Advanced SQL
Sommersemester 2022
Prof. Torsten Grust, Christian Duta
WSI — Database Systems Research Group

Assignment 9 (08.07.2022)

Handin until: 15.07.2022, 09:00

Important: All the queries you write for this assignment require **recursive queries**.

1. [10 Points] **Perrin Numbers**

   Write a SQL query which calculates each value of the Perrin sequence[1] up to any value $i \in \mathbb{N}$ starting with 0. For example, for $i = 7$, your query produces:

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

2. [10 Points] **Bill of Materials**

   Recursive queries shine whenever they are used to process hierarchical data. Consider the tables `products` and `parts`:

```sql
CREATE TABLE products (
  id   int  PRIMARY KEY,
  name text NOT NULL);
```

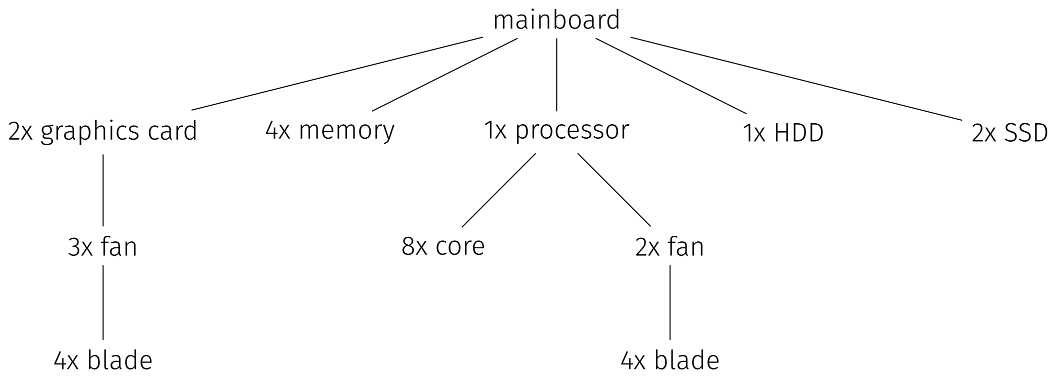
```sql
CREATE TABLE parts (
  part     int NOT NULL REFERENCES products(id),
  sub      int NOT NULL REFERENCES products(id),
  quantity int CHECK (quantity > 0));
```

   If a product consists of other products, it has an entry in table `parts`. For example, consider a mainboard:

```sql
INSERT INTO products(id, name) VALUES
(1, 'mainboard'),
(2, 'graphics Card'),
(3, 'memory'),
(4, 'processor'),
(5, 'HDD'),
(6, 'SSD'),
(7, 'core'),
(8, 'fan'),
(9, 'blade');
```

```sql
INSERT INTO parts(part, sub, quantity) VALUES
(1,2,2),(1,3,4),(1,4,1),(1,5,1),(1,6,2),
(2,8,3),
(4,7,8),(4,8,2),
(8,9,4);
```

[1] https://en.wikipedia.org/wiki/Perrin_number



Construct a SQL query that lists all parts (and their overall quantity) contained in a mainboard (i.e., the product with `id = 1`). For the example above, we expect the following result:

name	total quantity
graphics card	2
memory	4
HDD	1
fan	8
processor	1
SSD	2
blade	32
core	8

3. [10 Points] **Tree Labels**

Table **trees** represents a number of trees as previously defined in the slides of Chapter 03 (Tree Encoding) and Assignment 05 Exercise 03.

```
CREATE TABLE trees (tree    int PRIMARY KEY,
                    parents int[] NOT NULL,
                    labels  text[] NOT NULL);
```

This query (also contained in SQL file **path-to-root.sql** distributed with the material of Chapter 6) finds all nodes on a path from a node with label **f** to the root node. This query expects each tree to have unique node labels.

```
-- Which nodes are on the path from node labeled 'f' to the
-- root and on which position on the path are these nodes?
WITH RECURSIVE
paths(tree, pos, node) AS (
  SELECT t.tree, 0 AS pos, array_position(t.labels, 'f') AS node
  FROM   trees AS t
    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
  -- avoid infinite recursion once we reach the root
)
SELECT p.tree, p.pos, p.node
FROM   paths AS p
WHERE  p.node IS NOT NULL
ORDER BY p.tree, p.pos;
```

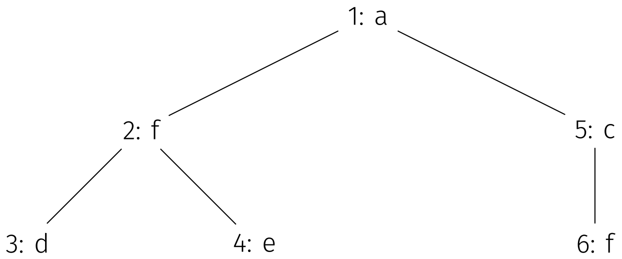
Your task is to adapt the query such that *multiple nodes of the same tree* may carry the same label (e.g., label **f** may occur more than once). Then, extend the output with a new column **path** which uniquely identifies on which of the (possibly many) paths a node has been found.

Hint: Consider using **array_positions(...)**[2].

[2] https://www.postgresql.org/docs/current/static/arrays.html

Example: Consider table **trees** with a simple tree:

```
INSERT INTO trees(tree, parents, labels)
  VALUES
  (1, ARRAY[NULL,1,2,2,1,5],
      ARRAY['a','f','d','e','c','f']);
```



Your adapted query produces the following result:

tree	id	pos	node
1	1	0	2
1	1	1	1
1	2	0	6
1	2	1	5
1	2	2	1