Databases Exam

(With solutions)

TDA357 (Chalmers), DIT621 (University of Gothenburg)

2020-01-15 14:00-18:00

Department of Computer Science and Engineering

Examiner: Jonas Duregård tel. 031-772 1028.

Results: Will be published within three weeks from exam date

Maximum points: 60

Grade limits Chalmers: 24 for 3, 36 for 4, 48 for 5.
Grade limits GU: 24 for G, 42 for VG.

Allowed material: One double sided A4 sheet with hand-written notes. If you bring a sheet, it must be handed in with your answers to the exam questions, add "+1" in the box for number of pages on the exam cover. Also, a standard reference is handed out in a separate document.

One English language dictionary is also allowed. You can answer in English or Swedish. Begin the answer to each question (numbers 1 to 6) on a new page. The a,b,c,... parts with the same number can be on the same page.

Write the question number on every page. Write clearly, unreadable answers give no points! Fewer points are sometimes given for solutions that are clearly unnecessarily complicated. Indicate clearly if you make any assumptions that are not given in the question. In particular: in SQL questions, use standard SQL or PostgreSQL. If you use any other variant (such as Oracle or MySQL), say this; but full points are not guaranteed since this may change the nature of the question.

Question 1: ER-design (10 points, 4+4+2)

a) Make an ER-diagram for a waiting list system for lab sessions, somewhat similar to the one used in this course.

The system can handle multiple waiting lists. Each waiting list has an owner (the course responsible for the course, identified by their email-address). When creating a waiting list, the owner provides the course name, a list of rooms for the lab sessions and a list of help topics.

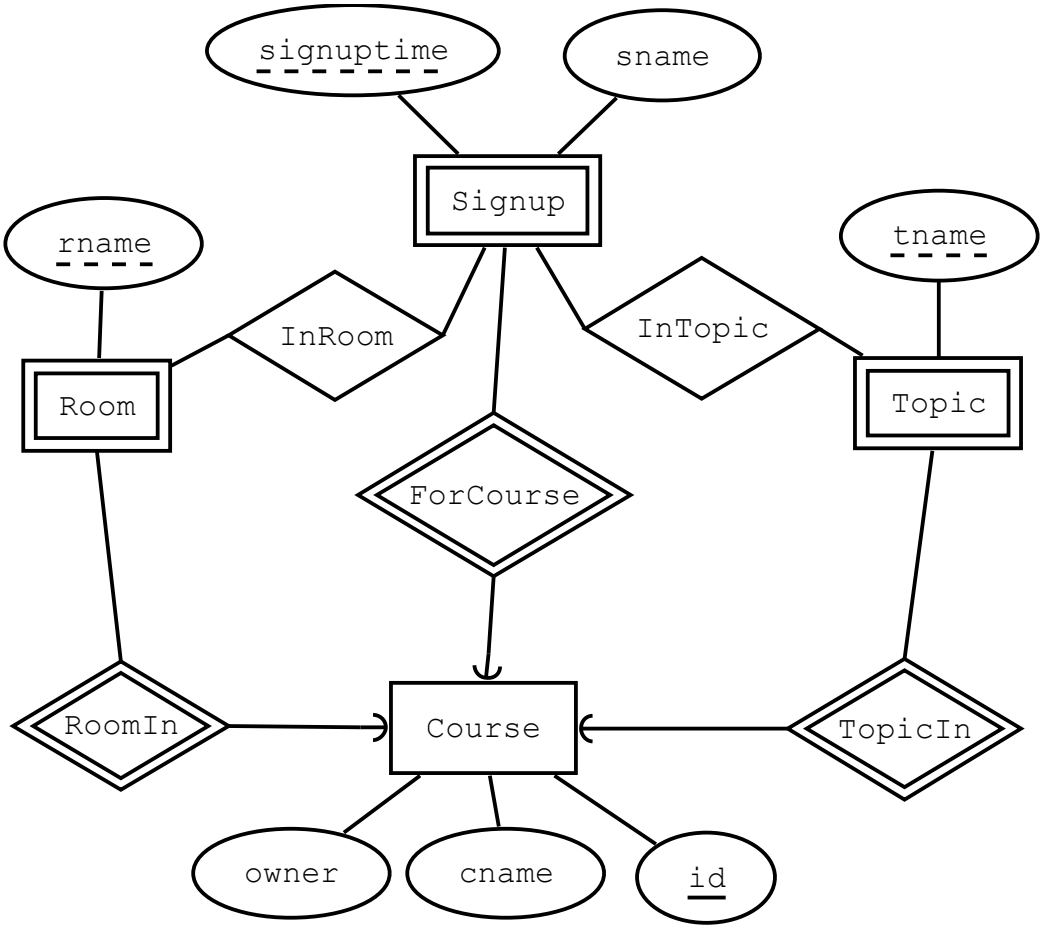
Students can sign up to any waiting list by providing their name and choosing one of the help topics and one of the rooms for the waiting list. Note: You may not be able to enforce that the topic and room are for the same course (see below), but it must be possible to list the help topics and rooms of any course using your design. The time the students sign up at should also be recorded.

