Databases Exam

TDA357 (Chalmers), DIT621 (University of Gothenburg)

2023-01-11 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.

Habitats(species, region, lowEstimate, highEstimate)
Endangered(species, globalLowEstimate)
Extinct(species)

The database contains information on animal species, their habitats and estimated populations.

The species columns are all names of known animal species. A region is a geographical area (like "North America" but could also be smaller areas).

A habitat indicates the presence of a species in a region. A species can have multiple habitats in different regions. Each habitat has a low and high estimate of the number of individuals in the area. The low estimate must be non-negative and cannot exceed the high estimate. The high estimate must be greater than zero.

The globalLowEstimate column contains the sum of the lower estimate of all habitats of a species. An animal is considered endangered (and should be included in Endangered) if this number is below 1000. A species is extinct if it has no remaining habitats.

Here is an example of valid content of (the public interface of) the database:

Habitats(species, region, lowEstimate, highEstimate)
(Mountain Gorilla, DRC, 400, 600)
(Mountain Gorilla, Uganda, 400, 500)
(Moose, North America, 900000, 1100000)

Endangered(species, globalLowEstimate)
(Mountain Gorilla, 800)

Extinct(species)
(Dodo)

b) Insertions: Show all the inserts required to get the data above.

Solution 1:

```
-- There needs to be some way to define known species that have no
habitats. The simple and elegant way to do that is to have a master table
for all known species, and a reference in habitats (the reference is not
strictly necessary) - and then the extinct species are simply those who are
in Species but not in Habitats

-- Adding some special encoding like setting lowEstimate to NULL in
Habitats for extinct animals does not work since it does not prevent the
animal from having other habitats with positive estimates (making them both
extinct and having habitats). Extinction needs to be defined as the absence
of habitats.

-- solutions based on triggers will not give full points, unless they cover
ALL the possible cases (like “what if I modify the name of a species column
of a habitat”) they can give points

-- a)
CREATE TABLE Species(name TEXT PRIMARY KEY);

CREATE TABLE Habitats(
  species TEXT REFERENCES Species,
  area TEXT,
  lowEstimate INTEGER NOT NULL,
  highEstimate INTEGER NOT NULL,
  CHECK (lowEstimate >= 0
     AND lowEstimate <= highEstimate
     AND highEstimate > 0),
  PRIMARY KEY(species, area)
);

CREATE VIEW Endangered AS
SELECT species, SUM(lowEstimate)
FROM Habitats
GROUP BY species
HAVING SUM(lowEstimate) <= 1000;

CREATE VIEW Extinct AS
(SELECT name AS species FROM Species)
EXCEPT
(SELECT species FROM Habitats);



-- b)
INSERT INTO Species VALUES('Dodo');
INSERT INTO Species VALUES('Mountain Gorilla');
INSERT INTO Species VALUES('Moose');

INSERT INTO Habitats VALUES('Mountain Gorilla', 'DRC', 400, 600);
INSERT INTO Habitats VALUES('Mountain Gorilla', 'Uganda', 400, 500);
INSERT INTO Habitats VALUES('Moose', 'North America', 900000, 1100000);
```

Question 2: SQL queries (10 p)

Consider this little database of scientific papers and citations between them:

Papers(<u>title</u>, <u>author</u>, year, journal)

Citations(<u>title</u>, <u>author</u>, <u>citedTitle</u>, <u>citedAuthor</u>)
(title, author) -> Papers.(title, author)
(citedTitle, citedAuthor) -> Papers.(title, author)

Each paper has a title and (for simplicity) a single author. They also have a year of publication and the name of the journal in which they were published.

For Citations, a row (p, x, q, y) means that the paper p (written by x) contains a citation of paper q (written by y).

a) Write an SQL query for finding all pairs of authors who have cited each other. That is: all pairs of distinct authors (a1, a2) such that a1 has written at least one paper citing a paper written by a2, and vice versa. No pair should appear more than once in the result regardless of internal order, so if (a1, a2) is included then (a2, a1) must be excluded.

b) One way to estimate of the impact of a journal is to track how many papers cite the papers it publishes, and how this number changes over time. Write an SQL query that: For each journal, counts the yearly number of papers that cite at least one paper published in the journal. The result should have three columns, for journal, year (when the citing papers were written) and number of citing papers.

Example: A row like (Nature, 1998, 321) in the result would mean that the database contains 321 <u>different</u> papers written in 1998 that each cite at least one article published in Nature.

Note: We are counting number of citing papers, not the number of citations - if a single paper cites multiple papers from the same journal, it should only be counted once!

