Advanced SQL
Forum: `https://forum-db.informatik.uni-tuebingen.de/c/ss20-asql`

Assignment 2

Relevant videos: up to #12
`https://tinyurl.com/AdvSQL-2020`
Submission: Tuesday, 12.05.2020, 10:00 AM

1. [15 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 t) she intends to store inside a RDBMS of your choice (PostgreSQL 12, of course). She explains 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) Calculate the global maximum of the measurements $m(t)$.

   (c) For each t, calculate the average of the measurement $m(t)$.

   (d) Find 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 every timestamp t at which the global maximum was recorded.

2. [15 Points] **Limited-depth trees and GROUPING SETS**

Describe a binary tree with depth of up to four, where each level is assigned a letter `A`,`B`,`C` and `D`. Leaves are any node without children and are the only nodes with values initially assigned to them. For a more visual representation, see Figure 1.

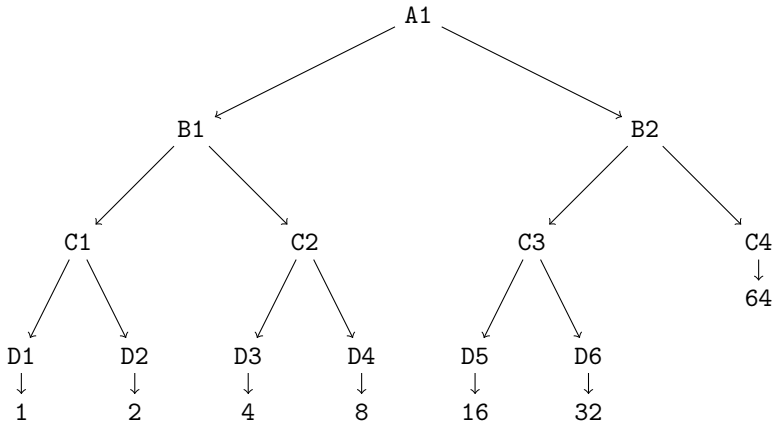

Figure 1: Representation of table `tree`. In this example, the nodes `D1`, `D2`, `D3`, `D4`, `D5`, `D6` and `C4` are leaves.

First, create the table `tree` and populate it with nodes and leaves.

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

```
INSERT INTO tree 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 where the value of each node is the sum of its children. Leaves keep their initial values.

For the sample tree above, 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'	□	64
'A1'	'B1'	'C1'	□	3
'A1'	'B1'	'C2'	□	12
'A1'	'B1'	'C3'	□	48
'A1'	'B1'	□	□	15
'A1'	'B2'	□	□	112
'A1'	□	□	□	127

For example, `B1`'s value is the sum of its children `C1` and `C2` for which we assume that their sum has already been computed.

(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'	□	0
'A1'	'B1'	□	□	4
'A1'	□	□	□	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: For 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'	□	8
...	...	...	...	...