Multiple waiting lists can have the same name. Rooms in different courses can have the same name, but not within a course. The same goes for help topic names.

b) Translate your diagram from a) into a relational schema using the standard translation algorithm. In this part you should not do any clever tricks, just translate your ER-diagram into an equivalent schema.

c) If your ER-design does not guarantee that students can only sign up for topics and room of the same course, make minimal changes to your schema to enforce this. You only need to provide the parts of the schema that you change compared to (b).

1 a) This turned out a bit more complicated than I had intended, grading will take this into account



2 b) Correct translations of your diagram give points. For the diagram above:

```
Courses(owner, cname, id)
Topics(tname, course)
  course -> Courses.id
Rooms(rname, course)
  course -> Courses.id
Signups(tname,tcourse,rname,rcourse, course, signuptime,sname)
  (tname, tcourse) ->  Topics(tname, course)
  (rname, rcourse) ->  Rooms(tname, course)
  course -> Courses.id
```

1 c) Either add tcourse=rcourse (=course) to signups, or just merge them to:

```
Signups(tname, rname, course, signuptime, sname)
  (tname, course) ->  Topic(tname, course)
  (rname, course) ->  Room(tname, course)
```

Question 2: Functional Dependencies, Normal Forms (10 p, 6+4)

Consider a relation R(A, B, C, D, E, F) with the following Functional Dependencies

$A \rightarrow B$
$A\ C \rightarrow D$
$A\ C \rightarrow E$
$A\ C \rightarrow F$
$D \rightarrow A$
$D \rightarrow C$
$A\ F \rightarrow E$
$E \rightarrow A$

a) Identify three different BCNF violations and compute the closure of the left hand side. Your solution should be three closures on the form:
$\{...\}^+ = \{...\}$

The right hand side of the equality should be different in all three, and all should be BCNF violations of R.

b) Study this table of courses, course classifications and exam registrations:

course	**examDate**	**student**	**classification**
TDA357	2020-01-15	Emil	Computer Science
TDA357	2020-01-15	Emilia	Computer Science
TDA357	2020-03-20	Emil	Computer Science
TDA357	2020-01-15	Emil	Fun!
TDA357	2020-01-15	Emilia	Fun!
TDA357	2020-03-20	Emil	Fun!
XYZ123	2020-01-15	Emil	Computer Science

Identify a non-trivial MVD that holds on this data and violates 4NF, and provide a schema in 4NF based on that MVD (you do not need to provide the data, just the schema and the MVD you use).

