Databases

TDA357/DIT622/DIT621 (4.5 hec)

Responsible: Ana Bove, tel: 1020

Friday 15th of March 2024, 8:30–12:30

Total: 60 points
TDA357/DIT622: $\geqslant$ 27: 3, $\geqslant$ 38: 4, $\geqslant$ 49: 5
DIT621: $\geqslant$ 27: G, $\geqslant$ 45: VG

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

The exam has 5 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, Constraints and RA (15.5 pts)

A cleaning company that was founded in year 2000 has the following information about its employees, its clients, and the assignment of employees to the client's properties that are to be cleaned.

Staff (id, name, salary, started)
Client (nr, name, address, area, frequency)
Assignment (staff, client, address)
staff $\rightarrow$ Staff.id
(client, address) $\rightarrow$ Clients.(nr, address)

Staff's id identifies the employee, of whom one also has the name and salary. Started is the year when the employee has started working in the company.

Client's numbers identify/determine the name of the client. Each client might have several properties in the database of the company. Each property has an area (in m^2) and a frequency in which the property is to be cleaned (once, twice or four times a month). Observe that the address doesn't identify a property!

The assignments show which employee is assigned to clear which of the client's properties. It is worth mentioning that there might be properties in the database of the company that are currently not being cleaned (hence they are not currently assigned an employee that would clean it) despite the frequency might suggest otherwise. The properties that are currently being cleaned are called *active*.

a) (4pts) Define SQL tables for the information above. You can define as many tables as you think reasonable. Make sure to define appropriate types and constraints, including possible missing ones in the schema.

Hint: A direct translation from the schema above into tables might not be the best way to go since the given schema has some redundancies. Read the description above carefully!

b) (2.5+3pts) Write an SQL query and a RA expression that output the name of each employee and the total number of properties the employee is assigned to. Even employees that are currently not assigned to clean any properties should be part of the output. Order the output in descending order with respect to the number of properties to clean.

c) (3+3pts) Write an SQL query and a RA expression that outputs all the active properties. The output should contain the identification (nr) and the name of the clients (that have at least an active property), and the address(es) of the property(properties) in question.

Solution:

```
CREATE TABLE Staff (
  id INT PRIMARY KEY,
  name TEXT NOT NULL,
  salary INT NOT NULL CHECK (salary > 0),
  started INT NOT NULL CHECK (started >= 2000)) ;

CREATE TABLE ClientsInfo (
  nr INT PRIMARY KEY,
  name TEXT NOT NULL) ;

CREATE TABLE CProperties (
  nr INT REFERENCES ClientsInfo,
  address TEXT,
  area INT NOT NULL CHECK (area > 0),
  frequency INT NOT NULL CHECK (frequency IN (1,2,4)),
  PRIMARY KEY (nr, address)) ;

CREATE TABLE Assignments (
  staff INT NOT NULL REFERENCES Staff,
  client INT,
  address TEXT,
  PRIMARY KEY (client, address),
  FOREIGN KEY (client, address) REFERENCES CProperties) ;
```

```
-- Nr of properties per employee
SELECT name, COUNT(address) AS sum
FROM Staff LEFT JOIN Assignments ON id = staff
GROUP BY id, name
ORDER BY sum DESC;


-- Active properties
CREATE OR REPLACE VIEW ActiveProp AS
SELECT nr, name, address
FROM ClientsInfo JOIN Assignments
WHERE nr = client;
```

RA for number of properties per employee:

$$\tau_{-\mathsf{sum}}(\pi_{\mathsf{name,sum}}(\gamma_{\mathsf{id,name,COUNT(address)}\rightarrow\mathsf{sum}}(\mathsf{Staff} \bowtie^{\mathsf{OL}}_{\mathsf{id}=\mathsf{staff}} \mathsf{Assignments})))$$

RA for active properties:

$$\pi_{\mathsf{nr,name,address}}(\mathsf{ClientsInfo} \bowtie_{\mathsf{nr}=\mathsf{client}} \mathsf{Assignments}))$$

2 Views and Triggers (14 pts)

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

Staff (<u>id</u>, name, salary, started)
Client (<u>nr</u>, name, <u>address</u>, area, frequency)
Assignment (staff, <u>client</u>, <u>address</u>)
staff → Staff.id
(client, address) → Clients.(nr, address)

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

a) (4pts) Some clients are considered *premium* clients. To become a premium client the person needs to have in the company's database, either a property with an area of at least 150 m^2, or at least 3 properties. These properties could be active or not, they count either way.

Write an SQL view *PremiumClients* that outputs the identification and name of all the premium clients, and the sum of the areas of his/her properties (active or not) in the database.

b) (4.5pts) Produce the monthly bill for cleaning the (active) properties.
The output should contain the name of the client, and the total amount to pay for all the (active) properties the company is cleaning for that client. A client should only appear in the output if he/she has at least an active property.

