Databases Exam

TDA357 (Chalmers), DIT622 and DIT621 (University of Gothenburg)

2024-08-29 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 of the exam date. Maximum points: 60

Grade limits: 27 for 3, 38 for 4, 49 for 5. Grade limits for DIT621: 27 for G, 45 for VG.

Question 1&2: SQL queries, Constraints and RA (21 p)

We will be working with a database for movies, and their actors and directors. A movie is identified by its year of production and its title. A movie has a director, a set of actors, the length (in minutes) and a genre. For the purpose of this exam, the possible genres are just crime, comedy and drama.

Actors and directors are both considered celebrities and are just distinguished by their role in a movie. For actors/directors we know their name (which serves to identify them), their year and place of birth, and the minimum salary (per film and in thousand dollars) they work for. Due to agreement with the unions, no celebrity can work in a movie for less than 100K U$. Observe that a celebrity can both direct and act in one and the same movie, in which case they will both get payment as an actor and payment as a director on top of that.

An incomplete definition of the necessary tables for this domain is as follows (next page):

```
CREATE TABLE Celebrities (
  name TEXT PRIMARY KEY,
  byear INT NOT NULL,
  bplace TEXT NOT NULL,
  minsalary INT NOT NULL ... );
CREATE TABLE Movies (
  year INT,
  title TEXT,
  director TEXT NOT NULL REFERENCES ...,
  length INT NOT NULL ...,
  genre TEXT NOT NULL ...,
  PRIMARY KEY (year, title) );

CREATE TABLE MovieActors (
  year INT,
  title TEXT,
  name TEXT REFERENCES ...,
  PRIMARY KEY (year, title, name),
  FOREIGN KEY (year, title) REFERENCES ... );
```

a) (3pts) Define reasonable constraints to fill in the ... parts in the SQL tables above (six in total).
Note: It is enough for you to just write which constraint to add to which column in a table, you do not need to write the complete table definition again.

b) (3+3pts) Write an SQL query and a RA expression that outputs the year of production and the title of all the movies where the director of the movie also acts in it. The output should be given in descending order.

c) (3+3pts) Write an SQL query and a RA expression that outputs the year of production and title of all the movies that have at least one actor who was a minor (below 18 years) when the movie was produced. Each movie should appear only once in the output even if there were several minors acting in the movie.

d) (3+3pts) Write an SQL query and a RA expression that outputs the average length per genre of all the movies with your favourite actor. The output should then contain both the genre and the average.

a) In Celebrities:

```
CHECK (minsalary >= 100)
```

In Movies:

```
REFERENCES Celebrities
CHECK (length > 0)
CHECK (genre IN ('crime','comedy','drama'))
```

In MovieActors:

```
REFERENCES Celebrities
REFERENCES Movies
```

b)

```
SELECT year, title
  FROM Movies as M
  WHERE director IN (SELECT name FROM MovieActors as MA
                              WHERE M.year = MA.year AND M.title = MA.title)

  ORDER BY (year, title) DESC;

-- Alternative
SELECT year, title
FROM Movies NATURAL JOIN MovieActors
WHERE director = name
ORDER BY (year, title) DESC;
```

$$\tau_{-(\mathsf{year,title})}(\pi_{\mathsf{year,title}}(\sigma_{\mathsf{director=name}}(\mathsf{Movies} \bowtie \mathsf{MovieActors})))$$

c)

```
SELECT DISTINCT year, title
FROM MovieActors JOIN Celebrities USING (name)
WHERE year - byear < 18;
```

$$\delta(\pi_{\mathsf{year,title}}(\sigma_{\mathsf{year-byear}<18}(\mathsf{MovieActors} \bowtie_{\mathsf{MovieActors.name=Celebrities.name}} \mathsf{Celebrities})))$$

d)

```
SELECT genre, AVG(length)
FROM Movies NATURAL JOIN MovieActors
WHERE name = 'XX'
GROUP BY genre;
```

$$\gamma_{\mathsf{genre,AVG(length)}}(\sigma_{\mathsf{name='XX'}}(\mathsf{Movies} \bowtie \mathsf{MovieActors}))$$

Question 3: Views and Triggers (9p)

NOTE: This is question 3, not 2. Mark it as such in your solutions.

