Databases Exam

TDA357 (Chalmers), DIT621 (University of Gothenburg)

2021-01-13 14:00-18:00

Department of Computer Science and Engineering

Examiner: Jonas Duregård, contact via Zoom supervision. In case of urgent technical issues, contact by phone: 031 772 1028

All examination aids are allowed. This includes browsing the Internet and using tools like psql. If you use any code or text you find anywhere, you need to include a reference to where you found it. You are not allowed to communicate with anyone other than the exam staff in any way during the exam, this includes publishing anything online if it is visible to anyone other than yourself.

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.

Instructions and submission: https://chalmers.instructure.com/courses/13904

If technical issues prevent you from accessing Canvas, send your solution to jonas.duregard@chalmers.se no later than 18:30 using this exact title:
TDA357 DIT621 exam submission

Question 1: ER-design (11 points, 6+5)

Submit a single PDF-file called q1.pdf for this question. You may draw the ER diagram by hand and scan it if the result is clearly readable.

a) Draw an ER-diagram for this domain:

You are making a database for recipes and ingredients. Every recipe consists of components, like "filling" or "glaze". Each component has its own set of ingredients, and an amount for each such ingredient.

The next page contains an example recipe that could be built by extracting data from this database.

Additional features:

- Each recipe has its own unique name. Each component has a name that is unique for the recipe it's in.
- Each recipe has an instruction text.
- The database should store the unit of measurement of each ingredient (e.g. butter is always measured in gram so if a component contains 100 of ingredient "butter", that means 100 grams).
- Some recipes require using an oven, for these recipes it should store temperature and time.
- Ingredients can have any number of alternate ingredients (globally for all recipes that use that ingredient). There is a conversion factor for each such alternate, e.g. 1 gram of sugar can be replaced by ¾ gram of honey (so the factor is 0.75).

b) Translate your diagram into a schema.

An example recipe for Cinnamon buns with three components ("Main ingredients", "Filling" and "Glaze"):

Uses an oven at 220 degrees Celsius for 6 minutes.

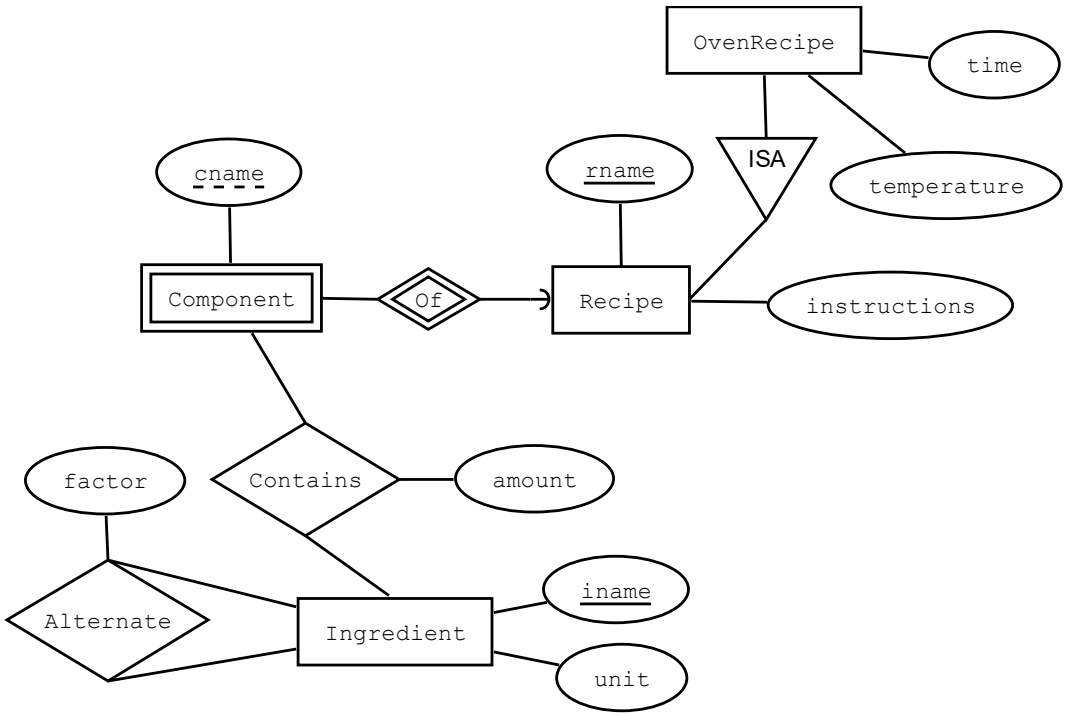
Main ingredients:
35 g yeast
100 g sugar
3 dl milk
1 x egg
120 g butter
1 tsp salt
1 tsp ground cardamom
750 g flour

Filling:
100 g butter
50 g sugar
2 tsp cinnamon

Glaze:
1 x egg
0.1 dl water
100 g pearl sugar

Instructions: Crumble the yeast in a bowl and stir in a few tablespoons of milk ...

