Databases

TDA357/DIT621 (4.5 hec)

Responsible: Ana Bove, tel: 1020

Friday 17th of March 2023, 8:30–12:30

Total: 60 points	
CTH: $\geqslant$ 27: 3, $\geqslant$ 38: 4, $\geqslant$ 49: 5	GU: $\geqslant$ 27: G, $\geqslant$ 45: VG

Make sure your handwriting and drawings are readable!
What we cannot read we cannot correct!

The exam has 6 questions. Make sure to turn pages! :)

Note: As said in the course page, if you brought **one hand written double sided A4** of notes to the exam, you **must hand it in** along with your solutions.

Good luck!

1 SQL and Constraints (8 pts)

The teacher of this course has decided to open a restaurant instead of teaching Databases. She has found a nice location and is now asking for your help.

There is a menu with different drinks and dishes, and their prices. The name of the drink or dish is enough to identify it. Items in the menu are also classified into starters, main dish, dessert, or drink.

There are a certain number of tables in the location, each of them with an identification number. Tables have a size (2, 4, 6 or 8) indicating how many people can sit around the table. Guests can also sit at the bar which has id 0 and no size.

There are a few waiters/waitresses taking care of the guests. They are identified by their names, have a telephone number and a salary which is at least 25.000 sek according to the agreement with the union.

The relational schema for the restaurant is the following:

Menu (item, price, type)
Tables (id, size)
Waiters (name, phone, salary)

a) (2 pts) Define SQL tables for the relational schema above. Make sure to define appropriate types and constraints.

b) (3 pts) Define two new SQL tables *Sittings* and *Orders*:

- The current sitting in the restaurant: the system needs to know which table a group is sitting at, how many people are sitting around the table (which could differ from the size of the table!) and who is the waiter/waitress taking care of the group. There is of course only at most a sitting per table in the current sitting! A waiter/waitress can on the other hand take care of many tables. When the group of people in a table pays and leaves the restaurant, the information about that particular sitting is removed from the system.

- The orders at each sitting: that is, what has been ordered in each of the tables that have guests at the moment; this information will be used when billing the table.

Observe that people at the bar are not part of the current sitting in the restaurant; also they must pay when they order, so their orders are not stored in the system.

c) (3 pts) Write an SQL query that for each table, outputs the table id and its status. The status of a table is *empty* if there is no group currently sitting at the table, or *lose*/*draw*/*win* if the table has less people that its size/as many people as its size/more people than its size, respectively. The output should be sorted by the table id.

Recall that the bar is not really a table and should not appear in the result of this query.

Solution:

```
CREATE TABLE Menu (
  item CHAR(20) PRIMARY KEY,
  price INT NOT NULL CHECK (price > 0),
  type TEXT NOT NULL
     CHECK (type IN ('drink', 'starter', 'main dish', 'dessert')) );

CREATE TABLE Tables (
  id INT PRIMARY KEY,
  size INT CHECK (id = 0 OR size IN (2,4,6,8)) );

CREATE TABLE Waiters (
  name TEXT PRIMARY KEY,
  phone INT NOT NULL UNIQUE CHECK (phone > 0),
  salary INT NOT NULL CHECK (salary >= 25000) );

CREATE TABLE Sittings (
  tableid INT PRIMARY KEY REFERENCES Tables CHECK (tableid != 0),
  nrguests INT NOT NULL CHECK (nrguests > 0),
  waiter TEXT NOT NULL REFERENCES Waiters );

CREATE TABLE Orders (
  tableid INT REFERENCES Sittings CHECK (tableid != 0),
  item CHAR(20) REFERENCES Menu,
  amount INT CHECK (amount > 0),
  PRIMARY KEY (tableid, item) );
```

```
-- Status of each table
-- Could be done with union as the example on big/small countries
-- from the lecture
SELECT id AS tableNr,
       (CASE WHEN (nrguests IS NULL) THEN 'empty'
             WHEN (size - nrguests) = 0 THEN 'draw'
    WHEN (size - nrguests) > 0 THEN 'lose'
    WHEN (size - nrguests) < 0 THEN 'win'
END) AS status
FROM Tables LEFT JOIN Sittings
ON id = tableid
WHERE id != 0
ORDER BY tableNr;
```

2 More SQL and Relational Algebra (14 pts)

We continue with the same domain as in question 1 on SQL:

Menu (item, price, type)
Tables (id, size)
Waiters (name, phone, salary)

together with the two other tables you have defined.

a) (2 + 2 pts) Write an SQL query and a relational algebra expression that for each waiter/waitress, outputs his/her name, the number of tables he/she is taken care of, and the total number of guests each waiter/waitress is taking care of at the moment.

