Assignment 2 (06.05.2022)

Handin until: 13.05.2022, 09:00

1. [10 Points] **Measurements**

   You are handed a note by a physicist, who kindly asks you to take a look at the measurements $m(t)$ (taken at time t) they intend to store inside a RDBMS of your choice (PostgreSQL 14, of course). They explain that at each timestamp t, one or more measurements $m(t)$ have been recorded.

   Measurements Note

t	$m(t)$
1.0	1.0, 3.0, 5.0, 5.0
2.5	0.8, 2.0
4.0	0.5
5.5	3.0
8.0	2.0, 6.0, 8.0
10.5	1.0, 3.0, 8.0

   Write SQL queries for the following tasks. Assume that, in the future, more measurements may be recorded.

   (a) Create a table `measurements(t numeric, m numeric)` based on the note given to you by the physicist. Next, define an artificial `PRIMARY KEY` by adding a new column `id` and populate the table with the measurements of the note.

   (b) Compute a one-column table with the global maximum of the measurements $m(t)$.

   (c) Compute a two-column table that lists each time t and the average of its measurements $m(t)$.

   (d) Compute a two-column table that lists the average of all measurements $m(t)$ in each timeframe $[0.0 - 5.0), [5.0 - 10.0), [10.0 - 15.0), \ldots$

   (e) Find all times t (there may be more than one) at which the global maximum was recorded. The result is a two-column table that lists time t and the global maximum.

2. [20 Points] **Limited-depth trees and GROUPING SETS**

Consider an **arbitrary** binary tree with depth of up to four, where each level is assigned a letter `A`, `B`, `C` and `D`. Leaf nodes hold integer labels. For a more visual representation, see Figure 1.

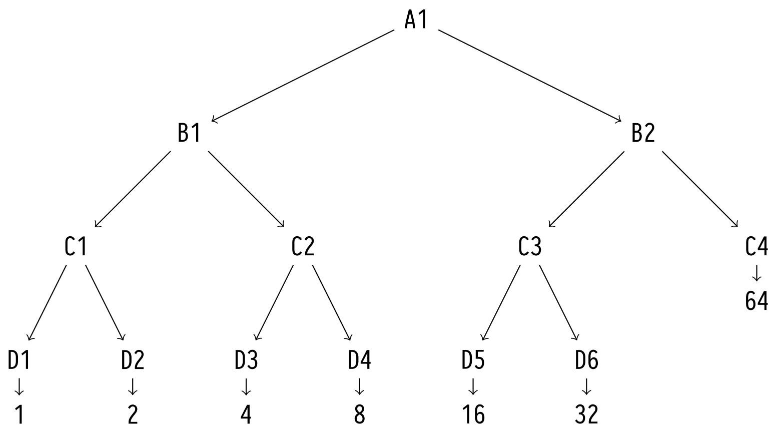

Figure 1: In this example instance of `tree`, the nodes `D1`, `D2`, `D3`, `D4`, `D5`, `D6` and `C4` are leaves.

First, create the table `tree` and populate it with inner nodes and leaves. You will find that all rows of table `tree` describe a *root-to-leaf-path*.

```sql
CREATE TABLE tree (
  level_a    char(2),
  level_b    char(2),
  level_c    char(2),
  level_d    char(2),
  leaf_value int
);
```

```sql
INSERT INTO tree(level_a, level_b, level_c, level_d, leaf_value)
VALUES ('A1', 'B1', 'C1', 'D1', 1),
       ('A1', 'B1', 'C1', 'D2', 2),
       ('A1', 'B1', 'C2', 'D3', 4),
       ('A1', 'B1', 'C2', 'D4', 8),
       ('A1', 'B2', 'C3', 'D5', 16),
       ('A1', 'B2', 'C3', 'D6', 32),
       ('A1', 'B2', 'C4', NULL, 64);
```

(a) Write a SQL query using `GROUP BY ROLLUP` (no `WITH RECURSIVE` or user-defined functions) which produces a tree in which each node holds the sum of all labels in its subtree. Leaves keep their original values.

For the **example tree** in Figure 1, your query should produce the following result:

level_a	level_b	level_c	level_d	sum
'A1'	'B1'	'C1'	'D1'	1
'A1'	'B1'	'C1'	'D2'	2
'A1'	'B1'	'C2'	'D3'	4
'A1'	'B1'	'C2'	'D4'	8
'A1'	'B2'	'C3'	'D5'	16
'A1'	'B2'	'C3'	'D6'	32
'A1'	'B2'	'C4'	NULL	64
'A1'	'B1'	'C1'	NULL	3
'A1'	'B1'	'C2'	NULL	12
'A1'	'B2'	'C3'	NULL	48
'A1'	'B1'	NULL	NULL	15
'A1'	'B2'	NULL	NULL	112
'A1'	NULL	NULL	NULL	127

(b) Similar to (a), write a SQL query to count the leaves below each node (the count of leaves below a leaf is 0).

Example:

level_a	level_b	level_c	level_d	count
…	…	…	…	…
'A1'	'B2'	'C4'	NULL	0
'A1'	'B1'	NULL	NULL	4
'A1'	NULL	NULL	NULL	7
…	…	…	…	…

(c) Let the tree describe a tournament in which the leaves and their values represent contestants and their skill level, respectively. When siblings compete, the contestant with the higher skill level proceeds to the parent node. On a tie, either contestant may proceed.

Write a SQL query similar to (a), to determine the winning skill values for each competition. The tree root will hold the overall winning skill value.

Example: Each leaf node competes with its sibling. So in one of the competitions `D3` and `D4` compete and `D4` is then declared the winner and moves up to its parent node `C2`.

level_a	level_b	level_c	level_d	skill_level
…	…	…	…	…
'A1'	'B1'	'C2'	'D3'	4
'A1'	'B1'	'C2'	'D4'	8
'A1'	'B1'	'C2'	NULL	8
…	…	…	…	…