2 a) I believe this is the only correct solution:
$\{A\}^+ = \{A,B\}$
$\{A, F\}^+ = \{A, B, E, F\}$
$\{E\}^+ = \{A,B,E\}$

2 b)
course ->> classification (or equivalently: course ->> examDate, student)

R1(course, examDate, student)
R2(course, classification)

Question 3: SQL Queries (10 p, 5+5)

Below is the schema for a database of posts and comments on some kind of web-application. Every post is written by an author. Comments can be written to posts, each comment has a number (unique within posts) and may be either a top level comment (when replyTo is null) or a reply to an earlier comment of the same post (replyTo is the number of the comment it replies to).

```
Posts(idnr, author, text)

Comments(number, inPost, commenter, text, replyTo (or null))
  inPost -> Posts.idnr
```

Example contents:

idnr	author	text
0	Jonas	...
1	Matti	...
2	Jonas	...
3	Test	...

number	inPost	commenter	text	replyTo
0	1	Jonas	First!	null
1	1	Sanoj	*sigh*	0
2	1	Jonas	:(	1
3	1	Matti	I agree	1
0	0	Matti	Nice	null
0	2	Jonas	Thoughts?	null

a) Find the number and post-id of all comments that are replies to replies. In this case (2,1) and (3,1).

b) For each commenter, find the post author(s) they have written most comments (including replies) to. The result for the example data should be:

commenter	author
Jonas	Matti
Sanoj	Matti
Matti	Jonas
Matti	Matti

Including test data (not part of solution):

```
CREATE TABLE Posts
  ( idnr SERIAL PRIMARY KEY
  , author TEXT NOT NULL
  , text TEXT NOT NULL
  );

CREATE TABLE Comments
  ( number INT NOT NULL
  , inPost INT NOT NULL REFERENCES Posts
  , commenter TEXT NOT NULL
  , text TEXT NOT NULL
  , replyTo INT
  , PRIMARY KEY (number, inPost)
  );

INSERT INTO Posts VALUES (0, 'Jonas', '...');
INSERT INTO Posts VALUES (1, 'Matti', '...');
INSERT INTO Posts VALUES (2, 'Jonas', '...');
INSERT INTO Posts VALUES (3, 'Test', '...');

INSERT INTO Comments VALUES (0, 1, 'Jonas', 'First!', null);
INSERT INTO Comments VALUES (1, 1, 'Sanoj', '*sigh*', 0);
INSERT INTO Comments VALUES (2, 1, 'Jonas', ':(', 1);
INSERT INTO Comments VALUES (3, 1, 'Matti', 'I agree', 1);
INSERT INTO Comments VALUES (0, 0, 'Matti', 'Nice', null);
INSERT INTO Comments VALUES (0, 2, 'Jonas', 'Thoughts?',
null);

-- 3 a
SELECT A.number, A.inpost
  FROM Comments A, Comments B
  WHERE A.replyTo = B.number
        AND A.inPost = B.inPost
        AND B.replyTo IS NOT null;

-- 3 b
WITH
  Counts AS
  (SELECT commenter, author, count(*) AS total
    FROM Posts JOIN Comments ON (idnr = inPost)
    GROUP BY commenter, author),
  Maxes AS
  (SELECT commenter, max(total) FROM Counts
   GROUP BY commenter)
SELECT commenter, author
  FROM Counts NATURAL JOIN Maxes
  WHERE total = max;
```

Question 4: Relational Algebra (10, 3+3+4)

a) On the same schema and data as the previous question, what is the result of this relational algebra expression? (Your answer should be a table with three columns)

$$\gamma_{\textbf{commenter, inPost, MIN(number) -> first}}(\textbf{Comments})$$

b) On the same schema as the previous question, write a relational algebra expression that gives the text of all posts and all comments written by Jonas, in a single column.