The output should be in descending order after the number of tables a waiter/waitress is serving.

Solution:
This solution outputs information only about the waiters/waitresses which are actually taking care of guest at the moment.

The alternative that outputs 0 for those not currently working at the moment is also correct (possibly even more correct, depending on how the text is interpreted).

```
-- Tables and guests per waiter/waitress
SELECT waiter, COUNT(*) AS nrTables, SUM(nrguests) AS totalNrGuests
FROM Sittings
GROUP BY waiter
ORDER BY nrTables DESC;
```

$$\tau_{-\mathsf{nrTables}}(\gamma_{\mathtt{waiter,COUNT(*)\to nrTables,SUM(nrguests)\to total}}\ \mathtt{Sittings})$$

b) (2 + 2 pts) Write an SQL query and a relational algebra expression that for each of the items in the menu that is not a drink and that costs more than 50 sek, outputs the number of times that the item has been ordered in the current sitting.

Solution:

```
SELECT item, SUM(amount)
FROM Orders NATURAL JOIN Menu
WHERE type != 'drink' AND price > 50
GROUP BY item;
```

$$\gamma_{\mathtt{item,\ SUM(amount)\to nr}}(\sigma_{\mathsf{type!='drink'}\wedge\mathsf{price}>50}(\mathtt{Orders} \bowtie \mathtt{Menu})))$$

c) (3 + 3 pts) Write an SQL query and a relational algebra expression that outputs the id of all the tables in the current sitting that have ordered an average of at least 2 drinks containing alcohol per person. Drinks that cost less than 40 sek are non-alcoholic, those costing 40 sek or more contain alcohol.

Solution:

```
SELECT tableid
FROM Sittings NATURAL JOIN Orders NATURAL JOIN Menu
WHERE type = 'drink' AND price >= 40
GROUP BY tableid
HAVING SUM(amount)/nrguests >= 2;
```

$$\begin{aligned}&\pi_{\mathtt{tableid}}(\sigma_{\mathtt{average}\geqslant 2}(\\&\quad\gamma_{\mathtt{tableid,\ SUM(amount)/nrguests\to average}}(\\&\qquad\sigma_{\mathsf{type='drink'}\wedge\mathsf{price}\geqslant 40}(\mathtt{Sittings} \bowtie \mathtt{Orders} \bowtie \mathtt{Menu}))))\end{aligned}$$

3 Views and Triggers (10 pts)

We continue with the same domain as in question 1 on SQL:

> Menu (item, price, type)
> Tables (id, size)
> Waiters (name, phone, salary)

together with the two other tables you have defined.

Propose a solution (meaning, the corresponding full SQL code) to the following tasks that need to be performed.

a) (4.5 pts) Produce the billing of all current sittings that have ordered.
For each of the sittings that have ordered food and/or drinks, produce the total amount that the group needs to pay. If the sitting has not yet ordered anything, it should not be part of the billing.
Besides charging for all the items that were ordered, the bill should include an obligatory tip. The obligatory tip for a particular sitting is computed by adding 1/1000 of the waiter/waitress salary for each of the people in the group. So if the salary of the waiter/waitress is 25.000 sek and there are 3 people in the group, a total of 75 sek should be added to the sum of the prices of all the items that were ordered by the group.

