Databases

TDA357/DIT622 (4.5 hec)

Responsible: Ana Bove, tel: 1020

Thursday 28th of August 2025, 14:00–18:00

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

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 (8.5 pts)

Here we work in the domain of artists and concerts efter the following (incomplete) relational schema:

Artists (name, contact, price, type)
ConcertHalls (name, address, nrseats)
Concerts (date, artist, arena, address, price)
artist $\rightarrow$ Artists.name
(arena, address) $\rightarrow$ ConcertHalls.(name, address)

Artists are identified by their name, have some contact information and a price they charge per concert. An artist can be a solo artist or a band; this is the *type* of the artist.

Different concert halls could have the same name, hence their addresses are also needed for their identification. Each concert hall has a certain number of seats which indicates the maximum number of tickets that can be sold for a concert in that particular concert hall.

A concert is given by a particular artist in a particular concert hall at a particular date. *All* tickets for that particular concert have the same price. An artist can only give a concert per day and a concert hall can only have a concert per day.

a) Without giving the complete tables:

i) (1pt) Give the corresponding SQL definition for the attribute (sometimes also called column) *type* in Artists and *nrseats* in ConcertHalls.

ii) (1.5pts) Give a primary key for Concerts and any secondary keys that may be needed.

iii) (1pt) Explain how to best modify the schema (by adding attributes, constraints or tables) in order to keep track of the number of tickets that have been sold for

a particular concert. (Seats are not numbered so it is enough to know the total number of sold tickets.)

iv) (1pt) Given your solution to the previous question, explain how to avoid selling more tickets that the available number of seats in the arena of the concert.

v) (1pt) Explain how to guarantee that a solo artist cannot charge more than 1M sek and that a band cannot charge more than 1.5M sek for a concert.

b) (3pts) Write an SQL query that list *all* the *past* concerts that were *not* profitable. A concert is profitable if the income from the sold tickets is higher than the price the artist charges for the concert. The output should contain the date of each concert, its artist and the lost in money. The SQL expression `CURRENT_DATE` gives today's date.

Solution:

```
CREATE TABLE Artists (
  name TEXT PRIMARY KEY,
  contact TEXT NOT NULL,
  price INT NOT NULL CHECK (price >= 0),
  type TEXT NOT NULL CHECK (type IN ('solo','band')),
  CONSTRAINT right_price CHECK ((type = 'solo' AND price <= 1000000) OR
                                (type = 'band' AND price <= 1500000)));

CREATE TABLE ConcertHalls (
  name TEXT,
  address TEXT,
  nrseats INT NOT NULL CHECK (nrseats > 0),
  PRIMARY KEY (name, address));

CREATE TABLE Concerts (
  date DATE,
  artist TEXT REFERENCES Artists,
  arena TEXT NOT NULL,
  address TEXT NOT NULL,
  tcktprice INT NOT NULL CHECK (tcktprice >= 0),
  nrsold INT NOT NULL CHECK (nrsold >= 0),
  PRIMARY KEY (date, artist),
  UNIQUE (date, arena, address),
  FOREIGN KEY (arena, address) REFERENCES ConcertHalls);
```

a) i-iii) See the tables.
iv) To guarantee that no more tickets than places in a particular concert hall are sold one could use a trigger that checks whether the number of sold tickets do not yet exceed the number of seats in the concert hall.
An alternative is adding the number of seats in Concerts with a reference to ConcertHalls

and then add a constraint that the number of sold tickets should be at most the number of seats.
v) The constraint `right_price` is the answer to the last question.

b)
```
SELECT date, artist, tcktprice*nrsold - price AS income
FROM Concerts JOIN Artists ON name = artist
WHERE date < CURRENT_DATE AND (tcktprice*nrsold - price) < 0;
```

Using `income` instead of `tcktprice*nrsold - price` in the `WHERE` clause does not work. Recall the `SELECT` part is perform at the end of the query.

