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

# Assignment 5

Relevant videos: up to #27
`https://tinyurl.com/AdvSQL-2020`
Submission: Tuesday, 09.06.2020, 10:00 AM

1. [10 Points] **Array Representations**

   Arrays can be represented in different ways. One is to use the built-in arrays of PostgreSQL. As such, we populate the following table `s`:

```
CREATE TABLE s (                        INSERT INTO s VALUES
  arr_id integer PRIMARY KEY,           (1, ARRAY['a','b','c','b']),
  arr    text[]);                       (2, ARRAY['e','f','e']);
```

   Alternatively, let us assume that there is **no built-in array data type**. We would then encode arrays in regular tables using explicit element positions (also see Chapter 4, slide 4). This leads us to table `t` which replaces table `s`:

```
CREATE TABLE t (                        INSERT INTO t VALUES
  arr_id integer,                       (1,1,'a'),(1,2,'b'),(1,3,'c'),(1,4,'b'),
  idx    integer,                       (2,1,'e'),(2,2,'f'),(2,3,'e');
  val    text,
  PRIMARY KEY(arr_id, idx));
```

   Likewise, queries that rely on built-in array operations (see the five queries below which refer to `s`) would need to be rewritten into queries over `t` without any reference to such operations. Queries that originally returned array values would now return their tabular encodings instead.

   Perform these rewrites for the five SQL queries below.

   (a)
```
SELECT s.arr[1] AS val
FROM   s AS s
WHERE  s.arr_id = 1;
```

   (b)
```
SELECT array_length(s.arr,1) AS len
FROM   s AS s
WHERE  s.arr_id = 1;
```

   (c)
```
SELECT a AS val
FROM   s              AS s,
       unnest(s.arr) AS a
WHERE  s.arr_id = 2;
```

   (d)
```
SELECT array_position(s.arr, 'b')
FROM   s AS s
WHERE  s.arr_id = 1;
```

   (e)
```
TABLE s
  UNION ALL
SELECT r.id + 1          AS arr_id,
       s.arr||'g'::text AS arr
FROM s AS s, (
  SELECT MAX(s.arr_id)
  FROM   s AS s
) AS r(id)
WHERE s.arr_id = 1;
```

2. [10 Points] **Transpose Two-Dimensional Arrays**

We provide you with a table definition `matrices` with two-dimensional arrays (matrices). You can assume that every matrix in this table has the same dimensions.

```
CREATE TABLE matrices (
  matrix text[][] NOT NULL
);
```

Write an SQL query to transpose each matrix.

**Example:**

```
INSERT INTO matrices VALUES
(array[['1','2','3'],
       ['4','5','6']]),
(array[['f','e','d'],
       ['c','b','a']]);
```

| transposed |
|---|
| {{1,4},{2,5},{3,6}} |
| {{f,c},{e,b},{d,a}} |

3. [10 Points] **Array Tree**

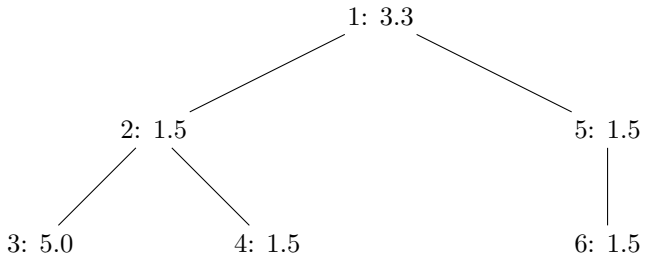
We introduced the possibility to represent trees in terms of two arrays representing parent and label information (see Chapter 4, Slide 6 **or** Video #22). For this assignment, we use `numeric` labels to define a table `trees` of array-encoded trees:

```
CREATE TABLE trees (
  tree    int PRIMARY KEY,
  parents int[],
  labels  numeric[]
);
```

**Example:** Populate the table `trees` with some sample trees.

```
INSERT INTO trees VALUES
(1, ARRAY[NULL,1,2,2,1,5],
    ARRAY[3.3,1.5,5.0,1.5,1.5,1.5]),
(2, ARRAY[3,3,NULL,3,2],
    ARRAY[0.4,0.4,0.2,0.1,7.0]);
```

Drawing `tree 1` would result in the following graph ($n : l$ indicates that node $n$ has label $l$):




Write a SQL query which, for each node $n$, determines the label sum of $n$'s children. For the example above the result would be:

| tree | node | sum |
|---|---|---|
| 1 | 1 | 3.0 |
| 1 | 2 | 6.5 |
| 1 | 3 | 0.0 |
| 1 | 4 | 0.0 |
| 1 | 5 | 1.5 |
| 1 | 6 | 0.0 |
| 2 | 1 | 0.0 |
| 2 | 2 | 7.0 |
| 2 | 3 | 0.9 |
| 2 | 4 | 0.0 |
| 2 | 5 | 0.0 |

## How To UTF-8: FAQ

Some assignments require your system to be set up to work with UTF-8 encoded files and characters. For this purpose we set up this *FAQ* which should help to alleviate some of the work for setting up your system to work with UTF-8 encoded files and characters.

**I think, my database is not set up for UTF-8. What to do?**
First check if your database your are using is set up to support UTF-8. Run `psql --list` to find out. If it turns out your database is **not** set up to support UTF-8, you can change that by running the following query `CREATE DATABASE <dbname> WITH ENCODING 'UTF8';`.
Another reason could be the client encoding not being set to UTF-8. You can verify this by running `psql` and then entering `show client_encoding;`. If you do not see `UTF8` as the result for `client_encoding`, run `\encoding UTF8`.

**I think, my terminal is not set up for UTF-8. What to do?**
If your terminal output scrambles UTF-8 characters, some of the most common reasons are:
Your terminal is not set up to support UTF-8. Possible solution:

**Windows:**
You are required to run `chcp 65001` before any UTF-8 characters can be printed correctly.

**Linux/MacOS:**
It should just work out of the box with most terminals.

The font your terminal or your system is using may also not support some specific UTF-8 characters. In which case, you'll have to switch to a font supporting these characters. For example, `Roboto Mono` at `https://fonts.google.com/` is a reasonable (and free) font to use.

**I think, my editor is not set up for UTF-8. What to do?**
Look up how to change the encoding of your editor.

**I saved my file using something else than UTF-8 encoding and now the characters of the file are all scrambled. What to do?**
If that happens you simply have to revert the files back to their initial encoding with the help of `git`. Simply run `git checkout <file>` to revert to it to the last commit state. In case you already committed the file, simply revert back to the file of an earlier commit by running `git checkout <commit> <file>`.