b) (5.5 pts) Optimise the use of the tables in the restaurant.
After a while one noticed that groups were not necessarily placed in the best tables of the restaurant in relation to their size. So a group of, say, 3 people could be placed in a table for 6 while there were tables for 4 still available.
So, when getting a new group in the restaurant, one should actually place the group in the table that best fits the group's size.
To find the table with the best fitting size one first looks for the smallest empty table that has a size which is at least the size of the group. So, for example, a group of 3 should primarily be placed in a table for 4, if not such table is empty then one should place them in a table of 6 if there is any such table available, and only if no such table is empty then one should use an empty table for 8.
If one cannot find an empty table that fits all the people as it is explained above, one should then look for an empty table whose size is one less than the number of people in the group (adding an extra chair to the table will work well!) So in the example of finding a table for a group of 3, if no empty table has enough sits, then one should look if there is an empty table of size 2. If so, the group of 3 should be assigned that table; if not, one simple cannot accommodate the group in the restaurant at the moment!

Solution:

```
CREATE OR REPLACE VIEW Billing AS
WITH
  Consumption AS (
    SELECT tableid, SUM(amount*price) AS sum_price
    FROM Sittings NATURAL JOIN Orders NATURAL JOIN Menu
    GROUP BY tableid )
SELECT tableid, (sum_price + salary/1000*nrguests) AS bill
FROM Consumption NATURAL JOIN
     (Sittings JOIN Waiters ON waiter = name);

-- Alternative
CREATE OR REPLACE VIEW Billing AS
SELECT tableid, SUM(amount*price) + salary/1000*nrguests AS bill
FROM (Sittings JOIN Waiters ON waiter = name)
     NATURAL JOIN Orders NATURAL JOIN Menu
GROUP BY tableid, salary, nrguests;


CREATE OR REPLACE FUNCTION add_group() RETURNS TRIGGER AS $$
DECLARE
  tablenr INT;
BEGIN
    SELECT id INTO tablenr
    FROM Tables
    WHERE id NOT IN (SELECT tableid FROM Sittings)
          AND size >= New.nrguests
    ORDER BY size
    LIMIT 1;
  IF tablenr IS NULL
  THEN SELECT id INTO tablenr
       FROM Tables
       WHERE id NOT IN (SELECT tableid FROM Sittings)
             AND size = New.nrguests - 1
       LIMIT 1;
       IF tablenr IS NULL
       THEN RAISE EXCEPTION 'No table available for this group!';
       END IF;
  END IF;
  RAISE NOTICE 'Table % should be used', tablenr;
  New.tableid = tablenr;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS addSitting ON Sittings;
```

```
CREATE TRIGGER addSitting
  BEFORE INSERT ON Sittings
  FOR EACH ROW
  EXECUTE FUNCTION add_group();
```

4 ER Modelling (10.5 pts)

a) (6pts) Create an ER-diagram for the domain of management of written exams at a university.

   The database should have these properties:

   - Each exam is given for a specific course (identified by its name) at a specific date. There may be several exams on the same date, but each course can have at most one exam on any given date.
   - Each exam has a responsible teacher.
   - Some exams are re-exams. In addition to all normal attributes of exams, every re-exam can have a comment to make notes like "extra exam occasion".
   - Each re-exam is connected to a specific original exam which could even be a re-exam itself but doesn't have to.
   - Exams can be online or on-campus. If they are on-campus, they have a room name and the name of a invigilator.
   - Each student at the university has a unique id-number.
   - The university has specific regulations stating which exams each student is allowed to register to. Whenever a student is allowed to register to an exam, one has a boolean attribute called *registered*. If *registered* is false, it means the student is allowed to register for the exam but has not registered to it yet. If *registered* is true, it means the student is registered for the exam (and is allowed to do so :).