Note: Years in which the journal was not cited at all do not need to be included.

Solution 2:

```
-- a)

SELECT DISTINCT author, citedAuthor
FROM Citations
WHERE
  author < citedAuthor AND (author, citedAuthor)
    IN (SELECT citedAuthor, author FROM Citations);

-- alternative solution:
(SELECT author, citedAuthor
FROM Citations WHERE author < citedAuthor)
INTERSECT
(SELECT citedAuthor, author FROM Citations);

-- You can also make a solution based on a self-join on Citations:
SELECT DISTINCT C1.author, C2.author
FROM Citations AS C1 JOIN Citations AS C2
  ON C1.author=C2.citedAuthor AND C2.author=C1.citedAuthor
WHERE C1.author < C2.author;


-- b)
-- CitationsPlus has, for each citing paper:
-- * The author and title of the citing paper
-- * the publication year of the citing paper
-- * the journal being cited
-- Having the author and title of (only) the citing paper makes sure
DISTINCT does the right thing.
-- (e.g. P3 is only counted as citing J1 once)
WITH CitationsPlus AS
     (SELECT DISTINCT P1.author, P1.title, P1.year, P2.journal
     FROM Citations AS C
          NATURAL JOIN Papers P1
          JOIN Papers AS P2 ON citedTitle=P2.title AND
citedAuthor=P2.author
     )
SELECT journal, year, COUNT(*) AS citations
FROM CitationsPlus
GROUP BY journal, year;
```

Question 3: Relational Algebra (10 p)

For the same database as in Question 2 (repeated here as a reminder):

Papers(<u>title</u>, <u>author</u>, year, journal)

Citations(<u>title</u>, <u>author</u>, <u>citedTitle</u>, <u>citedAuthor</u>)
(title, author) -> Papers.(title, author)
(citedTitle, citedAuthor) -> Papers.(title, author)

a) Write a relational algebra expression that finds how many times each author has been cited, excluding self-citations (a self-citation is when an author cites a paper they have written themselves). You do not need to include authors that have never been cited by other authors. The result should contain authors and number of citations.

b) Write a relational algebra expression that finds all authors that have published at least ten papers but never been cited by another author (self-citations do not count).

Solution 3:

$\text{nonSelf} = \sigma_{\text{citedAuthor} \neq \text{author}}(\text{Citations})$

$\text{manyPapers} = \pi_{\text{author}}(\sigma_{\text{papers} >= 10}(\gamma_{\text{author, count(*)->papers}}(\text{Papers})$

a)

$\gamma_{\text{citedAuthor, count(*)->citations}}(\text{nonSelf})$

b)

$\text{manyPapers} - \pi_{\text{citedAuthor}}(\text{nonSelf})$

Question 4: ER-modelling (10p)

a) (6 p) A web service for creating and answering quizzes has a large dataset of shared questions (so the same question can appear in multiple different quizzes).

- Each quiz has a name and a collection of questions. Two quizzes cannot have the same name. The questions appear in a specified order (so each question appearing in a quiz has a position).
- Questions have a question text and an answer text (for the correct answer). Neither of these are unique between questions.
- Each question can have a set of tags (typically for subjects like "2022" or "Lord of The Rings"). The tags are selected from a predefined set of tags.
- Questions can be open (the user types in an answer) or have alternatives. Alternative questions are distinguished by having a (non-empty) set of incorrect alternatives connected to them. There is no set of predefined alternatives (so when an alternative is not applied to any question, it should no longer exist).
- The database has users, each user has an email address. Each question is written by a user, and each quiz is owned by a user.

Hint: There may be entities that have no natural keys to them, and you will probably need to introduce some synthetic keys (like id-numbers).

b) (4 p) Draw a diagram that translates into this relational schema (using the standard rules for translating ER-diagrams, choosing names during the translation as needed):