The amount to pay for cleaning a particular property depends on the property's area, the salary of the employee that cleans the property and the amount of times per month that the property is cleaned (frequency). Each time the property is cleaned, the client needs to pay 1% of the salary of the employee that cleans that property for every 30 m^2 of the area of the property. For the bill, the area of the property is round up to the closest multiple of 30. So for example, a property of 80 m^2 cleaned twice a month by an employee whose salary is 50 costs 3 x 0.5 x 2 to clean per month: 3 since 3 x 30 = 90 m^2 (the closest round up), 0.5 is 1% of the salary, and 2 since the property is cleaned twice a month.

Note: Here you can assume that you have a correct implementation of the active properties query (from 1c) in the form of a view called *ActiveProp* if you wish to use it.

Hint 1: You can use the function CEIL to round up (CEIL(15/30::FLOAT) gives 1).
Hint 2: Start by computing the cost of one property, and take care of the total per client afterwards.

c) (5.5pts) If a premium client wishes to start cleaning yet another property, only experienced employees should be assigned to the job. An employee is considered experienced if he/she has worked at least 10 years in the company.

Observe that you don't need to look after an experienced employee to assign the job to, only make sure that only experienced employees are assigned to the job!

Note: Here you can assume that you have a correct implementation of the view *PremiumClients* (from 2a) that identifies premium clients if you want to use it.

Hint: The expression DATE_PART('year', CURRENT_DATE) gives the current year.

Solution:

```
-- Premium clients
CREATE OR REPLACE VIEW PremiumClients AS
With Premium AS
(SELECT nr
FROM CProperties
GROUP BY nr
HAVING COUNT(*) >= 3
UNION
SELECT nr
FROM CProperties
WHERE area >= 150)
SELECT nr, name, SUM(area)
FROM Premium NATURAL JOIN CProperties NATURAL JOIN ClientsInfo
GROUP BY nr, name;

-- Monthly bill
CREATE OR REPLACE VIEW Bill AS
WITH CostProp AS
(SELECT nr, AP.name,
        CEIL(area/30::FLOAT)*frequency*salary/100 AS cost
FROM (ActiveProp NATURAL JOIN CProperties) AS AP
      JOIN Assignments AS A
           ON (nr = client AND A.address = AP.address)
      JOIN Staff ON id = staff)
SELECT name, SUM(cost) AS total
FROM CostProp
GROUP BY nr, name;


-- New assignment
CREATE OR REPLACE FUNCTION check_assignment() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.client IN (SELECT nr FROM PremiumClients) AND
       (SELECT started FROM Staff WHERE id = NEW.staff) + 10 >
            DATE_PART('year', CURRENT_DATE)
    THEN RAISE EXCEPTION 'Cleaner not experienced enough';
    END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER NewAssig
  BEFORE INSERT ON Assignments
  FOR EACH ROW
  EXECUTE FUNCTION check_assignment();
```

3 ER Modelling (9 pts)

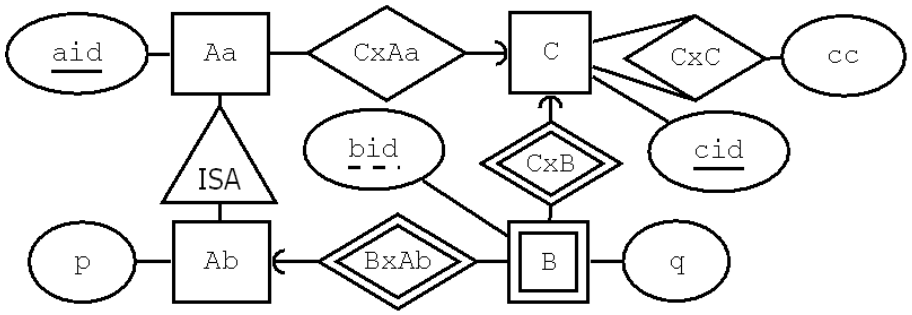
a) (5.5pts) Make an ER-diagram for a database of online chess games, where players can play games as well as observe games played by other people. The database should satisfy these conditions:

- Players have unique user names, and each player has a score. Players can subscribe to other players, for example to be able to observe other player's games.
- Some games are currently active and some have already finished.
- Each game has two players, designated "white" and "black" respectively. Games don't have names, so they need to be identified in some other way (like a game number or similar).
- The starting time of every game should be in the database (a single timestamp value for each game).
- All games have a record of performed moves. A move is from one position to another at a given time (counted from the start of the game). There are always 64 positions on the board identified by coordinates from A1 to G8 (so a move could be something like "A4 to C3 at 43.12 seconds"). The move history should not store what piece is being moved or which player it belongs to, since that can be computed from the history and a fixed starting state. There cannot be two moves performed at exactly the same time in a game (but there can be in different games).
- Each active game also has a current state, meaning which pieces are currently in what positions. By the rules of chess there can only be one piece in any specific position in a particular game (different games could have different pieces in the same position). Pieces are just textual names like queen or pawn, so a part of a game state could be something like "pawn at position A2".

