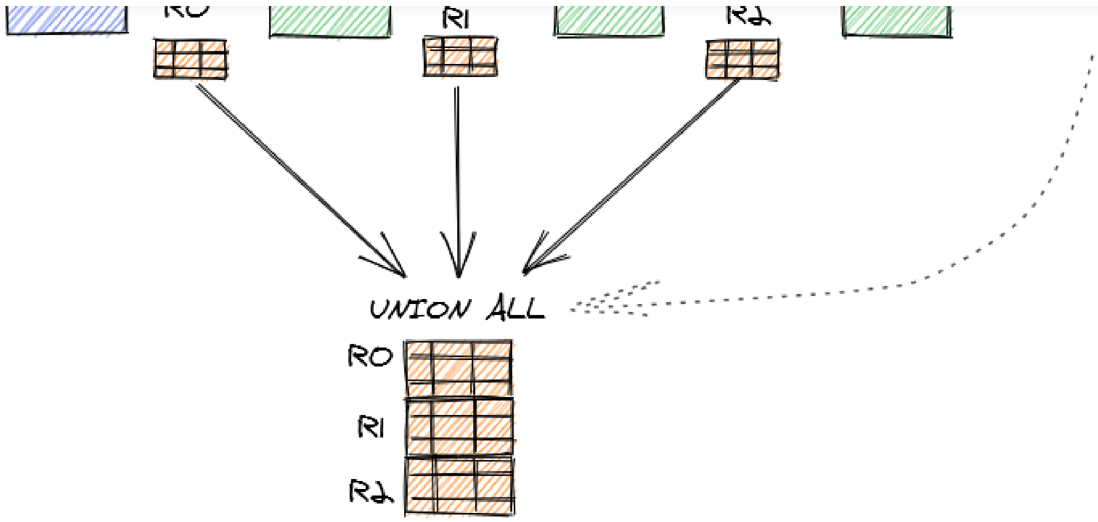
Results of running the recursion.

Important to note that base query doesn't involve R, but recursive query references R. From the first look it seems like infinite loop, to compute R we need compute R. But

Here’s what is happening: base query executed first, taking whatever it needs to compute the result R0. Second recursive query is executed taking R0 as input, that is R references R0 in the recursive query when first executed. Recursive query produces the result R1 and that is what R will reference to at the next invocation. And so on until recursive query returns empty result. At that point all intermediate results are combined together.

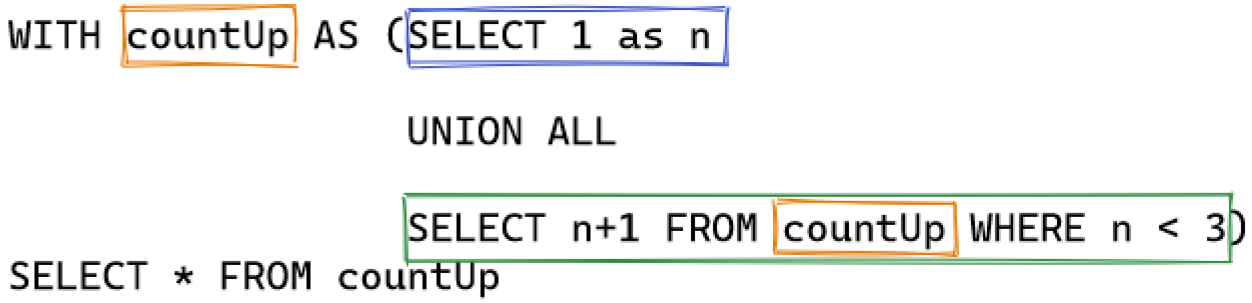

start
execute <base query>
R0
execute <recursive query involving R>
R1
R2
Rn
Accessible under the R name inside the recursive query
No
is Rn empty?
Yes
UNION ALL R0, R1.., Rn
R0
R1
Rn
return result of UNION ALL

Recursive query execution algorithm flow chart



Recursive query execution sequence

Quite abstract now. Lets take a concrete example, count until 3.



```
WITH countUp AS (SELECT 1 as n

                 UNION ALL

                 SELECT n+1 FROM countUp WHERE n < 3)
SELECT * FROM countUp
```

Running a recursive statement with count until three.