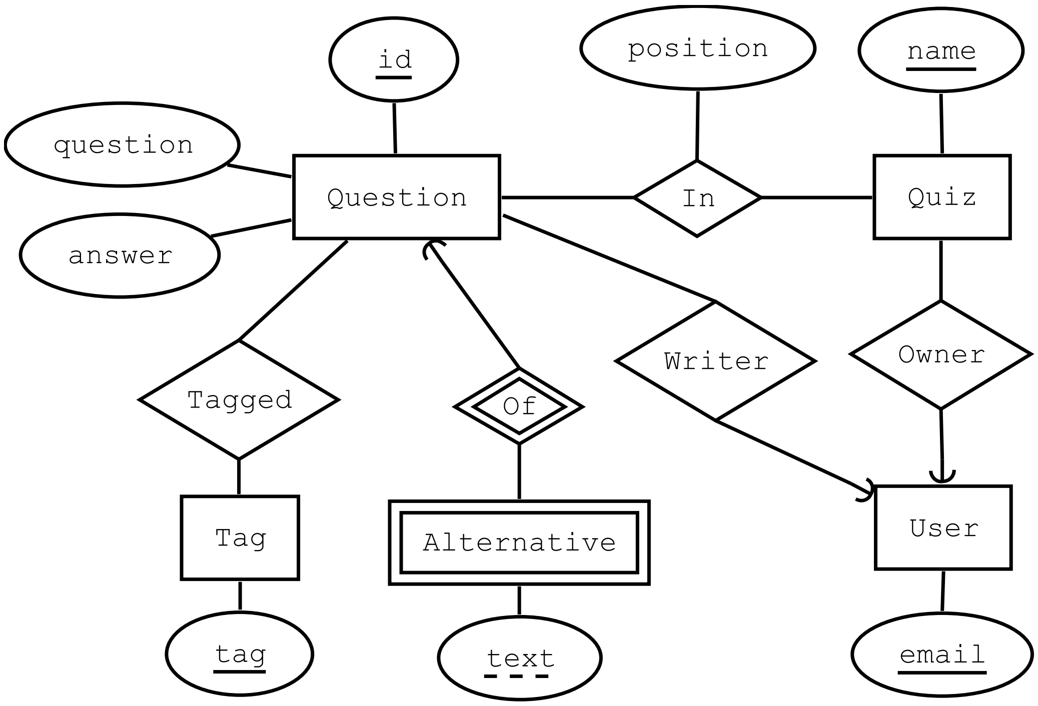
R($\underline{a}$, b)

S($\underline{c}$, $\underline{d}$, e)
  e → R.a

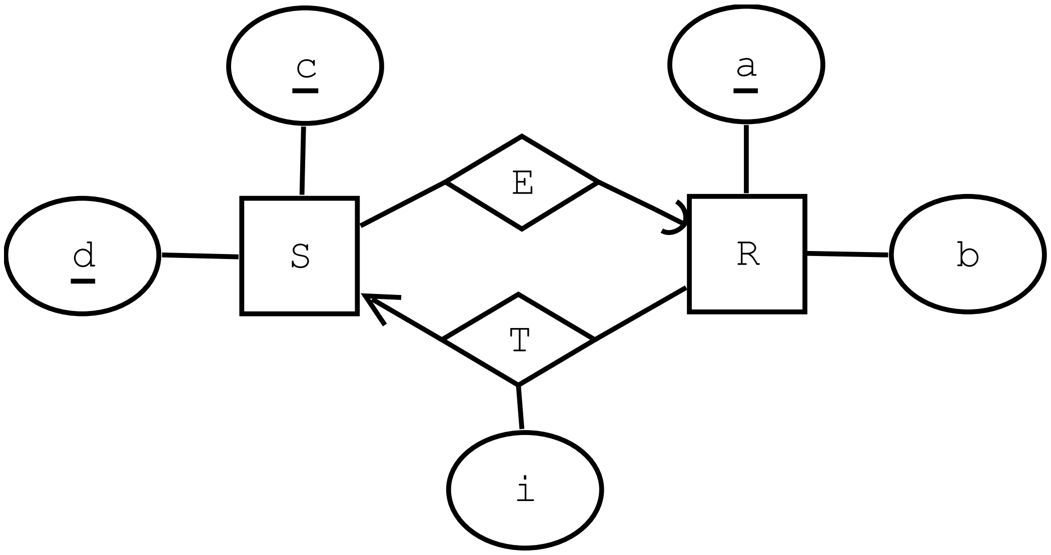
T($\underline{f}$, g, h, i)
  f → R.a
  (g,h) → S.(c,d)

Solution 4:

a)



b) For T, you could use an ISA and a many-to-exactly-one for an equivalent diagram (or a weak entity that is equivalent to an ISA). E can be named anything.



Question 5: Dependencies and normal forms (10p)

You are designing a database for an online computer games store, where users can own games and have store credits etc.

R(credit, developer, game, location, revenue, user)

- credit: Store credit of a user.
- developer: The name of a game developer.
- game: The name of a computer game. Different games always have different names.
- location: The location (country) in which a developer pays taxes.
- revenue: The estimated yearly revenue of a developer.
- user: A player who owns a set of computer games.