You do not need to consider duplicate texts, and may either include or exclude duplicates. With duplicates included, the result would have five rows for the example data.

c) On the same schema as the previous question, write a relational algebra expression that for each reply, finds its number, post, and text along with the text of the comment it replies to. Expected result for given data

number	inPost	text	parentText
1	1	*sigh*	First
2	1	:(	*sigh*
3	1	I agree	*sigh*

4 a)

```
commenter   | inpost | first
------------+--------+-------
 Jonas      |      2 |     0
 Sanoj      |      1 |     1
 Matti      |      1 |     3
 Jonas      |      1 |     0
 Matti      |      0 |     0
```

4 b) and c), including SQL code just if you want to test

```sql
-- 4 b
-- pi[text] (sigma[author='Jonas'] Posts)
-- U
-- pi[text] (sigma[author='Jonas'] Comments)
SELECT text FROM Posts WHERE author = 'Jonas'
UNION
SELECT text FROM Comments WHERE commenter = 'Jonas';


-- 4 c
--pi[A.number, A.inPost, A.text, B.text]
--  sigma[A.replyTo = B.number & A.inPost = B.inPost]
--    (rho[A] Comments) X (rho[B] Comments)
SELECT A.number, A.inPost, A.text, B.text
  FROM Comments A, Comments B
  WHERE A.replyTo = B.number
        AND A.inPost = B.inPost;
```

Question 5: Views, constraints and triggers (10 p)

Database integrity can be improved by several techniques:

- Views: virtual tables that show useful information that would create redundancy if stored in the actual tables
- SQL Constraints: conditions on attribute values and tuples
- Triggers (and assertions): automated checks and actions performed on entire tables

As a general rule, these methods should be applied in the above order: if a view or constraint can adequately do the job, do not use a trigger.

The task in this question is to implement a small database. You may use any SQL features we have covered in the course. While the description below gives requirements for what should be in the database, you are allowed to divide it across as many tables and views as you need to. Points will be deducted if your solution uses a trigger where a constraint or view would suffice, or if your solution is drastically over-complicated. For triggers, it is enough to specify which actions and tables it applies to, and PL/(pg)SQL pseudo-code of the function it executes.

The domain is the same as the previous questions, extended with revision history for comments, which contains all texts that a comment has had (more than one if they have been edited by the author).

```
Posts(idnr, author, text)

Comments(number, inPost, commenter, text, replyTo (or null))
  inPost -> Posts.idnr

Revisions(revisionId, inPost, number, text)
  (inPost, number) -> Comments(inPost, number)
```

Implement the following additional constraints in your design. Put letters in the margin of your code indicating where each constraint is implemented (possibly the same letter in several places):

a) If a comment is a reply, it must be to a comment with a lower number.
b) Whenever a comment is created or updated, its updated text should automatically be added to revision history.
c) If a post is deleted, its comments should automatically be deleted as well
d) Replies must be to an existing number (in the same post)
e) If there are multiple revisions of a comment, the revision with the highest revisionId should be the current (most recent) text of the comment.

