EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

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

Assignment 1 (29.04.2022)

Handin until: 06.05.2022, 09:00

1. [0 Points] **Introduction**

   (a) **Before we are able to grade you** and/or your team, you have to complete this initial task first. In your team's GitHub Classroom repository, there exists a file called `README.md`. Add both team member names, surname, matrikel number, etc. to the file.
   This task does not grant you any points, but is a **requirement** for your team to be graded in the first place. This file has to persist throughout the entire semester.

   (b) Each submission of each assignment in "Advanced SQL" must be put into its own folder in the `root` directory of your team's GitHub Classroom repository. The name of the folder has to be in the format `solution<number>`. For example: the folder of your submission for this assignment must be called `solution01`.

   (c) In general, the only accepted file format is plain text (*.txt, *.sql for SQL code). Other file formats are now allowed, unless stated otherwise. Your submitted code has to be consistently formated, well-documented and has to run without errors using PostgreSQL 14. Older versions of PostgreSQL may also work, but for those you are on your own.

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

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

   https://forum-db.informatik.uni-tuebingen.de/c/ss22-asql.

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

   Create an instance for table `r` with schema `r(a int, b int)` such that the two queries `Q1` and `Q2` compute **different** 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].

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

   Consider table `production` which tracks the progress of items (column `item`) currently in production. Production occurs in multiple steps (column `step`). If a step has been completed for an item, column `completion` indicates the time and date. If the step is still in progress, column `completion` holds `NULL` for that step. Table `production` is defined as follows:

```sql
CREATE TABLE production (
  item       char(20) NOT NULL,
  step       int      NOT NULL,
  completion timestamp,          -- NULL means incomplete
  PRIMARY KEY (item, step));
```

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

For **example**, in this instance

product_name	step	completion_date
'TIE'	1	'1977/03/02 04:12'
'TIE'	2	'1977/12/29 05:55'
'AT-AT'	1	'1978/01/03 14:12'
'AT-AT'	2	NULL
'DSII'	1	NULL
'DSII'	2	'1979/05/26 20:05'
'DSII'	3	'1979/04/04 17:12'

step 2 of the production of AT-AT is not complete yet. Likewise, Step 1 of DS II is pending, while Steps 2 and 3 are complete.

Formulate a SQL query that lists the names of all `items` for which all production steps are complete. Avoid duplicate item names. For the instance above, we expect the following result:

complete
'TIE'

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

You are given two tables representing two matrices A and B of sizes $m \times n$ and $n \times p$, respectively. 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.

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

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

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

Example:

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

So, for these instances

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

```sql
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) In *sparse matrices*, all entries of value 0 are missing. See the examples of sparse matrices A and B below. Does your SQL query of (a) also apply to sparse matrices?

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

```sql
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);
```