Base query returns number `1` , recursive query takes it under the `countUp` name and produces number `2` , which is the input for the next recursive call. When recursive query returns empty table (`n >= 3`), the results from the calls are stacked together.

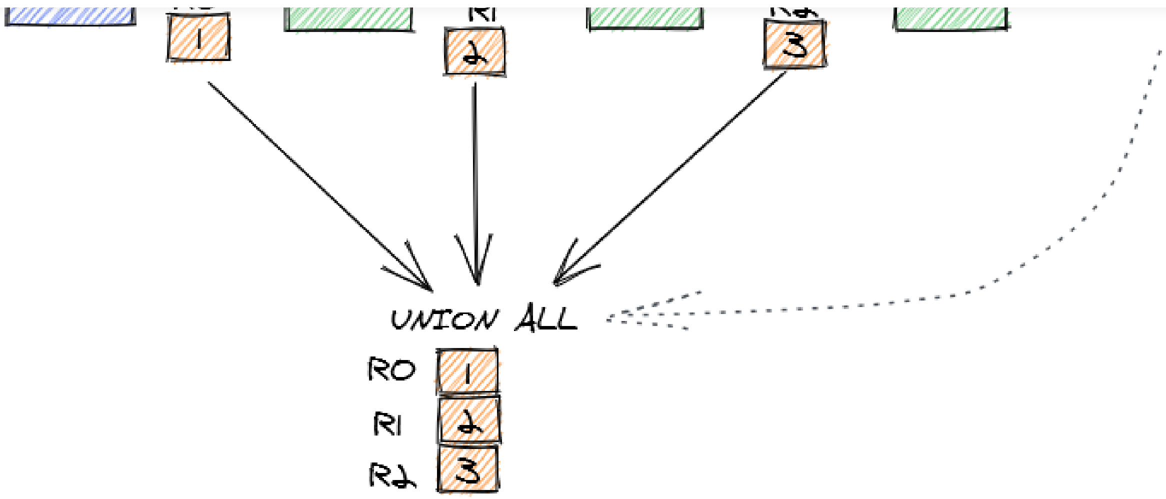
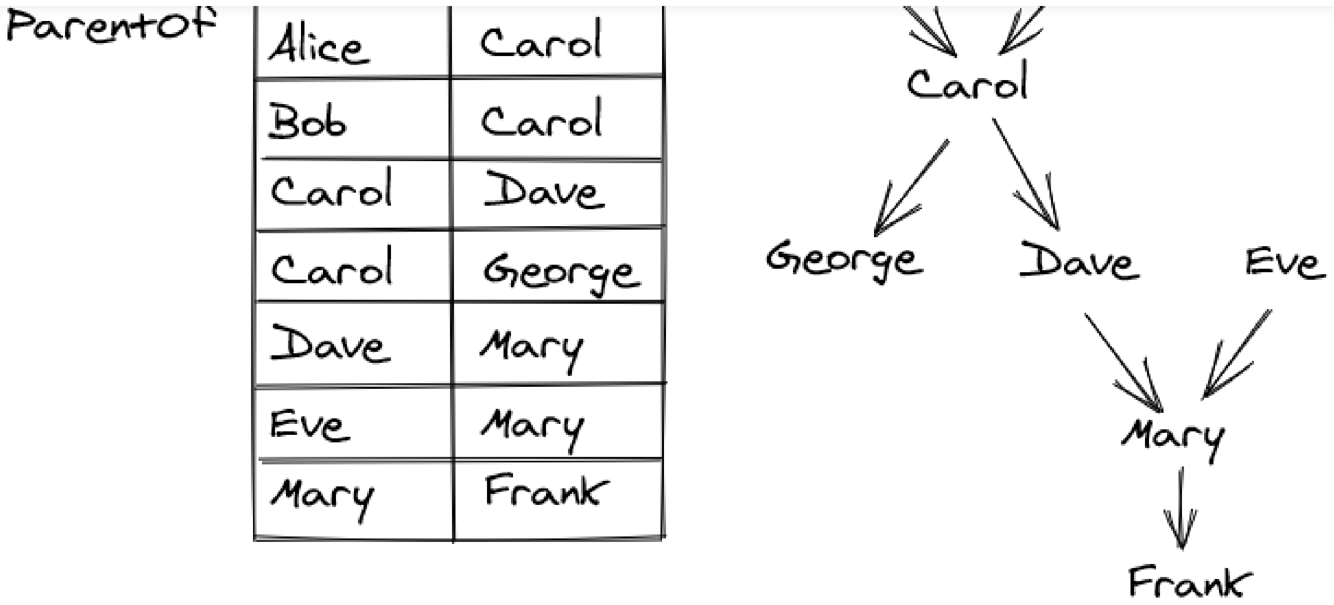


Illustration of the results from the call stacked together.

