SOLUTIONS INCLUDED

Databases Exam

TDA357 (Chalmers), DIT621 (University of Gothenburg)

2022-01-12 14:00-18:00

Department of Computer Science and Engineering

Examiner: Jonas Duregård.

Will visit the exam hall at least twice.

Phone: 031 772 1028

Allowed aids: One double sided A4 page of hand-written notes, the notes should be handed in along with your solution. Write your anonymous code on the notes if you wish, but do not write your name on it.

Results: Will be published within three weeks from exam date

Maximum points: 60

Grade limits Chalmers: 27 for 3, 38 for 4, 49 for 5.
Grade limits GU: 27 for G, 45 for VG.

Question 1: SQL Data Definition Language (10 p)

a) Create an SQL database with the following public interface (meaning tables and/or views). You may have additional tables that are not part of the public interface.

Players(id, name, score)

Matches(player1, player2, winner)

The database contains player information for a game, and played matches between players.

Note: Who is designated player1 and player2 in a match is of no importance. You are free to enforce any constraints you like on this.

The following constraints should be satisfied. Use table constraints, views, and if necessary triggers to implement the constraints.

I. The id column has unique values in Players. Player ids should be numbers.
II. All attributes of Match are valid player ids.
III. The winner of a match is always one of the two players in the match.
IV. The score value of a player is the number of matches they have won.
V. Players cannot play matches against themselves.
VI. Any pair of players can play at most one match against one another (regardless of who is player1 and who is player2).

b) Insertions: Show all the inserts required to get the following content in Players:

Players(id, name, score):
(1, 'P1', 1)
(2, 'P2', 0)
(3, 'P3', 0)

Solution 1:

```
a)
CREATE TABLE Player_t(
 id INTEGER PRIMARY KEY, -- I
 name TEXT
 );

CREATE TABLE Matches(
 player1 INTEGER REFERENCES Player_t, -- II
 player2 INTEGER REFERENCES Player_t, -- II
 winner INTEGER REFERENCES Player_t, -- II
 PRIMARY KEY (player1, player2),
 CHECK (player1 < player2), -- easy solution for V and VI
 CHECK (winner = player1 OR winner = player2) -- III
 );

CREATE VIEW Players AS
SELECT id, name, COUNT(winner) AS score -- IV, using * here is acceptable.
FROM Player_t LEFT OUTER JOIN Matches ON winner=id
GROUP BY id, name;

b)
INSERT INTO Player_t VALUES (1, 'P1');
INSERT INTO Player_t VALUES (2, 'P2');
INSERT INTO Player_t VALUES (3, 'P3');

INSERT INTO Matches VALUES (1, 2, 1); -- The 2 can be 3 instead
```

Notes on grading:

Point distribution 8/2.

a) Deduct two points for each missed condition. So at least three correct ones for any points.

b) The most important thing is to realize that this requires insertions into both matches and players. Inserting into a view is acceptable if the obvious intention is for the view to just forward the insert to an underlying table.

Question 2: SQL queries (10 p)

Use the same public interface as in Question 1 for this question (you may assume all the constraints from Question 1 are enforced):

Players(id, name, score)

Matches(player1, player2, winner)

a) Write an SQL query for finding the names of all players with above average score. Only players who have won at least one match should be included in the average.

b) Write an SQL query for finding all pairs of players who have not played against each other yet. Each row should have a pair of player ids, and no pair should appear more than once (regardless of order).

Solution 2:

a)

```
SELECT name FROM Players
WHERE score > (SELECT AVG(score) FROM Players WHERE score>0);
```

```
-- b)
SELECT P1.id, P2.id
FROM Players AS P1, Players AS P2
WHERE P1.id < P2.id
EXCEPT
SELECT player1, player2 FROM Matches
EXCEPT
SELECT player2, player1 FROM Matches -- To ensure order does not matter.
```

Notes on grading:

Point distribution 5/5

a) It's vital to understand that the average is for the whole table (or the parts with positive scores at least). Computing an average for each player using group by or such does not make sense here.

b) Not having the final EXCEPT can be acceptable if it's clear that you make some additional assumption on the contents of Matches other than the ones listed in the first question.

Question 3: Relational Algebra (10 p)

A small database contains observation values. Values are numbers, and the time the observation was performed is recorded (times are also numbers). Additionally, there is a set of intervals, stored as a sequence of times when one interval ends and a new one begins.