Note: There is a slight redundancy between storing all the moves and the current game state (since the latter can be calculated from the former) but that is intended.

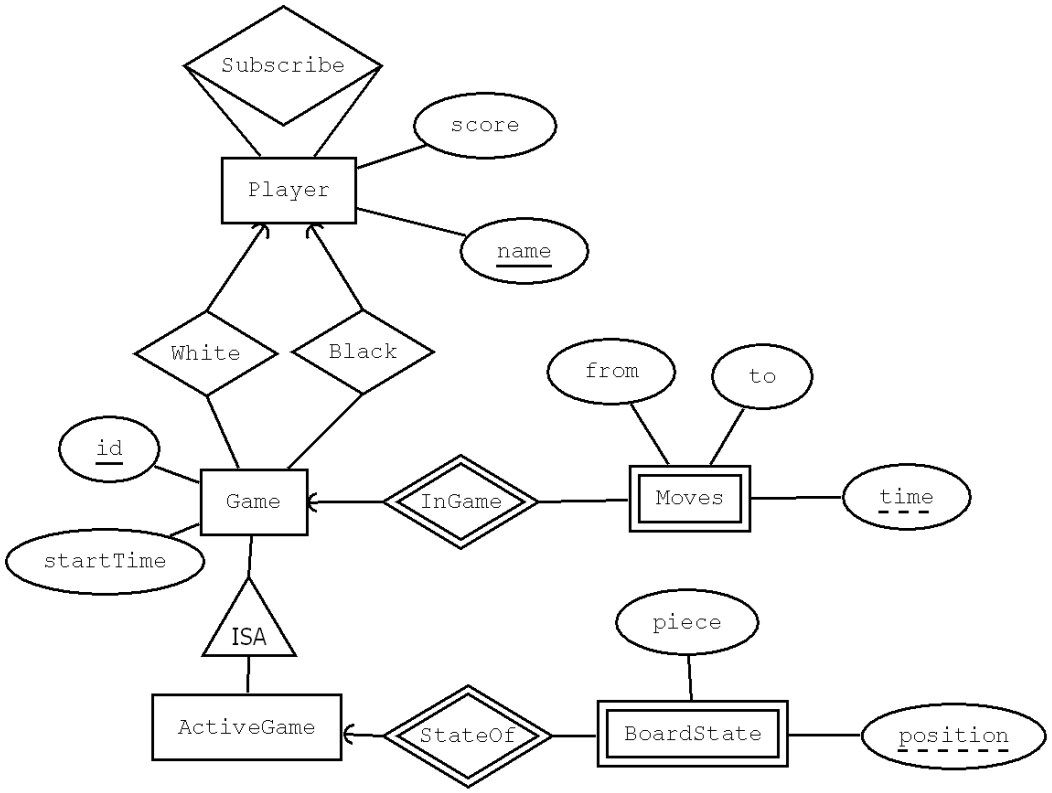
Hint: Don't think too hard about special moves of chess, rules not stated here or the format of the positions (just use text for them)

b) (3.5pts) Translate the following diagram into a relational schema.



Solution:

a)



b)

Aas (<u>aid</u>, cid)
Abs (<u>aid</u>, p)
    aid → Aas.aid
Cs (<u>cid</u>)
CxCs (<u>cid1</u>, <u>cid2</u>, cc)
    cid1 → Cs.cid
    cid2 → Cs.cid
Bs (<u>cid</u>, <u>aid</u>, <u>bid</u>, q)
    aid → Abs.aid
    cid → Cs.cid

4 Functional and Multivalued Dependencies (11.5pts)

Consider the schema R(a,b,c,d,e) and the following table:

a	**b**	**c**	**d**	**e**
0	0	1	1	2
0	0	2	1	1
0	1	1	1	3
1	1	3	2	2
1	1	3	2	1

a) (3pts) For each of the following FD, explain whether it holds or not. Here you are NOT asked for a yes/no answer but for a *justification* (referring to the data) on why the corresponding FD holds or not. (Note that just a yes/no answer gives no points!)

a b → d
c → e
b e → d

b) (2pts) Decompose your schema into a BCNF one using just the FDs from the list above that you found hold. Apply them in the same order as they are given in the list/above. Make sure to indicate primary and foreign (references) keys in your final schema.

c) (2pts) After you have decomposed the schema, split the table above into the corresponding tables for the schema you have obtained from the decomposition.

d) (1.5pts) Does the MVD a b ↠ c hold? If yes, explain why. If not, add the necessary information in the table(s) for the MVD to hold.

e) (1.5+1.5pts) Decompose the schema from b) into a 4NF one using the MVD a b ↠ c, and split the tables accordingly. Here you need to assume that the rows that needed to be added in part d) (if any) in order to make a b ↠ c hold are now part of the table. State the primary keys after the decomposition.