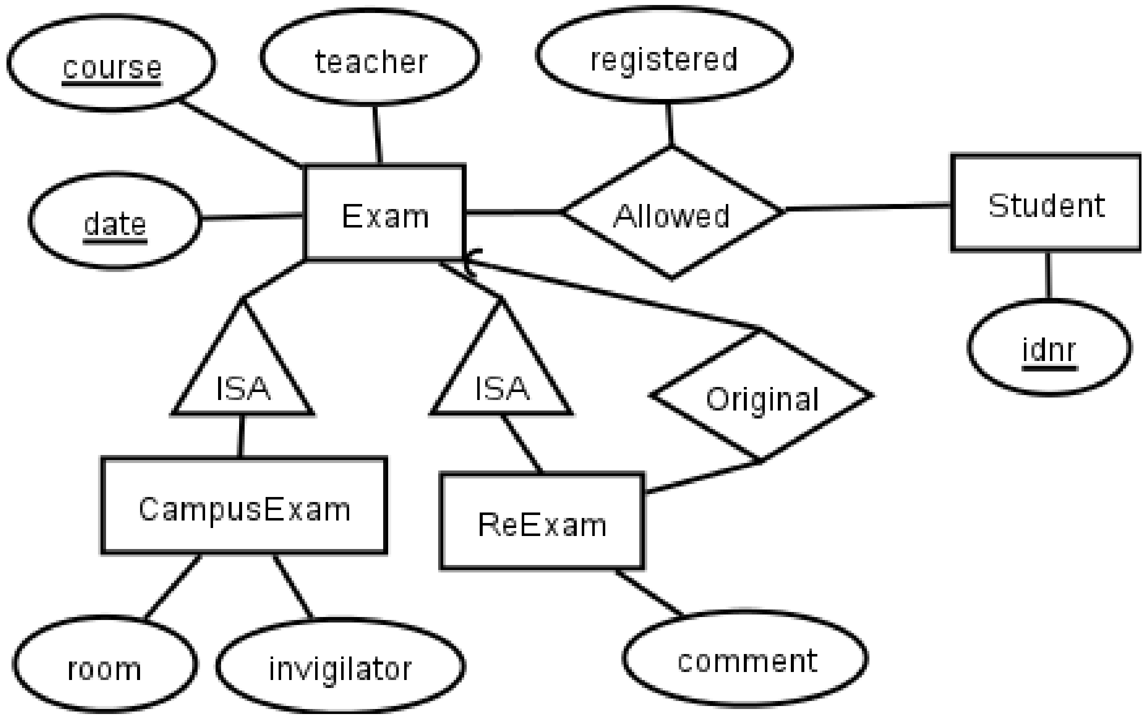
   Notes:

   - You should not add any additional attributes to students, exams or any other entity or relationship you have defined.
   - You do not need to ensure that a re-exam is in the same course as its original exam in the ER-diagram.

b) (4.5 pts) Translate your diagram into a relational schema.

   To the schema, add a constraint that makes sure re-exams are always in the same course as their original exams.

Solution:



It's fine to have made courses an enitity and the exam a weak entity.

It's also fine to have made teachers, rooms and supervisors entities with many-to-exactly-one relationships to exams and on-campus exams.

Exam (<u>course</u>, <u>date</u>, teacher)
ReExam (<u>course</u>, <u>date</u>, originalCourse, originalDate, comment)
    (course, date) → Exam (course, date)
    (originalCourse, originalDate) → Exam (course, date)
CampusExam (<u>course</u>, <u>date</u>, room, supervisor)
    (course, date) → Exam (course, date)
    course = originalCourse Student (<u>idnr</u>)
Allowed(<u>student</u>, <u>course</u>, <u>date</u>, registered)
    student → Students (idnr)
    (course, date) → Exam (course, date)

5 Functional and Multivalued Dependencies (9.5pts)

Consider the following table and its data:

a	b	c	d	e
1	2	0	2	4
1	2	0	2	5
1	3	0	2	4
2	4	1	3	6
2	4	1	3	7
3	5	2	2	7

a) (3 pts) For each of these four functional dependencies, justify whether they hold in this data or not.
Note: Just a yes/no answer gives no point; what you are asked here is your justification of whether they are valid or not.
Note: What you are asked is not the definition of what a FD is but why that definition actually holds or not in this data.

c $\rightarrow$ d
a $\rightarrow$ b e
a $\rightarrow$ c
d e $\rightarrow$ b

Solution:

c $\rightarrow$ d: it is valid since row 1, 2 and 3 have the same value in "c" and they also have the same value in "d". Same happens with rows 4 and 5.

a $\rightarrow$ b e: it is not valid since row 2 and 3 have the same value in "a" but neither "b" nor "e" have the same value in these two rows.

a $\rightarrow$ c: it is valid since row 1, 2 and 3 have the same value in "a" and they also have the same value in "c". Same happens with rows 4 and 5.

d e $\rightarrow$ b: it is not valid since row 1 and 3 have the same value in both "d" and "e" but not in "b".

b) (3 pts) Normalise your data to BCNF using the FDs from a) (and no others) that you found valid.

