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

Assignment 10

Relevant videos: up to #51
`https://tinyurl.com/AdvSQL-2020`
Submission: Tuesday, 14.07.2020, 10:00 AM

1. [10 Points] **Transport by Plane**

   Consider the table `planes` which lists all available planes and their respective total weight `allowance` (in gram). Another table lists all `items` and their `weight` (in gram).

```
CREATE TABLE planes(
  plane     int PRIMARY KEY,
  allowance int NOT NULL
);
```

```
CREATE TABLE items(
  item   int PRIMARY KEY,
  weight int NOT NULL
);
```

   Your task is to construct a SQL query, which independently puts any subset of items on a plane making sure that the total `weight` leaves as little weight `allowance` as possible unused. The result should include the `planeid`, the `items` to put on the plane, the total `weight` of these items and the weight `allowance`. You may assume that we want to board one plane in the end, so putting the same item onto different planes is allowed.

   Example:

```
INSERT INTO planes(plane, allowance)
  VALUES
  (1, 25000),
  (2, 19000),
  (3, 27000);
```

```
INSERT INTO items(item, weight)
  VALUES
  ( 1,  7120), ( 2,  8150),
  ( 3,  8255), ( 4,  9051),
  ( 5,  1220), ( 6, 12558),
  ( 7, 13555), ( 8,  5221),
  ( 9,   812), (10,  6562);
```

   This would produce the following result:

planeid	items	weight	allowance
1	{2,3,5,9,10}	24999	25000
2	{5,6,8}	18999	19000

   Note, how plane `3` would leave more than one gram of the weight allowance unused. In the best case, you can choose items `1,5,6,8,9` which uses `26931` gram of the total weight allowance of `27000`. This leaves us with more than one gram unused weight allowance and therefore we do not list plane `3` in the result.

2. [10 Points] **Heat Flow**

Consider table `heat` which represents the heat distribution in a rectangular space of arbitrary size:

```
CREATE TABLE heat (
  x int,
  y int,
  z float,
  PRIMARY KEY (x,y)
);
```

Columns `x` and `y` describe the position in the rectangular space while column `z` measures the temperature at that position. You can assume that all measurements are non-negative. For example, the following is a valid instance of table `heat` with position (0,0) being the only position where heat was measured:

```
INSERT INTO heat(x,y,z) VALUES
(0,0,10), (1,0, 0), (2,0, 0), (3,0, 0), (4,0, 0),
(0,1, 0), (1,1, 0), (2,1, 0), (3,1, 0), (4,1, 0),
(0,2, 0), (1,2, 0), (2,2, 0), (3,2, 0), (4,2, 0),
(0,3, 0), (1,3, 0), (2,3, 0), (3,3, 0), (4,3, 0),
(0,4, 0), (1,4, 0), (2,4, 0), (3,4, 0), (4,4, 0);
```

Write a SQL query which simulates the flow of the heat based on what is measured in table `heat` and produces the heat distribution on the n-th iteration of the simulation. The following discrete heat flow equation:

$$\Delta z(x,y) = C \cdot (z(x-1,y) - 2z(x,y) + z(x+1,y) + z(x,y-1) - 2z(x,y) + z(x,y+1))$$

where $C = \frac{1}{10}$ and $z(x,y)$ calculates the heat `z` at position (x,y). The result is the same format as the table `heat`.

Hint: Model your query like a two-dimensional cellular automaton. The concept of two-dimensional cellular automata was introduced through the *Game of Life* and the *liquid flow simulation* in the lecture (Video #49 and #50 respectively) .

3. [10 Points] **Traveling by PL/SQL**

Consider Exercise 3 of Assignment 8. **Without** the use of *recursive queries*, write a table-valued PL/SQL function `reachable(city text) RETURNING SETOF text` which, similar to the previous exercise, returns every reachable city by car. But, unlike before, where the starting city was *Saarbrücken*, the starting city is now provided by the `city` parameter of the function.

Example:

For example, the query

```
SELECT *
FROM   reachable('Saarbruecken');
```

produces the following result:

reachable
Augsburg
Darmstadt
Erlangen
Heidelberg
Kaiserslautern
Karlsruhe
Landshut
Mannheim
Muenchen
Nuernberg
Rosenheim
Saarbruecken
Stuttgart
Ulm
Wuerzburg