Solution:

a) **a b → d:** This FD holds. Rows 1 and 2 in the table have the same values in both a and b and they also have the same value in d. The same happens with rows 4 and 5.

c → e: This FD doesn't hold. For example, rows 1 and 3 in the table have the same value in c but the e-value differs.

b e → d: There are no two rows with the same values in both b and e, so the FD trivially holds.

b) When decomposing after a b → d we get
R1(a,b,d)
R2(a,b,c,e)
(a,b) → R1.(a,b)
No schema has now attributes b, e and d, so there is nothing else to do.

c) The tables become now:

a	**b**	**d**
0	0	1
0	1	1
1	1	2

a	**b**	**c**	**e**
0	0	1	2
0	0	2	1
0	1	1	3
1	1	3	2
1	1	3	1

d) No, for a b ↠ c to hold we need to add the rows (0,0,1,1) and (0,0,2,2) to the table for R2.

e) R2 needs to be decomposed into R21(a,b,c) and R22(a,b,e).

The table for R2 (with the two extra rows) is now split into the following two tables:

a	**b**	**c**
0	0	1
0	0	2
0	1	1
1	1	3

a	**b**	**e**
0	0	2
0	0	1
0	1	3
1	1	2
1	1	1

5 JSON (10 pts)

A riding school has both horses and ponnies, and information about them both.

For ponnies one must have their name, class (which is either "A", "B", "C", or "D"), and age (in years). If possible, also their country of origin.

For horses one must have their name, their height ("mankhöjd"), and age (in years). If possible, also their country of origin.

Here is a table with information about two ponnies and two horses:

name	**class/height**	**age**	**origin**
Valle	D	24	
Granit	159	19	
Maxi	D	8	Hungary
Holstein	167	18	Denmark

a) (3pts) Create a JSON **object** (that is, not an array) containing the information about the ponnies and horses that is contained in the table above.

b) (5pts) Define a JSON schema that validates the document you created. The schema should ensure the following:

- All types are correct (for example, changing any string into an integer or vice versa should invalidate the document).
- Empty strings shouldn't be allowed, and heights and ages cannot be negative.
- Ponnies should contain only the information related to ponnies, and horses only the information related to horses. That is, no other property than those stated above should be allowed.
- The "must have" properties should always be present.

Hint: Use definitions and $ref for the frequently recurring sub-schemas.

c) (2pts) Write a JSON Path expression for finding the age of all D-ponnies in the riding school.

Solution:

a)
```
{"Valle":{"class":"D",
          "age":24},
 "Granit":{"height":159,
           "age":19},
 "Maxi":{"class":"D",
         "age":8,
         "origin":"Hungary"},
 "Holstein":{"height":167,
             "age":18,
             "origin":"Denmark"}
}
```

One could consider having the name of the ponny/horse as one of the properties but this would required to have “artificial” keys for each ponny/horse (something like “ponny1”, “ponny2”, ..., “horse1”, “horse2”, ...) .

Another possibility would be to have two keys, “ponnies” and “horses”, and then an array with the corresponding information.

b) The important thing is to have the right definitions for ponnies and for horses. Those for non-empty strings or positive numbers are good to have but not needed.

If other solutions were given for a), the schema needs to be changed accordingly.

```
{"type": "object",
  "definitions": {
    "nonemptystr" : {"type": "string", "minLength": 1},
    "posint" :{"type":"integer", "minimum": 1},
    "ponny":{"type":"object",
             "properties":{
               "class":{"type":"string",
                        "enum":["A","B","C","D"]},
               "age":{"$ref": "#/definitions/posint"},
               "origin":{"$ref": "#/definitions/nonemptystr"}
             },
             "required": ["class","age"],
             "additionalProperties": false
            },
    "horse":{"type":"object",
             "properties":{
               "height":{"$ref": "#/definitions/posint"},
               "age":{"$ref": "#/definitions/posint"},
               "origin":{"$ref": "#/definitions/nonemptystr"}
             },
             "required": ["height","age"],
             "additionalProperties": false
            }
  },
  "additionalProperties":
     {"oneOf":[{"$ref": "#/definitions/ponny"},
               {"$ref": "#/definitions/horse"}]}
}
```

c) Both paths would work for this object:

```
'$.*?(@.class == "D").age'
'$.**?(@.class == "D").age'
```