EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Advanced SQL
Winter Term 2025/2026
Prof. Torsten Grust, Tim Fischer, Björn Bamberg
WSI — Database Systems Research Group

Assignment 10

Hand in this assignment until **Friday, January 16th 2026, 12:00 pm** at the latest.

> 🤔 Running out of ideas?
> Are you hitting a roadblock? Are some of the exercises unclear? Do you just need that one hint to get the ball rolling? Refer to the `#forum` 🌐 channel on our Discord server—maybe you'll find just the help you need.

> Ⓔxam-style Exercises
> Exercises marked with Ⓔ are similar in style to those you will find in the exam. You can use these to hone your expectations and gauge your skills.

Task 1: Tree Labels

RECURSIVE CTES TREES LISTS

Table ▦ `trees` represents a collection of trees using the list representation we've previously detailed and played around with in the slides of Chapter 3 (Tree Encoding) and Assignment 6 Exercise 1, respectively.

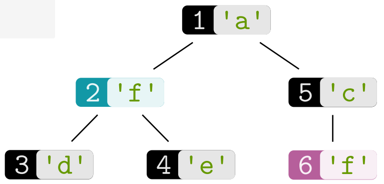
▦ trees

tree	parents	labels
❶	[NULL, 1, 2, 2, 1, 5]	['a', 'b', 'd', 'e', 'c', 'f']
❷	[4, 1, 1, 6, 5, NULL, 6]	['d', 'f', 'a', 'b', 'r', 'g', 'c']
❸	[NULL, 1, NULL, 1, 3]	['a', 'b', 'd', 'c', 'e']
❹	[NULL, 1, 2, 2, 1, 5]	['a', 'f', 'd', 'e', 'c', 'f']

We've also touched on how we can write our `WITH RECURSIVE` based tree-traversals over this tree represenation in the lecture material of Chapter 6 — see 📄 `path-to-root.sql`. There you will find the following query, which finds all nodes which lie on paths from the node with label `'f'` to the root node.

```sql
WITH RECURSIVE paths(tree, pos, node) AS (
  SELECT t.tree,
         0 AS pos,
         list_position(t.labels, 'f') AS node
  FROM   trees AS t
  WHERE  'f' = ANY(t.labels)
    UNION
  SELECT t.tree,
         p.pos + 1 AS pos,
         t.parents[p.node] AS node
  FROM   paths AS p, trees AS t
  WHERE  p.tree = t.tree
  AND    p.node IS NOT NULL
)
SELECT p.tree,
       p.pos,
       p.node
FROM   paths AS p
WHERE  p.node IS NOT NULL
ORDER  BY p.tree, p.pos;
```

Importantly, this query expects node labels to be unique within a tree, which presents a problem for tree ❹ in ▦ `trees` — depicted to the right. In this tree the start label `'f'` for this search is duplicated!



Adapt the SQL query above such that *multiple nodes of the same tree* may have the same label. Then, **extend its output by a column** `path` which *uniquely identifies the path* a given node has been found on.

tree	path	pos	node
…	…	…	…
❹	1	0	2
❹	1	1	1
❹	2	0	6
❹	2	1	5
❹	2	2	1

> 😵‍💫 Instructions unclear?
> Your query's output should contain two paths for tree ❹, one starting at the node 2 (`path` = 1) and the other starting at node 6 (`path` = 2).

> 💡 Blast from the Past
> You can make use of the `list_positions` macro from back in Assignment 6 here!

Task 2: Perrin Numbers Ⓔ

RECURSIVE CTES

Write a SQL query which, starting with 0, calculates the first $i \in \mathbb{N}$ values of the Perrin sequence 🌐. **Use a macro to define the sequence limit** i. For $i = 7$, your query should produce the result to the right.

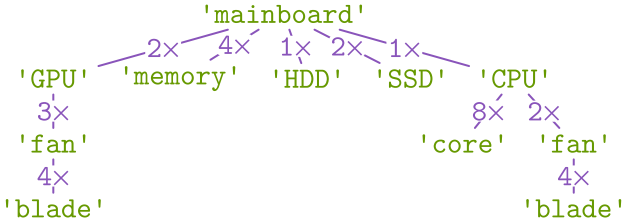
n	per
0	3
1	0
2	2
3	3
4	2
5	5
6	5
7	7

> 🤨 Perrin *what now*?
>
> The Perrin sequence is a *recursive integer sequence*, like the Fibonacci numbers. That is, a sequence of integers that is defined through a combination of a few base cases and a recursive formula for the rest. This sequence is defined by the following equations to the right.
>
> $$P(0) = 3$$
> $$P(1) = 0$$
> $$P(2) = 2$$
> $$P(n) = P(n-2) + P(n-3) \text{ for } n > 2$$

Task 3: Bill of Materials Ⓔ

RECURSIVE CTES TREE TRAVERSAL

Recursive queries shine whenever they are used to process hierarchical data. Consider the tables ▦ products and ▦ parts, these two tables represent a collection of individual products and to what quantity they are "part of" one another. If you squint your eyes a bit, you will find that these to tables encode a tree where ▦ products holds all nodes, *i.e.*, their labels, and ▦ parts the dependencies, *i.e.*, edges and labels thereof between them.



▦ products

id	name
1	'mainboard'
2	'GPU'
3	'memory'
4	'HDD'
5	'SSD'
6	'CPU'
7	'core'
8	'fan'
9	'blade'

▦ parts

prod	part	quant
1	2	2
1	3	4
1	4	1
1	5	2
1	6	1
2	8	3
4	7	8
4	8	2
8	9	4

Write a SQL query that computes the *bill of materials* for a *single mainboard*, *i.e.*, lists all parts and their overall quantities that are contained within a single mainboard (*i.e.*, the product with `id` = 1). The actual order of the items "on the bill" is irrelevant to the correctness of your results!

> 📋 The actual bill…
>
> The bill of materials for the example above should list the quantities depicted in ▦ bom to the right. Though the order is irrelevant, to make things easier to compare we have sorted ▦ bom by the names of the parts. 😉

▦ bom

name	quant
'CPU'	1
'GPU'	2
'HDD'	1
'SSD'	2
'blade'	32
'core'	8
'fan'	8
'memory'	4