EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

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

Assignment 10 (15.07.2022)

Handin until: 22.07.2022, 09:00

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

1. [7 Points] **Heat Flow**

   You are given a table **heat** which represents the heat distribution in a rectangular material of arbitrary size. The table **heat** is defined as follows:

```
CREATE TABLE heat (
  x int           CHECK (x > 0),   -- 2D location (x,y)
  y int           CHECK (y > 0),
  z float NOT NULL CHECK (z >= 0),  -- heat at (x,y)
  PRIMARY KEY (x,y)
);
```

   Each part of the material has unique coordinates **x** and **y** assigned to determine its position. The column **z** describes the heat measured at that position. All measurements are positive. For example, the following is a valid instance of table **heat** with position **(0,0)** being the only heated part of the material:

```
INSERT INTO heat(x,y,z) VALUES
  (1,1,10), (2,1, 0), (3,1, 0), (4,1, 0), (5,1, 0),
  (1,2, 0), (2,2, 0), (3,2, 0), (4,2, 0), (5,2, 0),
  (1,3, 0), (2,3, 0), (3,3, 0), (4,3, 0), (5,3, 0),
  (1,4, 0), (2,4, 0), (3,4, 0), (4,4, 0), (5,4, 0),
  (1,5, 0), (2,5, 0), (3,5, 0), (4,5, 0), (5,5, 0);
```

   Consider 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 $\Delta z(x,y)$ defines the change of heat at the position (x,y) dependent on its neighboring parts. Assume that the heat surrounding the material is **0**.

   Write a SQL query which simulates the flow of the initially defined **heat** and calculates the final heat distribution after n iterations. The result table of the query produces the same columns as table **heat**.

   Hint: Model this query based on a two-dimensional cellular automaton. The concept of two-dimensional cellular automata was introduced through the *Game of Life* in the lecture.

2. [15 Points] **Stormtroopers On Patrol**

The rebels ask for your help in infiltrating an imperial checkpoint. They want you to write a SQL query which calculates every safe position in a rectangular room of arbitrary size. A position is unsafe if it can be spotted (seen) by a stormtrooper. They hand you the following definition for table `room`:

```
CREATE TABLE room (
  x   int      CHECK(x > 0),
  y   int      CHECK(y > 0),
  dir char(1) NOT NULL
      CHECK (dir IN ('E','W','N','S','X')),
  PRIMARY KEY (x,y)
);
```

Stormtroopers always begin their patrol at the north-west corner of the room at (1,1) and stop at where the × is placed. Assume that stormtroopers *only* look in the direction they move and *always* reach × which is never placed at (1,1).

The path they take depends on the directions `E`, `W`, `N` and `S` at specified positions in the room. Stormtroopers spot *all* positions in their walking direction (starting from their current position up to the wall) and split up to take all of them. Assume that there are always enough stormtroopers patroling to do so.

Example: We define the dimensions `width` and `height` and populate table `room` with the following sample directions, visualized in Figure 1:

```
\set width  4
\set height 4

INSERT INTO room(x,y,dir) VALUES
  (1, 1, 'E'), (1, 2, 'E'),
  (3, 1, 'S'), (4, 1, 'S'),
  (2, 3, 'S'), (3, 3, 'W'),
  (2, 4, 'E'), (3, 4, 'X'),
  (4, 3, 'X');
```

In this example the stormtroopers begin their patrol at (1,1) and go east immediatelly and may then either go south at (3,1) or (4,1). Both paths are taken which ends their patrol in (3,4) and (4,3). In this example, your query should produce the following *safe* positions:

x	y
1	4
2	2
1	2

Note, how (1,3) can be *spotted* by stormtroopers looking west as they pass (2,3) and (3,3) making this position *unsafe*.

	1	2	3	4
1	→		↓	↓
2	←			
3		↓	←	×
4		→	×	

Figure 1: A 4x4 room of an imperial base with directions (→) and ×s.

3. [8 Points] **Travel by Car**

You decide to travel by car to various cities in Germany starting from *Saarbrücken*. But your car only holds up to 100 units of fuel and each unit lets you travel a single distance unit. And if the fuel runs out in the middle of the road, your travels come to an abrupt end. Luckily, some cities allow for refueling.

Based on the roadmap provided in the SQL file **`travel.sql`**, write a SQL query, which lists every city you are able to reach with your car.

Example: As an example, we take a look at a simplified roadmap (Figure 2).

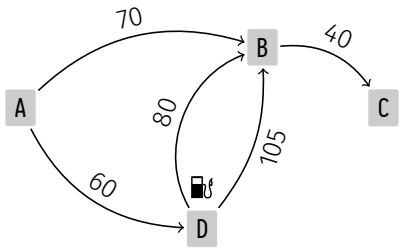

Figure 2: Road network with travel distances between cities, locations of fueling stations (⛽).

We want to calculate which cities we can reach when starting at `A`. We need 70 units of fuel to reach `B`, which leaves us with 30. This is not enough to reach `C`. We are also able to reach `D` using 60 units of fuel. `D` provides refueling (⛽) and therefore we can reach `B` from here, if we travel the road with distance 80. This path leaves us with 20 units of fuel at `B` which is still not enough to reach `C`. We conclude that `C` as unreachable.

The result of the query is therefore:

reachable
D
B
A