Example contents (should clarify any ambiguities):

credit	developer	game	location	revenue	user
100	Dev1	G1	L1	1000	U1
200	Dev1	G1	L1	1000	U2
100	Dev1	G2	L1	1000	U1
100	Dev2	G2	L2	2000	U1
100	Dev3	G3	L1	2000	U3

a) (3 p) Normalize R to BCNF, showing all decomposition step. For each decomposition, indicate clearly which violation you use, and which relation is decomposed.

b) (2 p) Further normalize your result from a) to 4NF. Again, indicate clearly which MVD you are using in each step.

c) (2 p) Draw tables representing the example data above in your normalized schema (in 4NF).

d) (3 p) This question is unrelated to previous questions. Consider this set of functional dependencies (seven in total!):

F={
 $A \rightarrow B$,
 $B\ C \rightarrow D$,
 $A\ C \rightarrow D\ B$,
 $C \rightarrow E$,
 $E \rightarrow C\ D$
}

Find a minimal basis F^- (also known as a minimal cover) for this set. As a reminder, this means a subset of F where no FD in F^- can be deduced from other FDs in F^-.

Solution 5:

a)

Split R on user -> credit
R1(user, credit)
R2(developer, game, location, revenue, user)

Split R2 on developer -> revenue location
R21(developer, revenue, location)
R22(developer, game, user)

R1, R21, R22 are all BCNF.

b)

Split R22 on game ->> user (or game ->> developer)

R221(game, developer)
R222(game, user)

c)

credit	**user**
100	U1
200	U2
100	U3

developer	**location**	**revenue**
Dev1	L1	1000
Dev2	L2	2000
Dev3	L1	2000

developer	**game**
Dev1	G1
Dev1	G2
Dev2	G2
Dev3	G3

game	**user**
G1	U1
G1	U2
G2	U1
G3	U3

d) Here is one solution (possibly the only correct one). E -> C, E ->D could be merged.

F- ={
 A -> B,
 C -> E,
 E -> C,
 E -> D
}

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

A JSON Schema (see next page) describes reference lists of academic literature. It defines two types of references: Books and websites. A third type of reference (thesis) is planned but not added yet.

a) (3p) Write an example JSON-document that validates against the schema and contains a book and a website. Names and such do not need to be realistic.

b) (4p) Write a JSON Path for finding the url of all websites accessed before February 2023.

c) (3 p) Write a JSON schema to replace `false` for the thesis part of the schema, enabling citations of thesis with (at least) author, title, and year of publication.

For full points, you should allow additional attributes (in addition to author, title, and year) as much as practically possible (remaining compatible with the rest of the schema).

You are not allowed to modify other parts of the schema.

```
{"type":"array",
 "description":"A reference list",
 "items":{"oneOf":[{"$ref":"#/definitions/book"},
                   {"$ref":"#/definitions/site"},
                   {"$ref":"#/definitions/thesis"}]},
 "definitions":{
   "book":{
     "type":"object",
     "properties":{
       "author":{"type":"string"},
       "title":{"type":"string"},
       "publisher":{"type":"string"},
       "year":{"type":"integer"}
     },
     "required":["author","title","publisher","year"]
   },
   "site":{
     "type":"object",
     "properties":{
       "author":{"type":"string"},
       "title":{"type":"string"},
       "url":{"type":"string"},
       "accessed":{
         "type":"object",
         "properties":{
           "year":{"type":"integer"},
           "month":{"enum":["Jan","Feb","Mar","Apr","May","Jun",
                            "Jul","Aug","Sep","Oct","Nov","Dec"]},
           "day":{"type":"integer"}
         },
         "required":["year","month","day"]
       }
     },
     "required":["author","title","url","accessed"]
   },
   "thesis":false
 }
}
```

Solution 6:

a)

```
[{"author":"x","title":"x","location":"x","isbn":"x","year":1999}
,{"author":"x","title":"x","website":"x","url":"x",
  "accessed":{"year":2022, "month":"Jan","day":11}}
]
```

b)

```
$[*]?(@.accessed.year < 2023 ||
      @.accessed.year==2023 && @.accessed.month=='Jan').url
```

c) Here is one way to make a schema that prevents overlap with the other two types, but still allows additional attributes:

```
{
     "type":"object",
     "properties":{
       "author":{"type":"string"},
       "title":{"type":"string"},
       "publisher":false,
       "accessed":false,
       "year":{"type":"integer"}
     },
     "required":["author","title","year"]
}
```