Databases

TDA357/DIT621 (4.5 hec)

Responsible: Ana Bove, tel: 1020

Friday 18th of March 2022, 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! :)
Good luck!

1 SQL and Constraints (9.5 pts)

A riding school has the following relational schema for the information about their horses, the rider (Clients) with their level of riding knowledge, the different lessons they offer (Groups) with their day, time and level, and the clients that ride in each of the groups (InGroups):

Horses (name, age, jump)
Clients (id, name, level)
Groups (day, time, level, jump)
InGroup (day, time, rider)
  (day, time) $\rightarrow$ Groups.(day, time)
  rider $\rightarrow$ Clients.id

where "jump" in horses indicates whether a horse can take part in a jump lesson or not, and "jump" in groups whether the group is a dedicated jumping group or a normal one.

a) (4 pts) Define SQL tables for the relational schema above. The following constraints should be part of your implementation:

- Lessons (Groups) run between Monday and Saturday, and only at 14, 15, 16, 18, 19 or 20 o'clock on week days but at 9, 10 or 11 on Saturdays.
- Possible level for both riders and non-jumping groups are 1 to 9. For jumping groups the level information is irrelevant (and hence could be any number or even empty).
- Make sure the age of a horse gets a positive number, and do not allow NULL values in the attributes unless otherwise stated.

b) (2.5 pts) To keep track of the daily planning (which horse each rider will have on each of the lessons of the day), a new table is added with the following (partial) relational schema:

Planning (year, week, day, time, rider, horse)

Define an SQL table for Planning. Make sure to define a good primary key and any other necessary additional constraint (foreign keys, unique constraints, etc) including reasonable ones for year and week.

c) (3 pts) Write an SQL query that for a particular year (say 2021), lists the number of times (in descending order) that each of the clients rides each of the horse. Only clients and horses that rode and were ridden during that year should be part of the answer. The output should have 3 columns: one for the name of the rider, one for the name of the horse, and one for the actual number.

Solution:

```
CREATE TABLE Horses (
  name CHAR(10) PRIMARY KEY,
  age INT NOT NULL CHECK (age > 0),
  jump BOOLEAN NOT NULL );

CREATE TABLE Clients (
  id INT PRIMARY KEY,
  name TEXT NOT NULL,
  level INT NOT NULL CHECK (level > 0 AND level < 10) );

CREATE TABLE Groups (
  day TEXT CHECK (day in ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat')),
  time INT CHECK ((time in (14, 15, 16, 18, 19, 20) AND
                   day in ('Mon', 'Tue', 'Wed', 'Thu', 'Fri')) OR
  (time in (9, 10, 11) AND day in ('Sat'))),
  level INT,
  jump BOOLEAN NOT NULL,
  PRIMARY KEY (day, time),
  CHECK ((level > 0 AND level < 10) OR JUMP) );

CREATE TABLE InGroup (
  day TEXT,
  time INT,
  rider INT REFERENCES Clients,
  PRIMARY KEY (day, time, rider),
  FOREIGN KEY (day,time) REFERENCES Groups );
```

```
CREATE TABLE Planning (
  year INT CHECK (year > 1968),
  week INT CHECK (week > 0 AND week <= 53),
  day TEXT,
  time INT,
  rider INT NOT NULL REFERENCES Clients,
  horse CHAR(10) REFERENCES Horses,
  FOREIGN KEY (day,time, rider) REFERENCES InGroup,
  PRIMARY KEY (year, week, day, time, horse),
  UNIQUE (year, week, day, time, rider) );

SELECT Clients.name, horse, COUNT(*) AS Nr
FROM Planning, Clients
WHERE year = 2021 AND id = rider
GROUP BY id, horse
ORDER BY Clients.name, Nr DESC;
```

2 More SQL and Relational Algebra (10 pts)

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

Horses (<u>name</u>, age, jump)
Clients (<u>id</u>, name, level)
Groups (<u>day</u>, <u>time</u>, level, jump)
InGroup (<u>day</u>, <u>time</u>, <u>rider</u>)
(day, time) → Groups.(day, time)
rider → Clients.id
Planning (year, week, day, time, rider, horse)

with the primary key and other constraints you defined for Planning.

a) (2.5 + 2.5 pts) Write an SQL query and a relational algebra expression that for a particular year (say 2021), computes the average number of lessons a horse is ridden per week. To compute the average assume the year has 52 weeks. Present the result in decreasing order. The output should have 2 columns: one for the name of the horse, and one for the actual number.