2 SQL and RA (12 pts)

We continue with the same domain as in question 1 on SQL with the constraints, keys and additional information you defined previously:

Artists (<u>name</u>, contact, price, type)
ConcertHalls (<u>name</u>, <u>address</u>, nrseats)
Concerts (date, artist, arena, address, price)
artist → Artists.name
(arena, address) → ConcertHalls.(name, address)

a) (3+3pts) Write an SQL query and a relational algebra expression that outputs the percentage of sold tickets for each of the concerts in the table. The output should contain the date of the concert, the artist and the percentage of sold tickets for that particular concert. Order the output in descending order after the percentage.

b) (3+3pts) Write an SQL query and a relational algebra expression that for every concert hall outputs the number of *concerts* with "exclusive" artists that *had* concerts in the particular hall. A solo artist is considered exclusive if its price is higher than 0.5Msek, and a band is considered exclusive if its price is higher than 0.75Msek. The output should consist of the full identification of the concert halls and the number of concerts with exclusive artists that took place in each of the concert halls. Use `CURRENT_DATE` for today's date even in relation algebra.

Solution:

a)
```
SELECT date, artist, nrsold*100/nrseats AS percentage
FROM Concerts JOIN ConcertHalls
ON name = arena AND Concerts.address = ConcertHalls.address
ORDER BY percentage DESC;
```

An alternative way of presenting the percentage could be something like `ROUND (nrsold::numeric/nrseats, 2)`. Presumably both `nrsold` and `nrseats` are defined as `INT` and simply `nrsold/nrseats` will be the integer division and hence give just 0 or 1 (if all tickets were sold).

RA:
With $R = \mathsf{Concerts} \bowtie_{\mathsf{name=arena} \wedge \mathsf{Concerts.address=ConcertHalls.address}} \mathsf{ConcertHalls})$
the RA is $\tau_{-\mathsf{percentage}}(\pi_{\mathsf{date,artist,nrsold*100/nrseats} \to \mathsf{percentage}}\ R)$

b)
```
SELECT arena, address, COUNT(*) AS nr
FROM (SELECT * FROM Concerts WHERE date < CURRENT_DATE) AS C,
     (SELECT * FROM Artists
      WHERE (type = 'solo' AND price > 500000) OR
            (type = 'band' AND price > 750000)) AS A
WHERE C.artist = A.name
GROUP BY arena, address;
```

$$\gamma_{\mathsf{arena,address,COUNT}(*) \to \mathsf{nr}}(\sigma_{\mathsf{date} < \texttt{CURRENT\_DATE}}\mathsf{Concerts} \bowtie_{\mathsf{artist=name}} \sigma_{(\mathsf{type}='\mathsf{solo}' \wedge \mathsf{price} > 0.5\mathsf{M}) \vee (\mathsf{type}='\mathsf{band}' \wedge \mathsf{price} > 0.75\mathsf{M})}\mathsf{Artists})$$

3 Views and Triggers (13.5 pts)

We continue with the same domain as in question 1 on SQL with the constraints, keys and additional information you defined previously:

Artists (<u>name</u>, contact, price, type)
ConcertHalls (<u>name</u>, <u>address</u>, nrseats)
Concerts (date, artist, arena, address, price)
artist → Artists.name
(arena, address) → ConcertHalls.(name, address)

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

a) (4pts) Keep track of the artists and their type with the *highest* number of concerts *this* year when considering *all* concert halls.

The output should just consist of the name of the artists and their type. If there are more than one artist with the same *highest* number of concerts, all these artists and their type should be part of the answer.

The SQL expression `date_part('year',date)` gives the year part of `date`. Recall the expression `CURRENT_DATE`.

b) (4.5pts) Keep track of how many *concerts* with solo artists and with bands take place on each concert hall. All the concerts recorded in the table should be counted.

The output should contain four columns: two for the full identification of the concert hall and two columns for the number of solo concerts and the number of band concerts, respectively, in that particular concert hall.

c) (5pts) It takes time to transport everything from one concert hall to another.
Make sure an artist *does not* have a concert two days in a row *unless* it is in the same concert hall.

Given a particular `date`, the expressions `date+1` and `date-1` give the dates for the following and the previous days.

