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

Assignment 1

Relevant videos: up to #10
`https://tinyurl.com/AdvSQL-2020`
Submission: Tuesday, 05.05.2020, 10:00 AM

1. [0 Points] **Introduction**

   (a) **Before we grade your** team, you have to complete a few administrative tasks first. In your team's GitHub Classroom repository, there exists a file called `README.md`. Add each team member's name, surname, matrikel number, subject, field of study, e-mail as well as forum username to the incomplete table and commit+push the changes.
   Completing this task is a **requirement** for your team to be graded in the first place. Please make sure the file always exists with the correct information present.
   Note: Because of the high demand/low supply of exercise slots, please note that **not completing this task until the submission date (05.05.2020)** will prompt us to remove anyone not present in the `README.md` from the exercises, freeing a slot for students on the waiting list.

   (b) All of your submissions will be placed inside the `assignments/` directory of your team's GitHub Classroom repository. Each submission requires its own subdirectory called `solution<number>`, where number is the current assignment number.
   For example, the submission of this assignment will be located in the `assignments/solution01/` directory.

   (c) In general, the only accepted file formats are plain text files (.txt) and source files (.sql, .py, ...). Other formats may not be graded, unless stated otherwise.

   (d) Your submitted code (SQL DDL statements and queries, primarily) has to work out of the box. If particular preparatory steps have to be taken to run your queries, document these steps properly.

   (e) Lastly, the usual rules for plagiarism and other academic integrity apply.

   (f) For any further questions about this lecture, assignment and other related topics, visit the forum at

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

2. [5 Points] **So similar, yet so different**

   Create an instance of the table `R` with schema `R(a int, b int)`, where the two **different** queries `Q1` and `Q2` compute differing results.

```sql
-- Q1
SELECT r.a, COUNT(*) AS c
FROM   R AS r
WHERE  r.b <> 3
GROUP BY r.a;
```

```sql
-- Q2
SELECT r.a, COUNT(*) AS c
FROM   R AS r
GROUP BY r.a
HAVING EVERY(r.b <> 3);
```

   Note: Query `Q2` uses the aggregate function `EVERY`. Read about it in the PostgreSQL documentation[1].

[1] `https://www.postgresql.org/docs/12/functions-aggregate.html`

3. [10 Points] **Production Steps**

You are handed a table of production steps, which tracks the progress of all products by providing a `product_name` and each production `step` number. The `completion_date` of each production step is set, if the step has been completed at that date. Otherwise, the `completion_date` is `NULL`. A product can have several incompleted production steps in no particular order. We define the table as follows.

```
CREATE TABLE production_steps (
  product_name    CHAR(20) NOT NULL,
  step            INTEGER  NOT NULL,
  completion_date DATE, -- NULL means incomplete
  PRIMARY KEY (product_name, step));
```

Write a SQL query which lists all `product_name`s of completed product once. A product is complete, once the `completion_date` of all production steps of a product is not `NULL`.

For **example** here

```
INSERT INTO production_steps VALUES
('TIE', 1, '1977/03/02'), ('AT-AT', 1, '1978/01/03' ), ('DS␣II', 1,  NULL       ),
('TIE', 2, '1977/12/29'), ('AT-AT', 2,  NULL        ), ('DS␣II', 2, '1979/05/26'),
                                                       ('DS␣II', 3, '1979/04/04');
```

we expect the following result

complete
'TIE'

4. [15 Points] **Matrix Multiplication**

Given two tables representing two matrices A and B of arbitrary size. We create those tables with three columns, where `row` represents the row index and `col` represents the column index of the matrix. The column `val` represents the value of a matrix element.

```
CREATE TABLE A (
  row int,
  col int,
  val int,
  PRIMARY KEY(row, col));
```

```
CREATE TABLE B ( LIKE A );
```

(a) Write a SQL query, which performs a matrix multiplication $A \cdot B$.

Example:

$$\text{Given } A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}, \text{ so } A \cdot B = \begin{pmatrix} 5 & 4 & 5 \\ 11 & 10 & 11 \end{pmatrix}.$$

So, for this input

```
INSERT INTO A (row,col,val)
VALUES (1,1,1), (1,2,2),
       (2,1,3), (2,2,4);
```

```
INSERT INTO B (row,col,val)
VALUES (1,1,1), (1,2,2), (1,3,1),
       (2,1,2), (2,2,1), (2,3,2);
```

we expect the following result from your query

row	col	val
1	1	5
1	2	4
1	3	5
2	1	11
2	2	10
2	3	11

(b) Is it possible to apply the SQL query from (a) to the following matrices in which some entries are missing? Explain briefly.

```
INSERT INTO A (row,col,val)
VALUES (1,1,1), (1,2,3),
       (2,3,7);
```

```
INSERT INTO B (row,col,val)
VALUES (1,1,4),          (1,3,8 ),
       (2,1,1), (2,2,1), (2,3,10),
       (3,1,3), (3,2,6);
```