Solution:

The rounding and casting below are not required for full points.

```
SELECT horse, ROUND(COUNT(*)/52 :: NUMERIC,2) AS Average
FROM Planning
```

```
WHERE year = 2021
GROUP BY horse
ORDER BY Average DESC;
```

$$\tau_{-\mathsf{average}}(\pi_{\mathsf{horse},\mathsf{average}}(\gamma_{\mathsf{horse},\mathsf{COUNT}(*)/52\rightarrow\mathsf{average}}(\sigma_{\mathsf{year}=2021}\ \mathsf{Planning})))$$

or even better

$$\tau_{-\mathsf{average}}(\gamma_{\mathsf{horse},\mathsf{COUNT}(*)/52\rightarrow\mathsf{average}}(\sigma_{\mathsf{year}=2021}\ \mathsf{Planning}))$$

b) (2.5 + 2.5 pts) Write an SQL query and a relational algebra expression that for a particular year (say 2021), lists all the horses which were ONLY ridden in advanced groups (those with level of at least 6). Recall that jumping groups don't actually have a level (even if they might have a level information on the table!).

Solution:

```
SELECT horse
FROM Planning NATURAL JOIN Groups
WHERE year = 2021 AND level > 5 AND NOT jump
EXCEPT
SELECT horse
FROM Planning NATURAL JOIN Groups
WHERE year = 2021 AND level < 6 AND NOT jump;
```

$$\delta(\pi_{\mathsf{horse}}(\sigma_{\mathsf{year}=2021\wedge\mathsf{level}>5\wedge\neg\mathsf{jump}}(\mathsf{Planning}\bowtie\mathsf{Groups})) - \pi_{\mathsf{horse}}(\sigma_{\mathsf{year}=2021\wedge\mathsf{level}<6\wedge\neg\mathsf{jump}}(\mathsf{Planning}\bowtie\mathsf{Groups})))$$

3 Views and Triggers (9 pts)

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

Horses (name, age, jump)
Clients (id, name, level)
Groups (day, time, level, jump)
InGroup (day, time, rider)
(day, time) → Groups.(day, time)
rider → Clients.id
Planning (year, week, day, time, rider, horse)

with the primary key and other constraints you defined for Planning.

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

a) (3.5 pts) Keep track of the monthly fee for each of the clients. Since name does not identify clients make sure to also have the id as part of the output and not just the name of the client. So your output will have 3 columns: the id and name of the client, and the fee they need to pay.
Taking part in a jumping group costs 1500 kr per month, riding in an advanced group (those with level of at least 6) costs 1200 kr per month, otherwise the price is 1000 kr per month. Assume clients belong to the same group during the whole month.

b) (5.5 pts) Add a new rider to a group.
Jumping groups cannot have more than 5 rider, normal groups cannot have more than 8 riders. For a jumping group, the rider should be advanced (at lest level 6), otherwise the level of the rider should be at least as high as the level of the group.

Solution:

```
-- View: Monthly bill
CREATE OR REPLACE VIEW Billing AS
WITH Prices AS
 (SELECT day, time, 1500 AS price
  FROM Groups
  WHERE jump
  UNION
  SELECT day, time, 1200 AS price
  FROM Groups
  WHERE (NOT jump AND level > 5)
  UNION
  SELECT day, time, 1000 AS price
  FROM Groups
  WHERE (NOT jump AND level < 6)
```

```
 )
SELECT id, name, SUM(price) AS Fee
FROM Clients, Prices NATURAL JOIN InGroup
WHERE id = rider
GROUP BY id;

-- Function and trigger: Insert a person in a group
CREATE OR REPLACE FUNCTION add2group() RETURNS TRIGGER AS $$
DECLARE
  nr INT;
  grouplevel INT;
  riderlevel INT;
  isjump BOOLEAN;
BEGIN
 nr = (SELECT COUNT(*) FROM InGroup
       WHERE day = NEW.day AND time = NEW.time);
 isjump = (SELECT jump FROM Groups
           WHERE day = NEW.day AND time = NEW.time);
 grouplevel = (SELECT level FROM Groups
               WHERE day = NEW.day AND time = NEW.time);
 riderlevel = (SELECT level FROM Clients WHERE id = NEW.rider);
 IF isjump
 THEN IF nr >= 5 OR riderlevel < 6
      THEN RAISE EXCEPTION 'cannot insert';
      END IF;
 ELSE IF nr >= 8 OR riderlevel < grouplevel
      THEN RAISE EXCEPtION 'cannot insert';
      END IF;
 END IF;
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS addTOgroup ON InGroup;

CREATE TRIGGER addTOgroup
  BEFORE INSERT OR UPDATE ON InGroup
  FOR EACH ROW
  EXECUTE FUNCTION add2group();
```