Watch out, counting up like that can only go that far. There is a limit for recursion. It defaults to 100, but could be extended with `MAXRECURSION` option (MS SQL Server specific). Practically, it could be a bad idea to crank recursion limit up. Graphs might have cycles and limited recursion depth can be a good defense mechanism to stop poorly behaving query.

```
OPTION (MAXRECURSION 200)
```

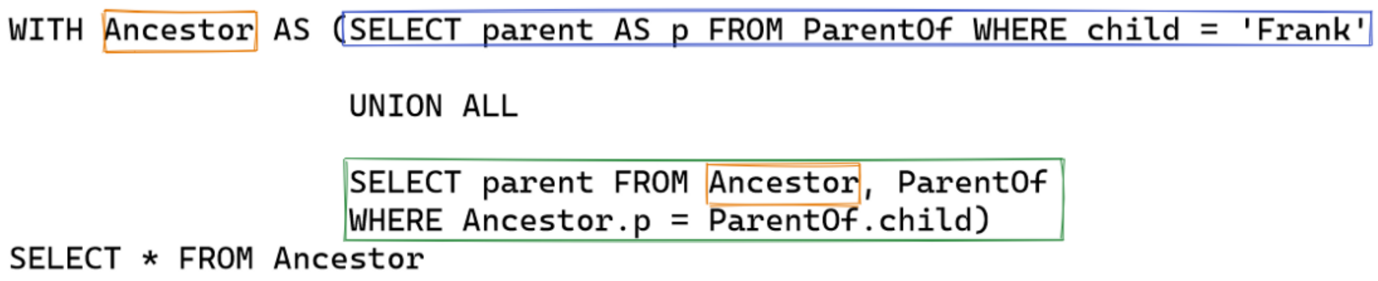
Here’s another example, find ancestors of a person:



ParentOf

Alice	Carol
Bob	Carol
Carol	Dave
Carol	George
Dave	Mary
Eve	Mary
Mary	Frank

Using recursion to find the ancestors of a person.



```
WITH Ancestor AS (SELECT parent AS p FROM ParentOf WHERE child = 'Frank'

                  UNION ALL

                  SELECT parent FROM Ancestor, ParentOf
                  WHERE Ancestor.p = ParentOf.child)
SELECT * FROM Ancestor
```

Code statement to use recursion to find the ancestors of a person.

Base query finds Frank’s parent — Mary, recursive query takes this result under the `Ancestor` name and finds parents of Mary, which are Dave and Eve and this continues until we can’t find any parents anymore.