We continue with the same domain as in the previous section. See below of an incomplete relational schema of the domain:

```
Celebrity (name, byear, bplace, minsalary)
Movie (year, title, director, length, genre)
MovieActor (year, title, name)
```

a) (4pts) For tax reasons, one needs to keep track of how much celebrities earn. Define a view that for each celebrity born after 1930, outputs the total amount the celebrity has earned a certain year (say year 2023). If the celebrity has not worked in any movie on that certain year, then he/she should not be part of the output.

As an actor, a celebrity earns their minimum salary per movie. As a director, a celebrity earns 1.5 times their minimum salary per movie.

b) (5pts) To be allowed to direct a crime movie, a celebrity must have previously acted in at least one crime movie before and have already directed at least 3 movies of any genre (meaning, there are corresponding entries in the database with the information in question).

Make sure that every crime movie inserted in the database complies with these regulations.

Note: You do not need to check that the movies already in the database comply with these requirements, only those that are newly inserted should be checked.

a)

```
CREATE OR REPLACE VIEW TaxInfo AS
  WITH Salaries AS
  (SELECT name, minsalary AS salary
   FROM Celebrities NATURAL JOIN MovieActors
   WHERE byear > 1930 and year = 2023
 UNION ALL
   SELECT name, minsalary*1.5 AS salary
   FROM Celebrities, Movies
   WHERE name = director AND byear > 1930 and year = 2023)

SELECT name, SUM(salary) AS TotalSalary
  FROM Salaries
  GROUP BY name;
```

b)

```
CREATE OR REPLACE FUNCTION crime_movie() RETURNS TRIGGER AS $$
BEGIN
  IF NOT (EXISTS (SELECT name FROM MovieActors NATURAL JOIN Movies
                            WHERE genre = 'crime' AND name = NEW.director))

  THEN
     RAISE EXCEPTION 'Celebrity has not acted in crime movie before';


  ELSE IF (SELECT COUNT(*) FROM Movies
       WHERE director = NEW.director) < 3
  THEN
     RAISE EXCEPTION 'Celebrity has not acted in enough movies';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER NewCrimeMovie
BEFORE INSERT ON Movies
FOR EACH ROW
EXECUTE FUNCTION crime_movie();
```

Question 4: ER-modelling (11p)

a) (7 p) You are designing a database for an online auction house.

Users are identified by their usernames. Users can both sell items, and place bids on items that other people sell (a user can make multiple bids on the same item). Users can also subscribe to items, receiving notifications when a new bid is added to it.

Every item offered up for auction is recorded and has a single user selling it. Items have names, but multiple items can have the same name. Each item has a time of expiration, when the bidding ends and the item is sold. Items remain in the database even after the auction has ended.

Users can bid on items. All bids are permanently recorded. There is a time stamp for each bid, and an amount of money offered. Two bids on the same item cannot have the exact same amount of money (but bids on different items can have).

Some items are promoted, meaning that they are advertised on the front page of the auction site. For each such promoted item, there is a time when the promotion starts, then it runs throughout the sale of the item.

b) (4 p) Draw an ER-diagram that translates into exactly this relational schema:

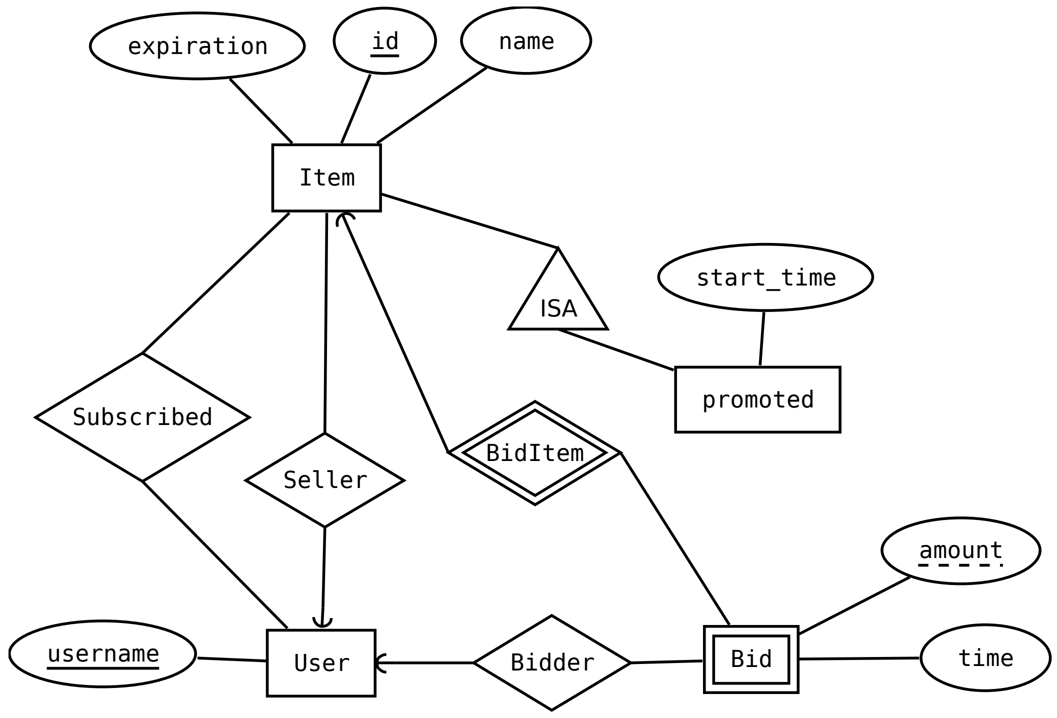
A(<u>a1</u>, a2)

B(<u>b1</u>, <u>b2</u>, ba (or null))
  ba → A.a1

C(<u>c1</u>, c2, ca)
  ca → A.a1

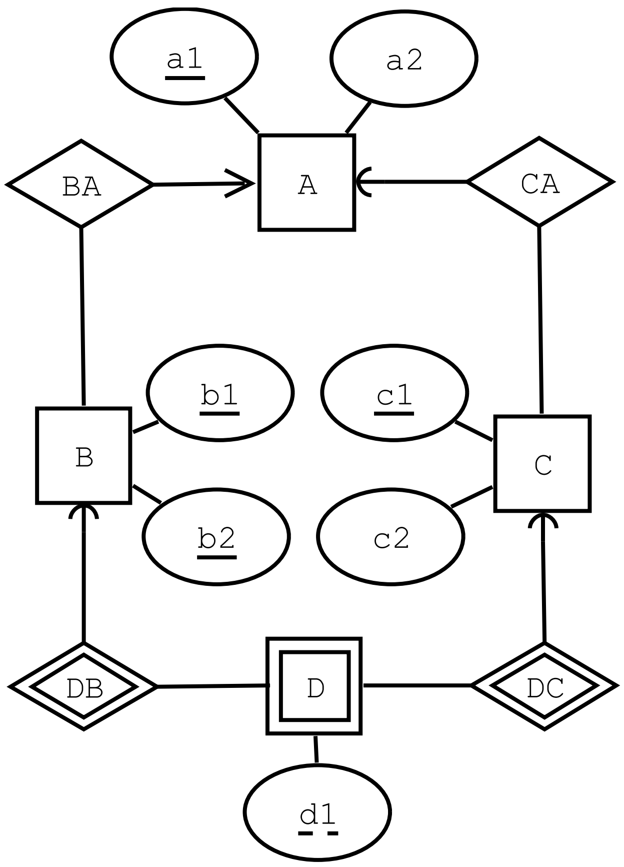
D(<u>db1</u>, <u>db2</u>, <u>dc</u>, <u>d1</u>)
  (db1, db2) → B.(b1, b2)
  dc → C.c1

a)



b) Names of relationships can be anything.

d1 needs to be undelined but it's fine if it's not dashed.



