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

Assignment 9

Relevant videos: up to #51
`https://tinyurl.com/AdvSQL-2020`
Submission: Tuesday, 07.07.2020, 10:00 AM

1. [7 Points] **Traveling Salesman**

   A map is defined by populating the table `roads`.

```
CREATE TABLE roads (
  here  text,
  dist  int,
  there text,
  PRIMARY KEY(here, there)
);
```

   The columns `here` and `there` represent the cities which are connected by roads from `here` to `there`. These roads can be traveled in either direction and have a traveling distance as recorded in column `dist`.

   (a) Write a common-table expression (CTE), which extracts each unique city name from the table `roads` into a table with a single column `name`.

   (b) Using the CTE of (a), write a SQL query which returns the shortest path which starts in an arbitrary city, visits every other city once and then returns back to the city you started from for any graph G.[1] **Hint:** To avoid running into cycles while you explore the road graph, add a column to your result table that maintains an array that contains all cities you have already visited.

   Example: Consider a simple map in Figure 1.

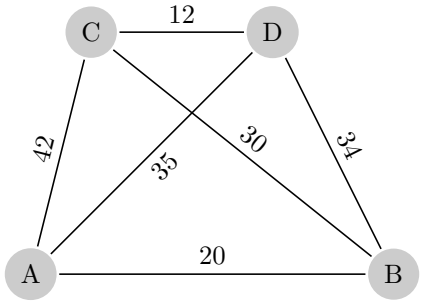

Figure 1: Weighted undirected complete graph example.

Applying the query to this graph would yield the following result:

visited in order	total distance
{A,B,C,D,A}	97

[1] Traveling Salesman Problem

2. [16 Points] **Stormtroopers On Patrol**

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

```
CREATE TYPE direction AS ENUM
    ('N','S','W','E');

CREATE TABLE room (
  x    int       NOT NULL,
  y    int       NOT NULL,
  dir direction NOT NULL,
  PRIMARY KEY (x,y));
```

The rebels are still scouting the location, so they still do not know the dimensions of the room nor the path the stormtroopers take. But one thing can be assumed: the stormtroopers *always* begin their patrol in the north-west corner of the room at (1,1) and end it whenever they run into a wall.
Once the scouts are back with the information, the rebels will set `width` and `height` of the room as well as populate table `room` with direction markers (`N`, `S`, `W`, `E`) to indicate the patrol path(s).

About the stormtroopers:

- If the stormtroopers have multiple possibilities to change direction on their patrol, they simply split up and take all of them.
- Stormtroopers have sight, so they can spot *all* positions in their walking direction (from their current position up to the wall) which makes those positions *unsafe*.

Example: An example room in Figure 2 is defined as follows:

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

The stormtroopers start moving at (1,1). They move east immediately and then move south at (3,1) and (4,1). They continue these patrol paths to the end (at (4,4)) and as such the following positions can be considered *safe*:

x	y
1	4
2	2
1	2

Note how (1,3) is considered *unsafe*, as it can be *spotted* by patrolling stormtroopers.

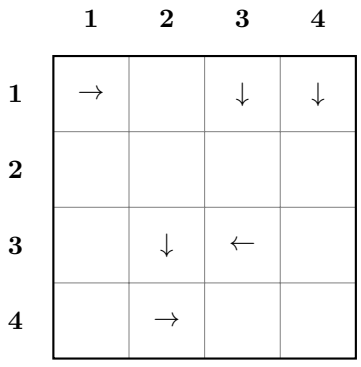

Figure 2: A 4×4 room with directions indicating the path the stormtroopers take.

3. [7 Points] **Logged in?**

Specifically for this task, we want you to solve this using *window functions*. Consider the `log` table which records whether a user is logged into their workstation at a given hour (a log entry is made for each user at every hour, logged in or not):

```
CREATE TABLE log (
  hour          int,
  userid        int,
  "logged in?" boolean);
```

Construct a SQL query to compute a table that, for each hour h, lists those users that had the longest aggregate login time up to and including h. For the result schema, see the table `result` below.

Example: For the sample `log` table we expect the following result:

`log`

hour	userid	logged in?
1	1	FALSE
2	1	TRUE
3	1	TRUE
4	1	TRUE
5	1	FALSE
6	1	TRUE
1	2	FALSE
2	2	FALSE
3	2	TRUE
4	2	TRUE
5	2	TRUE
6	2	FALSE

`result`

hour	userid	total logged
1	1	0
1	2	0
2	1	1
3	1	2
4	1	3
5	1	3
5	2	3
6	1	4