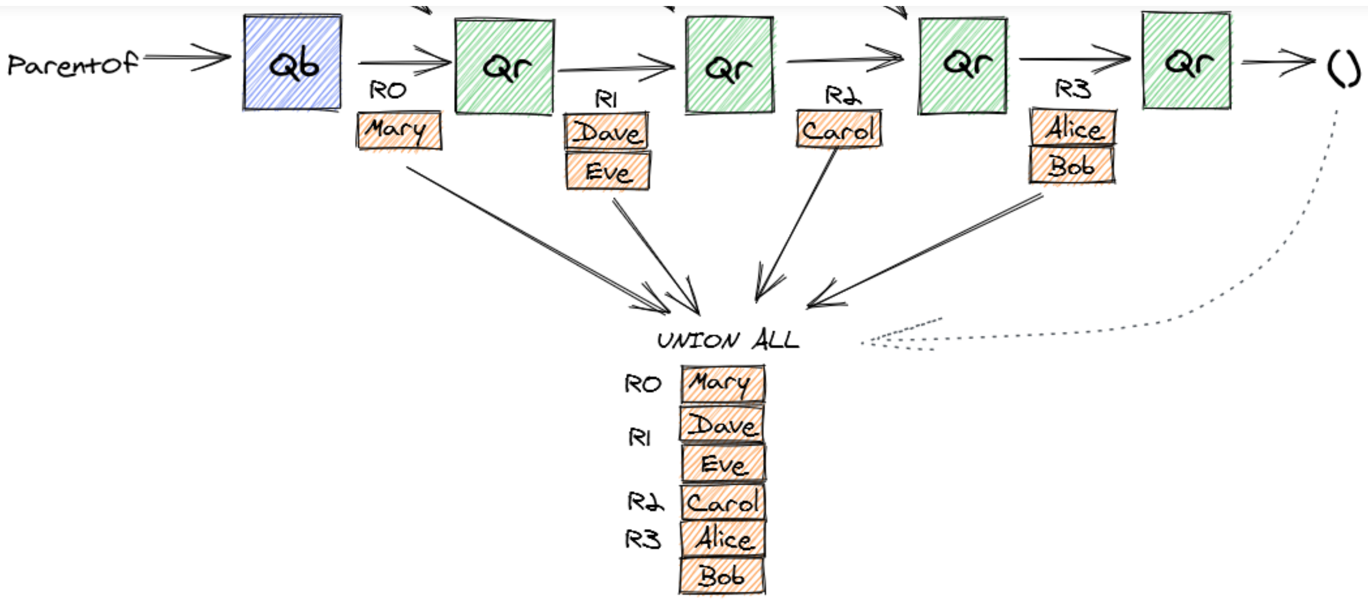
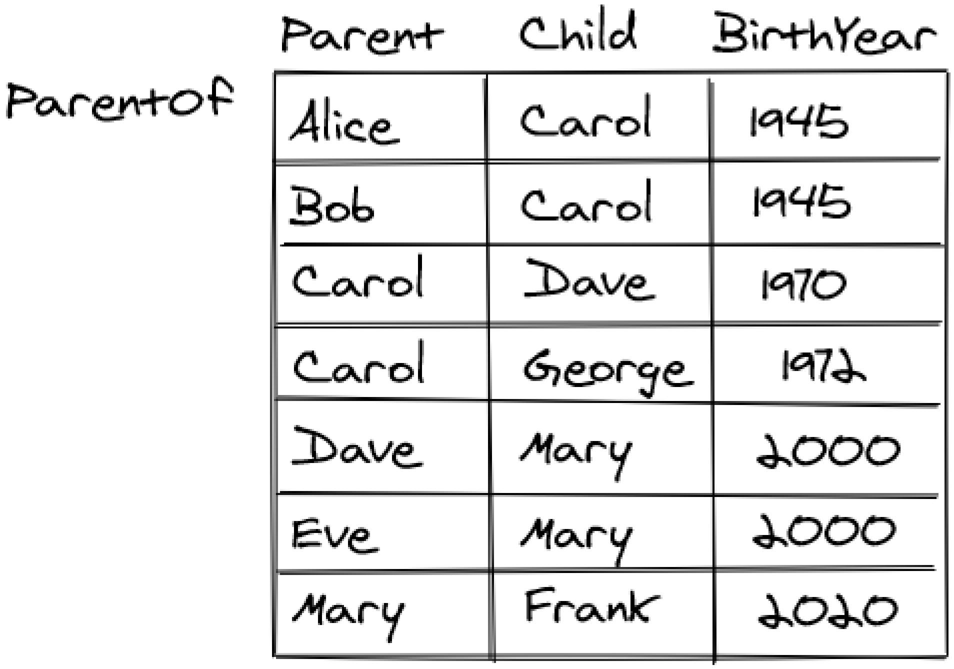