Question 5: Dependencies and normal forms (10p)

a) (4 p) Identify three functional dependencies that hold on this data. Your solution should be a minimal cover, meaning that there are no redundant FDs (that are implied by other FDs) in your answer, and the left hand side of every FD should be minimal.

Hint: For each attribute, try determining "what is sufficient to determine this?", starting with all other attributes and then narrowing it down.

x	y	z	w
1	2	3	4
2	3	4	4
3	3	4	4
3	4	4	6
4	4	5	6

b) (2 p) Consider this set of functional dependencies (seven in total):

c → d
e b → a
e b → d
a b → c
d → e
e → d
e → c

List all the dependencies from the set that are redundant (i.e. that are implied by the other FDs and can be removed from the original set without altering the meaning).

c) A domain R(course, student, grade, teacher) is used to keep track of student grades in various courses, and who is the (single) teacher of each course. For instance two tuples like {(TDA357, Emilia, 4, Jonas), (TDA357, Emil, 4, Jonas)} would mean that Emilia and Emil both have grade 4 in the course TDA357, for which Jonas is the teacher.

Note: There can only be one teacher on a course!

(2p) Is course ↠ student a reasonable MVD here? Explain why (one sentence) or provide a counterexample (a table where the MVD does not hold).

(2p) Is course ↠ student grade a reasonable MVD here? Explain why (one sentence) or provide a counterexample (a table where the MVD does not hold).

a)

x -> z

z w -> y

y -> w

b)

e b -> d

e -> d

c)

I: No. Consider {(TDA357, Emilia, 4, Jonas), (TDA357, Emil, 3, Jonas)}. Here student is not independent from grade (and it clearly shouldn't be).

II: Yes. Considering the dual course $\twoheadrightarrow$ teacher makes this more obvious, the teacher of a course is independent from which students has what grades in the course (This is in fact a functional dependency).

(Answering something like "no because course -> teacher is a functional dependency" is acceptable even though it's not technically correct - since the MVD would not affect 4NF normalization)

Question 6: Semi-structured data and other topics (9p)

Consider this JSON document for storing general tree structures and numbers:

```
{"branch": [
    {"branch": [
        {"branch": [1,2,3]},
        {"branch": [
            {"branch": []}
        ]},
        {"branch": [20,30,40]}
    ]}
]}
```

a) (4 p) Sketch a JSON Schema for documents like this (meaning any tree with a similar structure). The trees can have arbitrarily many levels. Note that the arrays that the "branch" keys map to either have only trees or only numbers, never a mix within a single list. The objects can have additional keys (other than "branch"), that do not need to be specified.

b) (2.5 p) Write a JSON Path for finding all arrays on the second level of trees like the one above, where the first element is a number below 10. In the example above it would give a single value: the array [1,2,3].

c) (2.5 p) Write a JSON Path for finding all branch-objects in a tree that contain an empty array. In the example above it would give a single result (`{"branch": []}`).

a)

```
{
  "type":"object",
  "properties":{
    "branch": {"anyOf":[
        {"type":"array",
         "items": {"$ref":"#"}},
        {"type":"array",
         "items": {"type":"number"}}
      ]}
  }
}
```

You could factor out the “type”:”array” to the same level as the anyOf (I think, haven’t tried it)

b)

```
$.branch[*].branch[*].branch?(@[0] < 10)
```

Having one fewer branch is also[*] is also okay since the question said “second level” which is a bit ambiguous.

c)

```
$**?(@.branch == [])
```