4 ER Modelling (11 pts)

A company needs help designing a database for managing the inventory and sale prices of a chain of stores.

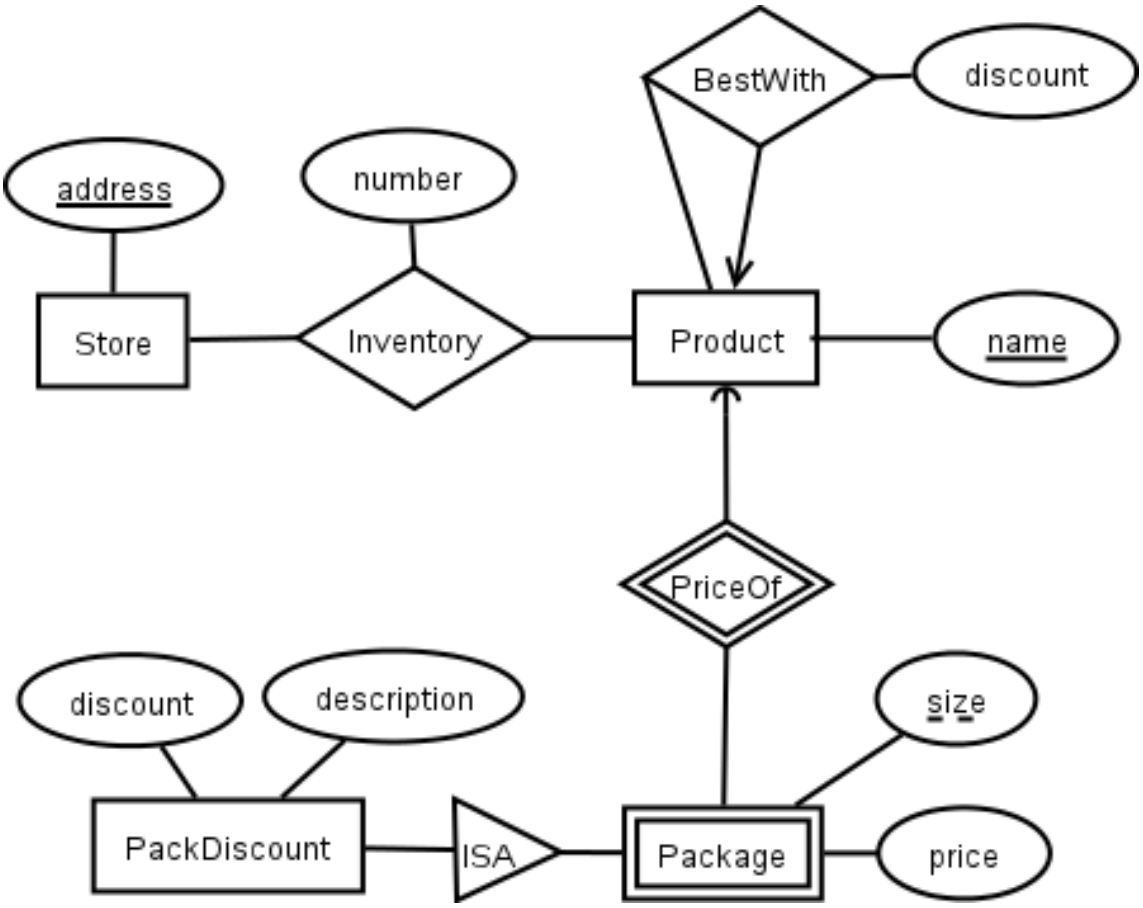
The description of the domain is as follows:

- The chain contains multiple stores, each identified by its address.
- Every product sold by the chain has its own name.
- Every store has an inventory of products in the store, that is, how many items of each product they currently have in stock.
- Products are sold in packages of different sizes. Different products are packaged in different sizes, for example one product could be sold in 5-packs and 10-packs, and another product only in single packs (size 1).
- Each package size of a product has its price, so a 5-pack of a product could cost 20 and a 10-pack of the same product cost 35. The price is the same in all stores.
- Discounts could be applied to particular size packages of a product, for example the 5-packs of a certain product could have a discount but not all other size packages for that product. Each discount should have a reduction and a description such as 25% and "Christmas sale". There cannot be multiple discounts on the same pack size of a product but different size packages of the some product could have their own discount.
- Some products have one other product as "best bought together with" that clients are recommended to also buy, for example when the client wants to buy a particular model of a Mac then he/she is recommended to also buy a certain external screen to maximise the experience. In these cases, even an extra discount for the second product is offered.

a) (6 pts) Construct an ER-diagram based on the description above. Be restrictive about making assumptions not stated in the description.

b) (5 pts) Translate the ER-diagram into a schema. This should be an exact translation of the solution to a) without any additional constraints.

Solution:



Store (address)
Product (name)
Inventory (store, product, number)
  store → Store.address
  product → Product.name
BestWith (product, bestwith, discount)
  product → Product.name
  bestwith → Product.name
Package (product, size, price)
  product → Product.name
PackDiscount (product, size, discount, description)
  (product, size) → Package.(product, size)

5 Functional and Multivalued Dependencies (11pts)

Consider this domain relation for student projects:

R(student, teacher, projectTitle, score, publicComment, hiddenComment)

The attributes student, teacher and projectTitle are unique identifiers for students, teachers and projects respectively. The score attribute is a number, while publicComment and hiddenComment are text values.

The domain has these requirements:

- Each student has a single project of their own.
- Teachers can assign a score to projects; multiple teachers can score each project, but a teacher can only give a single score to any specific project.
- Teachers can add two kinds of comments to projects: public and hidden. The hidden comments can only be seen by teachers. Any teacher can assign any number of both types of comments to any project.

Note: Your solution should be based on the description above. Avoid making additional assumptions based on real life experience.

a) (3 pts) List three functional dependencies which cannot be derived from each other.

b) (5 pts) Normalise R to BCNF. If you use any derived functional dependencies in the process (that is, not just the dependencies you listed above but dependencies that can be derived from those listed above), state them here with some information on how you derived them.

Describe the intermediate states so we can follow your normalisation process. Mark very clearly which are the (final) relations in the BCNF schema.

Do not forget to mark primary keys and any additional secondary keys (unique constraints) you identify.

Hint: If you only perform a single normalisation step, your result is probably not correct. Consider carefully which FD you start the process with or possible derived FD!

c) (1 pt) Identify at least one MVD that violate 4NF in your solution to b).

d) (2 pts) Further normalise your solution to 4NF. You do not need to repeat the whole schema, just the parts that are decomposed. Mark primary keys in the result.

Solution:

a)

student $\rightarrow$ projectTitle
projectTitle $\rightarrow$ student
projectTitle teacher $\rightarrow$ score

b) If one starts the normalisation with the FD
projectTitle → student
(and obtain R1 with it) then one quite straightforwardly gets the following schema (efter a couple of steps):

R1(<u>projectTitle</u>, student)
  UNIQUE student
R2(<u>projectTitle</u>, <u>teacher</u>, score)
R3(<u>projectTitle</u>, <u>teacher</u>, <u>publicComment</u>, <u>hiddenComment</u>)

If one instead starts with the FD
student → projectTitle
then we need the derived FD
student teacher → score
to continue the normalisation process and obtain R2 and R3.