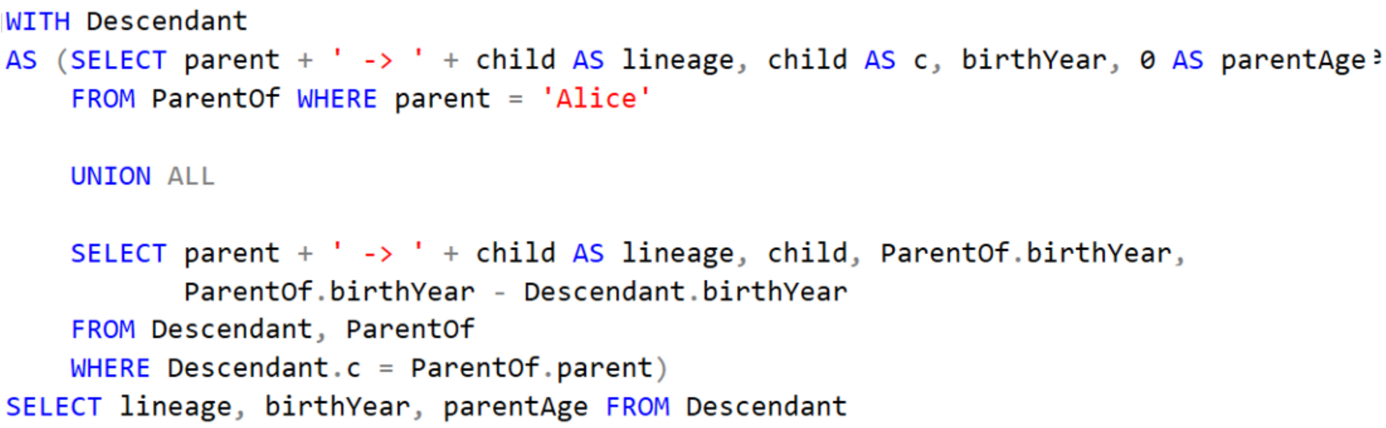
Illustration of the results from the recursion to find the ancestors of a person.


Parentof

Parent	Child	BirthYear
Alice	Carol	1945
Bob	Carol	1945
Carol	Dave	1970
Carol	George	1972
Dave	Mary	2000
Eve	Mary	2000
Mary	Frank	2020

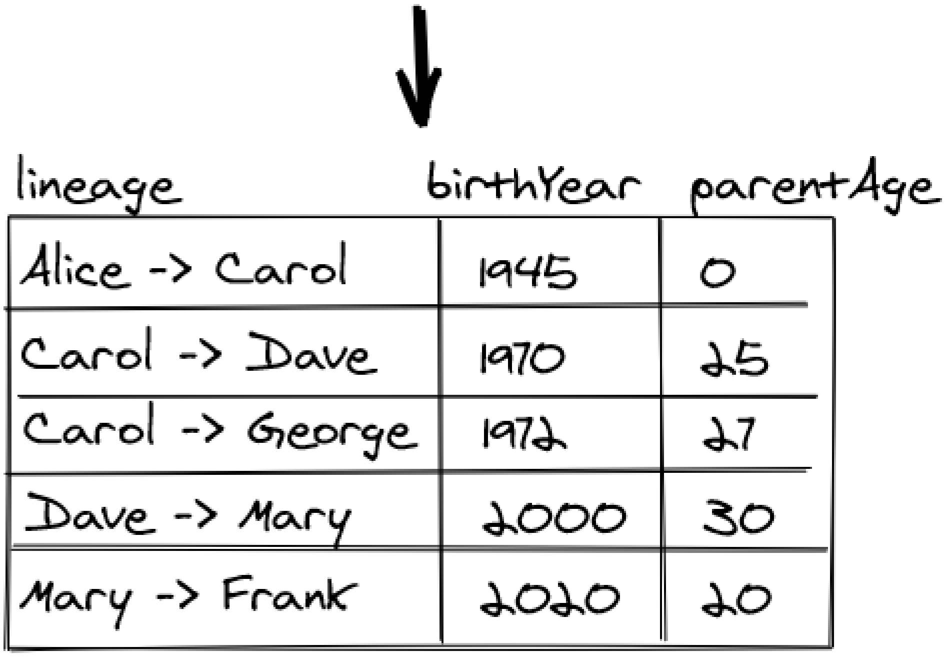
A table that includes the birth year to find the parents of a person.