IntervalStopTimes(<u>stopTime</u>)

Observation(<u>time</u>, value)

a) Write a relational algebra expression that gives start and stop times for each interval. The start time is always the stop time of the previous interval (so the earliest stopTime-value is actually the start time of the first interval).

Example, if IntervalStopTimes contain four rows with stop times 10, 20, 40 and 55, the result should be:
(10 ,20)
(20, 40)
(40, 55)

b) Assume ***Intervals(startTime, stopTime)*** is the solution to a). You can use ***Intervals*** in your solution without defining it.

Write a relational algebra expression for finding the average and maximum observation value of every interval that has at least one observation. Each observation should be counted for at most one interval. The result should have four columns, for interval start/stop times, average and maximum.

Solution 3:

a) Basic idea, pair all times with all greater times, then use grouping to select the smallest greater time and use that as the stop time.

$\gamma_{\text{startTime, MIN(stopTime)->stopTime}}($

$\sigma_{\text{startTime <stopTime}}($

$\rho_{\text{Start(startTime)}}(\texttt{IntervalStopTimes}) \times \texttt{IntervalStopTimes}$

)

)

b)

$\gamma_{\text{startTime, stopTime, AVG(value) ->average, MAX(value)->max}}($

Observation $\bowtie_{\text{time>=startTime AND time<stopTime}}$ `Intervals`

)

Notes on grading:

Point distribution 5/5

Question 4: ER-modelling (10p)

You will make an ER-diagram for a small part of a communications platform, somewhat similar to Slack that we used in this course (no familiarity with Slack is required).

Here is a list of all required features:

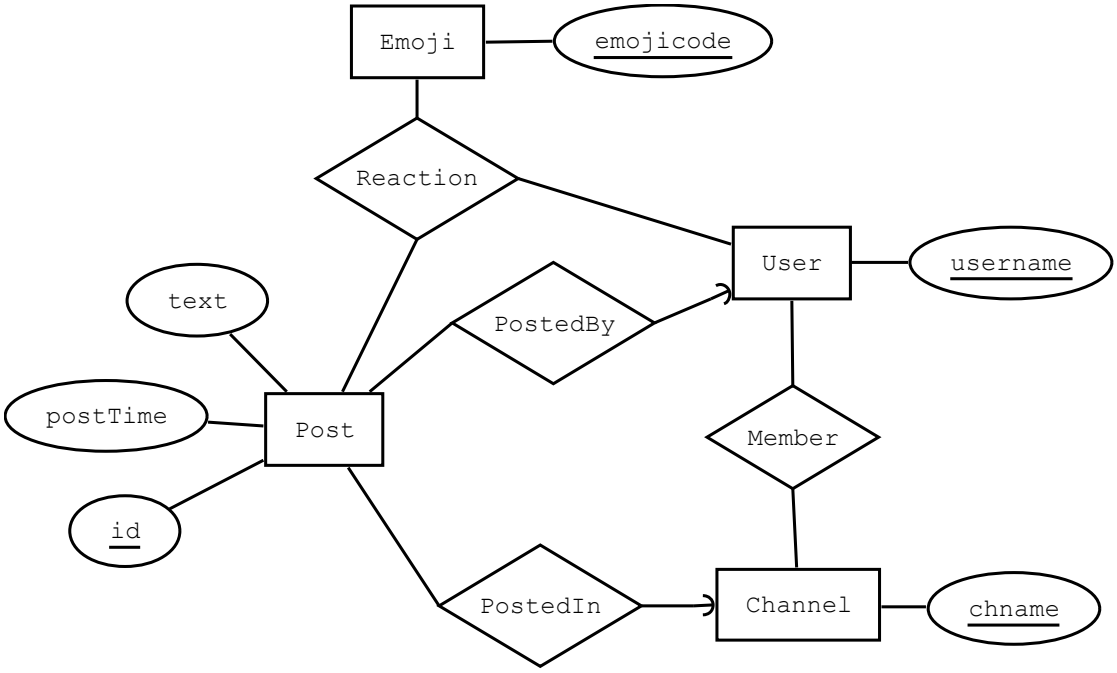
- There are users with usernames.
- There are channels with channel names, and users can be members of channels.
- Users can post messages. Each message is posted in a channel, and contains a text, a time and a unique identification number.
- Users can react to messages with emoji. There is a fixed set of emoji, each identified by a code. Note that a user can react multiple times to the same message with different emoji, but at most once with any specific emoji.

a) Draw an ER-diagram for the domain as described above.

b) Translate your diagram into a relational schema.