Solution:

a)
```
CREATE OR REPLACE VIEW ArtistMaxConcerts AS
WITH NrConcerts AS
  (SELECT artist, COUNT(*) AS nr
   FROM Concerts
   WHERE date_part('year',date) = date_part('year',CURRENT_DATE)
   GROUP BY artist)
SELECT artist, type
FROM NrConcerts JOIN Artists ON name = artist
WHERE nr = (SELECT MAX(nr) FROM NrConcerts);
```

b)
```
CREATE OR REPLACE VIEW ConcertsPerArena AS
WITH
ConcertsPerType AS
  (SELECT arena, address, type, COUNT(*) AS nr
   FROM Concerts JOIN Artists ON artist = Artists.name
   GROUP BY arena, address, type),
CPTS AS (SELECT arena, address, nr AS nrsolo
         FROM ConcertsPerType WHERE type = 'solo'),
CPTB AS (SELECT arena, address, nr AS nrband
         FROM ConcertsPerType WHERE type = 'band')
SELECT arena, address,
       COALESCE(nrsolo,0) AS nrsolo, COALESCE(nrband,0) AS nrband
FROM CPTS FULL OUTER JOIN CPTB USING (arena,address);
-- alternative: FROM CPTS NATURAL FULL OUTER JOIN CPTB;
```

c)
```
CREATE OR REPLACE FUNCTION book_concert() RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT * FROM Concerts
               WHERE artist = NEW.artist AND
                     (date = NEW.date + 1 OR date = NEW.date -1) AND
                     (arena != NEW.arena OR address != NEW.address))
    THEN RAISE EXCEPTION 'Not enough time between concerts in different halls';
    END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER BookConcert
  BEFORE INSERT ON Concerts
  FOR EACH ROW
  EXECUTE FUNCTION book_concert();
```

Defining the trigger `AFTER INSERT` would also have worked.

4 ER Modelling (10 pts)

a) (6pts) Your task is to make an ER-diagram for the employee database of your friendly neighborhood mega-corporation. Their HR department sent you this list of demands:

- Each employee has a unique corporate ID number, a name, and belongs to a department of the corporation. Most of the employees are unpaid interns, those who aren't have a salary in addition to their other attributes.
- Some of the paid employees are mentors. Mentors have a salary bonus (negotiated individually for each mentor). Every Mentor is responsible for mentoring another specific employee (the mentored employee could be paid or unpaid).
- Occasionally, paid employees get suspended without pay. For every such troublesome case, there is a date on which the suspension was issued and the number of days the employee was suspended. An employee can be suspended multiple times (but never more than once on the same day). The full history of suspensions should be recorded in the database.

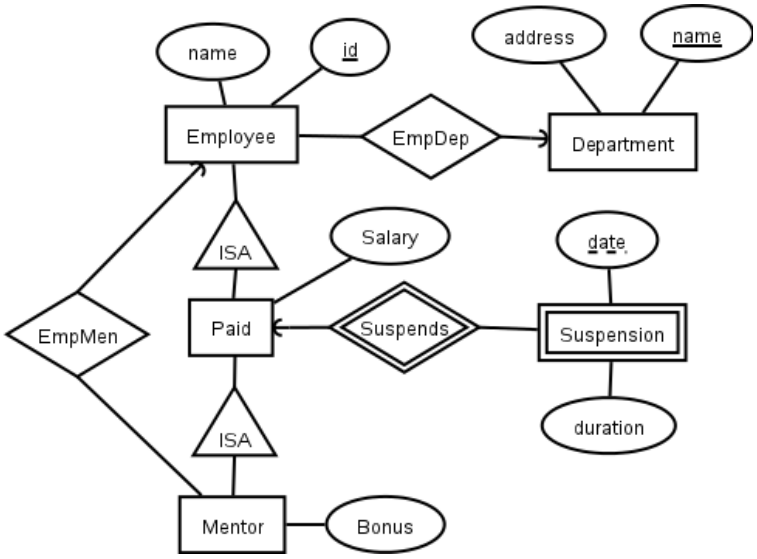
  Note: Only dates that have a suspension should appear in the database/included in the tables. Suspension periods might overlap.
- Every department of the corporation has a name and an address.

The HR department also sent along this hint: Consider how to model unpaid/paid/mentoring employees. What's the most special/most general case? Make sure there is no fourth kind of employee.

b) (4pts) Translate your diagram into a relational schema.