Hint: Reference constraints do allow null values.
Hint: Remember to be careful about comparisons involving null values in constraints.

```sql
CREATE TABLE Posts
  ( idnr SERIAL PRIMARY KEY
  , author TEXT NOT NULL
  , text TEXT NOT NULL
  );

CREATE TABLE Comments
  ( number INT NOT NULL
  , inPost INT NOT NULL REFERENCES Posts ON DELETE CASCADE -- c
  , commenter TEXT NOT NULL
  , text TEXT
  , replyTo INT
  , PRIMARY KEY (number, inPost)
  , FOREIGN KEY (inPost, replyTo) REFERENCES Comments(inPost, number) -- a
  , CHECK (replyTo IS NULL OR replyTo < number) -- d
  );

CREATE TABLE Revisions
  ( revisionID SERIAL PRIMARY KEY  -- e
  , inPost INT NOT NULL REFERENCES Posts
  , number INT NOT NULL
  , text TEXT
  , FOREIGN KEY (number, inPost) REFERENCES Comments(number, inPost)
  );

-- b
CREATE OR REPLACE FUNCTION revise() RETURNS TRIGGER AS $$
    BEGIN
        INSERT INTO Revisions VALUES
          ( DEFAULT  -- e (or: SELECT MAX(revisionID)+1 FROM Revisions)
          , NEW.inPost
          , NEW.number
          , NEW.text
          );
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER revise
AFTER INSERT OR UPDATE ON Comments
FOR EACH ROW EXECUTE PROCEDURE revise();

-- Testing, not part of solution
INSERT INTO Posts VALUES (0, 'Jonas', '...');
INSERT INTO Posts VALUES (1, 'Matti', '...');
INSERT INTO Posts VALUES (2, 'Jonas', '...');
INSERT INTO Posts VALUES (3, 'Test', '...');

INSERT INTO Comments VALUES (0, 1, 'Jonas', 'First!', null);
INSERT INTO Comments VALUES (1, 1, 'Sanoj', '*sigh*', 0);
INSERT INTO Comments VALUES (2, 1, 'Jonas', ':(', 1);
INSERT INTO Comments VALUES (3, 1, 'Matti', 'I agree', 1);
INSERT INTO Comments VALUES (0, 0, 'Matti', 'Nice', null);
INSERT INTO Comments VALUES (0, 2, 'Jonas', 'Thoughts?', null);

UPDATE Comments SET text = 'foo';

SELECT * FROM Revisions;
```

Question 6: Semi-structured data and other topics (10 p, 3+4+3)

a) Suppose a fellow student claims that SQL injection vulnerabilities can never occur as long as the JDBC PreparedStatement class is used for queries. Either provide a counterexample to this (a sketch of Java code using PreparedStatement that is vulnerable to SQL injection) or if you agree with the claim provide a convincing explanation of why it is correct.

b) Study this JSON Schema for posts and comments (same domain as previous questions):

```
{
  "type": "array",
  "items": {
    "type": "object",
    "required": ["author","text","comments"],
    "properties": {
      "author":    {"type": "string"},
      "text":      {"type": "string"},
      "comments":  {"$ref": "#/definitions/comments"}
      }
    },

  "definitions": {
    "comments": {
      "type": "array",
      "items": {
        "type": "object",
        "required": ["commenter","text"],
        "properties": {
          "commenter": {"type": "string"},
          "text":      {"type": "string"},
          "replies":   {"$ref": "#/definitions/comments"}
        }
      }
    }
  }
}
```

Write a JSON document encoding this data, valid with the given schema:

idnr	author	text
0	Matti	...
1	Jonas	...

number	inPost	commenter	text	replyTo
0	0	Jonas	First!	null
1	0	Sanoj	*sigh*	0
2	0	Jonas	:(	1
3	0	Matti	I agree	1
0	1	Matti	Nice	null

idnr and number can mostly be ignored, except that if two items are in the same JSON array then the one with the lower number should be at a lower array index.

c) Based on the same JSON Schema, write a JSON Path expression that finds the texts of all comments written by Jonas (in all posts and including replies to comments). Should give two values for the example data in (b).

6 a) It is possible to have SQL injection vulnerabilities. Anything like

```
string x = getUserInput();
PreparedStatement ps = conn.prepareStatement(
  "DELETE FROM A where b='"+x+"'");
```

6 b)

```
[
{ "author" : "Matti",
  "text" : "...",
  "comments" : [
    {"commenter" : "Jonas",
     "text" : "First!",
     "replies" : [
       {"commenter" : "Sanoj",
        "text" : "*sigh*",
        "replies" : [
          {"commenter" : "Jonas", "text" : ":("},
          {"commenter" : "matti", "text" : "I agree"}
        ]
       }
     ]
    }
  ]
},
{ "author" : "Jonas",
  "text" : "...",
  "comments" : [
    {"commenter" : "Matti", "text" : "Nice"}
  ]
}
]
```

6 c) one solution:

$.*.comments.**?(@.commenter="Jonas").text

Another (simpler) one:

$.**?(@.commenter="Jonas").text