Now this tree traversal query could be the basis to augment the query with some other information of interest. For example, having a birth year in the table we can calculate how old the parent was when the child was born. Next query do exactly that, together



```
WITH Descendant
AS (SELECT parent + ' -> ' + child AS lineage, child AS c, birthYear, 0 AS parentAge
    FROM ParentOf WHERE parent = 'Alice'

    UNION ALL

    SELECT parent + ' -> ' + child AS lineage, child, ParentOf.birthYear,
           ParentOf.birthYear - Descendant.birthYear
    FROM Descendant, ParentOf
    WHERE Descendant.c = ParentOf.parent)
SELECT lineage, birthYear, parentAge FROM Descendant
```

Running a recursion to find the birth year of a person and their ancestors.



lineage	birthYear	parentAge
Alice -> Carol	1945	0
Carol -> Dave	1970	25
Carol -> George	1972	27
Dave -> Mary	2000	30
Mary -> Frank	2020	20

Table representing the results from the recursion to find the birth year and ancestors of a person.

Take away — recursive query references the result of base query or previous invocation of recursive query. Chain stops when recursive query returns empty table.

[UPDATE] Post updated with comments from kagato87 and GuybrushFourpwood reddit users.

[NOTE] Code samples are for MS-SQL. Other DBMS could have slightly different syntax.

SQL Server solution to simple recursive function

Asked 8 years, 7 months ago Modified 8 years, 7 months ago Viewed 1k times

0

I am looking for a SQL Server solution for a simple recursive formula. In the example, X is my column of numbers and Y is the column I am trying to create with a SQL Query.

I have a list of numbers, denoted X, and wish to produce a special kind of running sum that is not allowed to go less than 0, denoted Y.

Base Case

Y_1 = MAX(X_1,0)

Recursive Rule

Y_i = MAX(X_i+Y_{i-1},0)

EXAMPLE:

```
id      X(Input)    Y(Output)
1       15          15
2       -87         0
3       26          26
4       -87         0
5       4           4
6       -19         0
7       34          34
8       -4          30
9       40          70
10      -14         56
```

sql sql-server recursion recursive-query

Share Improve this question Follow

edited Mar 18, 2014 at 6:03
marc_s
714k 171 1315
1434

asked Mar 18, 2014 at 0:49
user3431083
404 5 19

Do you have a column that specifies the ordering of the rows? SQL tables are inherently unordered, so if this is your entire table, you cannot do what you want. – Gordon Linoff Mar 18, 2014 at 0:52

1

Assuming you have an `id` column that specifies the ordering, I am pretty sure you have to do this with a recursive CTE. The problem is that the "set negative numbers to zero" complicates the situation.

Let me assume that the `id` identifies the ordering.

```
with t as (
      select t.*, row_number() over (order by id) as seqnum
      from table t
     ),
     cte as (
      select X,
             (case when X < 0 then 0 else X end) as Y
      from t
      where id = 1
      union all
      select tnext.X,
             (case when tnext.X + cte.Y < 0 then 0 else tnext.X + cte.Y end) as Y
      from cte join
           t tnext
           on t.id + 1 = tnext.id
     )
select *
from cte;
```

Share Improve this answer Follow

answered Mar 18, 2014 at 0:55

Gordon Linoff

1.2m 53 599 743

I successfully implemented this type of solution, but was wondering if it is flexible enough to handle the extension of the problem wherein I add a varchar column "product" and need to keep this same special running positive sum unique to each "product". When ordered by id, the products would occur multiple times and appear randomly in the list. – user3431083 Mar 18, 2014 at 22:30

0

Using a cursor and a table variable to catch the calculated values might be good for performance.

```
declare @T table
(
  id int,
  X int,
  Y int
);

declare @id int;
declare @X int;
declare @Y int;

set @Y = 0;
```

```
  order by T.id;

open C;

fetch next from C into @id, @X;
while @@fetch_status = 0
begin
  set @Y = case when @X + @Y < 0 then 0 else @X + @Y end;
  insert into @T(id, X, Y) values (@id, @X, @Y);
  fetch next from C into @id, @X;
end

close C;
deallocate C;

select T.id, T.X, T.Y
from @T as T
order by T.id;
```

SQL Fiddle

Have a look at Best approaches for running totals by Aaron Bertrand

Share Improve this answer Follow

edited May 23, 2017 at 12:11
Community Bot
1 1

answered Mar 18, 2014 at 6:45
Mikael Eriksson
134k 22 202 274