Solution:

a)



b)

Departments (<u>name</u>, address)
Employees (<u>id</u>, name, dep)
dep → Departments.name
Paids (<u>id</u>, salary)
id → Employees.id
Mentors (<u>id</u>, bonus, emp)
id → Paids.id
emp → Employees.id
Suspensions (<u>id</u>, <u>date</u>, duration)
id → Paids.id

5 Functional and Multivalued Dependencies (8.5pts)

Consider the relation `Exams(course, student, score, number, text, date)` containing information about exams at a university.

The attributes are to be understood as follows:

`course` is the course code the exam is for;
`student` is the id of a student;
`score` is the *total* score a student got on the exam;
`number` is the number of an exam question;

`text` is the text of an exam question;
`date` is the date of the exam.

A row like `(TDA357, Emilia, 40, 5, 'Consider the relation...', 2025-08-25)` means Emilia got 40 points on the August exam of the TDA357 course, and also that question five on the August exam had the specified question text 'Consider the relation...'.

Note 1: The relation contains all students and their scores, and all questions of all exams ever (many exams for each course).

Note 2: The score on individual questions is not kept in the database, just the total score of every student on each exam.

a) (2.5pts) Show using a counterexample why `student course` $\twoheadrightarrow$ `date` is not a valid MVD for Exams. Explain! (for full points the explanation should not leave gaps to fill in)

b) (2pts) Give a correct MVD for Exams that is not a functional dependency. (No motivation is needed.)

c) (3pts) State two functional dependencies for Exams, and normalise it to BCNF, marking primary keys. You do not need to show your steps, just the FDs and the final schema with the primary keys.

d) (1pt) We will not ask you to normalise the schema from c) to 4NF using the MVD from b). Why do you think that is?

Solution:

a) Here is an example with a student that has taken two exams for the same course and gotten different scores in those exams (? can be anything, it could also have different question texts):
(C,S,10,?,?,D1)
(C,S,20,?,?,D2)
For `course` $\twoheadrightarrow$ `date` to be a MVD the following tuples need to also be in the table:
(C,S,10,?,?,D2)
(C,S,20,?,?,D1)
but that would mean that the student has 2 different total scores in the exam of a course for a particular date and that will not be correct.

b) One of those:
course date $\twoheadrightarrow$ student score
course date $\twoheadrightarrow$ number text

c) course, date, student $\rightarrow$ score
course, date, number $\rightarrow$ text

Schema:
R1(course, date, student, score)
R2(course, date, number, text)
R3(course, date, student, number)

d) The schema is already in 4NF.

6 JSON (7.5 pts)

Consider this JSON schema for representing messages posts in an online message board. Each post has a header and a text content. Every post is either a top level post, or a reply to another post:

```
{
  "type": "array",
  "items": [
    {
      "title": "post",
      "type": "object",
      "properties": {
        "header": {"type": "string"},
        "txt": {"type": "string"},
        "replies": {"$ref":"#", "minItems":1},
      },
      "additionalProperties": false,
      "required": ["header","txt"]
    }
  ]
}
```

a) (3pts) Translate this relational data (five posts) into the given format. Your answer should be a JSON document that validates against the schema above. In arrays, posts should be ordered by increasing id values in the table.

Posts(<u>id</u>, header, txt, replyTo (or null))
replyTo → Posts.id

(id, header, txt, replyTo)
(0, h1, x1, null)
(1, h2, x2, 0)
(3, h3, x3, 1)
(4, h4, x4, 0)
(5, h5, x5, null)

b) (2pts) Write a JSON path for finding the text of the first reply to the top level post with the header "h1". In the test data above, the result would be "x2". Note that the path should work for any valid document, not just this particular one.

c) (2.5pts) Write a JSON path for finding the headers of all posts that are replies. In the test data above, it would give three strings: [h2, h3, h4]

Solution:

a)
```
[{"header":"h1", "txt":"x1", "replies":[
    {"header":"h2", "txt":"x2", "replies":[
      {"header":"h3", "txt":"x3"}]},
    {"header":"h4", "txt":"x4"}]},
  {"header":"h5", "txt":"x5"}
 ]
```

b)
```
$[*]?(header=="h1").replies[0].txt
```

c)
```
$[*].replies.**.header
```