Describe the intermediate states so we can follow your normalisation process. Mark clearly which are the (final) relations in the BCNF schema, and the primary keys and any additional secondary keys (unique constraints) you identify.

Solution:

- Using c $\rightarrow$ d: $c^+ = \{c,d\}$.
  So we get $R_1(c,d)$ and $R_2(a,b,c,e)$.
- Using a $\rightarrow$ c: $a^+ = \{a,c,d\}$.
  So we get $R_{21}(a,c)$ and $R_{22}(a,b,e)$.

All of them are now in BCNF.

Setting the primary keys we get: $R_1(\underline{c},d)$, $R_{21}(\underline{a},c)$ and $R_{22}(\underline{a},\underline{b},\underline{e})$.

c) (1.5 pts) Split the original table into tables according to the result of your BCNF normalisation. The splitting should not add any new data and should also be a lossless join.

Solution:

a	c
1	0
2	1
3	2

c	d
0	2
1	3
2	2

a	b	e
1	2	4
1	2	5
1	3	4
2	4	6
2	4	7
3	5	7

d) (2 pts) Is the multivalued dependency a $\twoheadrightarrow$ b valid in this data? If it is valid, explain why. If it is not, would it be possible to add one or more rows in one of the tables so that the MVD is valid? State which row(s) in that case. Justify your answer.

Solution:

Adding the row “1 3 5” in the relation $R_{22}(\underline{a},\underline{b},\underline{e})$ would make the MVD valid.

Alternative, adding the row “1 3 0 2 5” in the original table.

6 JSON (8 pts)

Consider this XML document for a book collection (don't worry, the little we have seen of XML in the course is enough here). This particular document/collection has two books with title, genre, author(s), year and sometimes even tags.

```
<books>
  <book>
    <year>1990</year>
    <title>Good Omens</title>
    <author>Terry Pratchett</author>
    <author>Neil Gaiman</author>
    <genre>Comedy</genre>
  </book>
  <book>
    <year>1960</year>
    <title>To Kill a Mockingbird</title>
    <author>Harper Lee</author>
    <genre>Fiction</genre>
    <tags>
      <tag>Classic</tag>
      <tag>Civil Rights</tag>
    </tags>
  </book>
</books>
```

Note: You may want to read all the tasks below before starting on your solution.

a) (3 pts) Create a JSON document for this particular book collection. The document needs to contain all the features/information as in the XML document.

   You can abbreviate all string values (e.g. "TKaM" for "To Kill a Mockingbird") to save some writing, as far as it is easy to understand what information you are referring to.

b) (3 pts) Write a JSON schema that validates your document. The schema should allow an arbitrary number of books, not just the two in your document. It needs to be specific enough to define the types of all values in the example, but can otherwise be permissive. The data that is present in both books in the example should be required for every book.

c) (2 pts) Write a JSON path query that finds the authors of all books written before 1925. The path needs to work for all documents that validate against your schema. Each result of the path should be an author.

Solution:

a)
```
[{"authors":["TP","NG"],
   "title":"GO",
   "year":1990,
   "genre":"C"
  },
  {"authors":["HL"],
   "title":"TKaM",
   "year":1960,
   "genre":"F",
   "tags":["C","CR"]
  }
 ]
```

b)
```
 {"type": "array",
   "items":
    {"type": "object",
     "properties":
        {"authors": {"type": "array", "items": {"type":"string"}},
         "tags":    {"type": "array", "items": {"type":"string"}},
         "title":   {"type": "string"},
         "genre":   {"type": "string"},
         "year":    {"type": "integer"}
        },
     "required": ["authors","title","year","genre"]
    }
 }
```

If one wants to be strict, one should even require that the array for authors have at least one element so it should contain `"minItems":1`.

c) `$[*]?(@.year < 1925).authors[*]`