Solution 4:

a) Arguably, post should be called “Message”, but naming is not really important.



b)

Users(username)

Channels(chname)

Posts(id, postTime, text, author, channel)

author -> User.username

channel->Channer.chname

Emoji(emojicode)

Reactions(user, post, emoji)

user -> User.username

post-> Post.id

emoji -> Emoji.emojicode

Members(user, channel)

user -> User.username

channel->Channer.chname

Notes on grading:

Point distribution 6/4

a) A *very* common mistake was making Post a weak entity (getting the PK wrong).

b) Graded on how well it corresponds to your diagram (not how similar it is to the solution here).

Question 5: Dependencies and normal forms (10p)

A database containing stars and planets should have these properties:

- Each star has its own unique name.
- Each star has a star type.
- Each planet has a number within the solar system (e.g. Earth would be number three, Mars four).
- Each planet has a set of properties (like "inhabitable" or "gas giant").
- Each planet has a planet type.
- Each planet has a set of satellites.

The relation R contains all attributes of the domain:

```
R(starName, starType, planetNumber, planetProperty, planetType, satellite)
```

a) Identify all FDs of R that violate BCNF, and then normalize R to BCNF.

b) Identify at least one MVD that violates 4NF in your solution to a), and then normalize your schema to 4NF.

Solution 5: Numbering below is not important.

Including satellite->planetNumber (and assuming satellites are unique within solar systems) is acceptable for a).

a)
starName -> starType
starName, planetNumber -> planetType

R1(starName, starType)
R2(starName, planetNumber, planetType)
R3(starName, planetNumber, satellite, planetProperty)

b)
In R3, starName, planetNumber ->> satellite
(and starName, planetNumber ->> planetProperty)

Replace R3 with:

R31(starName, planetNumber, planetProperty)

And

R32(starName, planetNumber, satellite)

GRADING:

Point distribution: 5/5

a) Roughly speaking two points for normalization and three for identifying correct FDs. Having planetNumber -> planetType is acceptable since the question was not 100% clear on this.

b) Roughly 2/3 points for normalization/identification of MVDs. Stating MVDs that are inconsistent with your FDs in a) gives a deduction.

Question 6: Semi-structured data and other topics (10p)

A JSON Schema describes documents for rooms in buildings, their total number of seats and the number of booked seats.

```
{"type":"object",
 "description":"Every key is the name of a building",
 "additionalProperties":{
   "type":"array",
   "description":"A list of rooms in the building",
   "items":{
     "type":"object",
     "description":"Rooms with names, nr. of total/booked seats",
     "properties":{
       "name":{"type":"string"},
       "totalSeats":{"type":"integer"},
       "bookedSeats":{"type":"integer"}
     },
     "required":["name","totalSeats","bookedSeats"]
   }
 }
}
```

a) Write a JSON-document encoding the following information: One building named “B1” having two rooms, R1 and R2. Fill in any additional information as needed for your document to validate against the given JSON schema.

b) Write a JSON Path for finding the names of all rooms in the building "EDIT" that have at least one free (non-booked) seat.

c) Extend the JSON Schema so that bookedSeats can optionally have the string value "unbookable". You only need to write the parts of the schema that you modify.

Solution 6:

a)

```
{"B1":[{"name":"R1", "totalSeats": 3, "bookedSeats":2}
      ,{"name":"R2", "totalSeats": 3, "bookedSeats":2}]}
```

b) Anything that looks vaguely like this:

```
$.EDIT[?(@.totalSeats > @.bookedSeats)].name
```

c) Something like this (anyOf also works, and the constant can be done in other ways). Specifying "type":"string" is fine, but not necessary (and not sufficient in itself, since only the string "unbookable" should be accepted).

```
"bookedSeats":{oneOf[{"type":"integer"}, {"const":"unbookable"}]}
```

Notes on grading:

Point distribution: 3/4/3

a) Important things: The top level being an object. The value being an array. Elements being objects. Elements having correct keys and types.

c) Main points is to identify what part of the schema to alter, and that it needs to have two alternative schemas. Remembering the keyword "const" is not necessary, but just specifying type string is not sufficient (since that allows all strings).