R1(<u>student</u>, projectTitle)
  UNIQUE projectTitle
R2(<u>student</u>, <u>teacher</u>, score)
R3(<u>student</u>, <u>teacher</u>, <u>publicComment</u>, <u>hiddenComment</u>)

Your solution needs to even contain the intermediate steps in the process and some explanation on how one obtains the derived FD.

c) projectTitle teacher ↠ publicComment

Alternative: student teacher ↠ publicComment

d)

R31(<u>projectTitle</u>, <u>teacher</u>, <u>hiddenComment</u>)
R32(<u>projectTitle</u>, <u>teacher</u>, <u>publicComment</u>)

6 JSON (9.5 pts)

The following is part of an inventory of computer components at a webstore, divided into categories (such as GPUs, CPUs, ...). For each product, the information about manufacturer, model, full price and current discount for the upcoming sale is stored.

...

GPUs				
	Nvidia	3070GTX	11890kr	5%
	AMD	RX6800	13490kr	10%
CPUs				
	Intel	i7-12700K	4590kr	30%
	AMD	Ryzen 5600X	2739kr	12%
Memory				
	Corsair	LPX 32GB	1489kr	15%
	Kingston	Fury 64GB	2999kr	20%
	Kingston	Fury 128GB	4999kr	10%

...

a) (2.5 pts) Write a JSON document that encodes the data above. Make sure to keep the same structure as the data above.
To avoid writing a big document, it is enough that the inventory in your document fully contains just the first category (GPUs with all its products), with "..." to indicate the rest.

b) (4.5 pts) Write a JSON schema that describes your encoding of the data that matches the JSON document you provided in a). The schema needs to (at least) specify types of every JSON value in your encoding and the required properties of the objects.

c) (2.5 pts) Write a JSON path query that finds those products that cost 2500kr or less after the given discount has been applied.

Solution:

a) This is the whole data (as an array):

```
```json
[   ...,
    {"category": "GPUs",
     "products": [
         {"manufacturer": "Nvidia",
          "model":  "3070GTX",
          "price": 11890,
          "discount": 0.05
```
```

```
            },
            {"manufacturer": "AMD",
             "model":  "RX6800",
             "price": 13490,
             "discount": 0.10
            }
        ]},
       {"category": "CPU",
        "products": [
            {"manufacturer": "Intel",
             "model":  "i7-12700k",
             "price": 4590,
             "discount": 0.3
            },
            {"manufacturer": "AMD",
             "model":  "Ryzen 5600X",
             "price": 2739,
             "discount": 0.12
            }
        ]},
       {"category": "Memory",
        "products": [
            {"manufacturer": "Corsair",
             "model":  "LPX 32GB",
             "price": 4590,
             "discount": 0.15
            },
            {"manufacturer": "Kingston",
             "model":  "Fury 64GB",
             "price": 2999,
             "discount": 0.20
            },
            {"manufacturer": "Kingston",
             "model":  "Fury 128GB",
             "price": 4999,
             "discount": 0.10
            }
        ]},
        ...
]
```

but it's OK if you just fully write the first category and then '...' as stated in the question. Observe that your category needs to still be one of the element of an array!

```json
[   ...,
    {"category": "GPUs",
     "products": [
         {"manufacturer": "Nvidia",
          "model":  "3070GTX",
          "price": 11890,
          "discount": 0.05
         },
         {"manufacturer": "AMD",
          "model":  "RX6800",
          "price": 13490,
          "discount": 0.10
         }
     ]},
    ...
  ]
```

b) The important bits here are that the outer structure is an array, and the discount and price are numbers and not strings.

```json
{
  "definitions": {
    "category": {
      "type": "object",
      "properties": {
        "category": {"type": "string"},
        "products": {
          "type": "array",
          "items": {"$ref": "#/definitions/product"}
        }
      },
      "required": ["category","products"],
      "additionalProperties": false
    },
    "product": {
      "type": "object",
      "properties": {
        "manufacturer": {"type": "string"},
        "model": {"type": "string"},
        "price": {"type": "integer"},
        "discount": {"type": "number"}
```

```
      },
      "required": ["manufacturer","model","price","discount"],
      "additionalProperties": false
    }
  },
  "type": "array",
  "items": {"$ref": "#/definitions/category"}
}
```

c) Here the solution depends on what the type of discount is.
If number as above then we have

```
''strict $.**?(@.price *(1- @.discount) <= 2500)''
```

If you model the discount as an integer, then you need to divide by 100 as in:

```
''strict $.**?(@.price *(1- (@.discount/100)) <= 2500)''
```