Solution 1:



Recipes(rname, instructions)

OvenRecipes(rname, time, temperature)
rname -> Recipes.rname

Components(cname, recipe)
recipe -> Recipes.rname

Ingredients(iname, unit)

Contains(component, recipe, ingredient, amount)
(component, recipe) -> Components.(cname, recipe)
ingredient -> Ingredients.iname

Alternate(base, alternate, factor)
base -> Ingredients.iname
alternate -> Ingredients.iname

Question 2: Functional Dependencies, Normal Forms (8 points, 3+2+3)

*Submit a file called q2.txt for this task. You may use .pdf if you prefer. Do **not** use other document formats like .docx or .pages.*

a) Consider this (symbolic) table:

A	B	C	D
0	0	0	0
0	0	0	1
0	1	1	0
1	0	0	0
2	0	2	2

Identify two non-trivial functional dependencies that hold on this data and normalize it to BCNF. Provide the resulting schema, and the data in each relation of the schema as a table like the one above.

b) Briefly explain the concept of lossless join using your result from a) as an example.

c) Some random person claims that it's impossible to replace the '?' below so that $A \twoheadrightarrow B$ holds but neither $A \rightarrow B$ nor $A \rightarrow C$ holds on the table. Either prove the random person wrong (by constructing a table) or write a short and compelling argument for why they are correct.

A	B	C
0	?	?
0	?	?
0	?	?

Solutions 2:

All functional dependencies are those derived from C->B and A B->C, (the non-trivial derived ones are A C->B and C D->B).

The simplest possible decomposition (decomposing on C->B with {C}+={B,C} giving R1(B,C) and R2(A,C,D).

The data in those tables would be:

B	C
0	0
1	1
0	2

A	C	D
0	0	0
0	0	1
0	1	0
1	0	0
2	2	2

b) Taking the natural join of R1 and R2 gives the original table. Thus no information has been lost in the decomposition.

c) It is not possible. For A ->> B to be true, the 6 unknown cells would need to be the Cartesian product of two relations, but the only way to get 3 elements in a Cartesian product is if one of the operands has three elements and the other has 1. Basically it would have to be the Cartesian product of something like {0,1,2} and {0}, but then the resulting table would have a column (either B or C) with only 0's, and thus either A->B or A->C would hold.

^ this is a more detailed argument than you need for full points.

Question 3: SQL Queries (12 points)

Submit a plain text file called q3.sql (or q3.txt) for this task. The file does not have to be executable. Use SQL comments (--) for text and to indicate where each part starts.

Consider this schema for an online sales platform:

```
Items(itemname, price)
Categories(catname)
Categorized(item, category)
  category -> Categories.catname
  item  -> Items.itemname
Discounts(category, pricefactor)
  category -> Categories.catname
```

Items are things that can be sold. Each item has a base price. Some items belong to a category, and some categories have an active discount. The attribute pricefactor specifies a discount as a factor, e.g. a pricefactor of 0.75 means all items in the specified category are discounted by 25%.

Write an SQL query for solving each of these tasks:

a) Find the name and actual price of each item, factoring in discounts where applicable.

b) Find the largest difference in base price (ignoring discounts) between any two products in the same category. The result should be a single number. Items that have no category are irrelevant to this query.

c) Find the average price of all products that do not have a category. The result should be a single number.

Solution 3:

```sql
-- a)
SELECT itemname, price*COALESCE(pricefactor,1) AS price FROM Items
             LEFT OUTER JOIN (Discounts NATURAL JOIN Categorized)
             ON item=itemname;

-- b)
WITH
ItemsWithCats AS (SELECT * FROM Items JOIN Categorized ON item=itemname)
SELECT MAX(A.price-B.price)
FROM ItemsWithCats AS A JOIN
     ItemsWithCats AS B USING (category);

-- c)
SELECT AVG(price) AS averagePrice
FROM Items
WHERE itemname NOT IN (SELECT item FROM Categorized)
```

Question 4: Relational Algebra (8 points, 4+4)

Submit a single plain text file q4.txt, or a pdf called q4.pdf if you prefer.

You can use [] to indicate subscript text. You may substitute Greek letters with corresponding English ones based on this table: https://web.mit.edu/jmorzins/www/greek-alphabet.html (use capital X for cartesian product, add a text comment if you use any other symbols). Of course, using Unicode is also acceptable.

A database containing historical periods and historical events has the following schema:

```
Periods(pname, started, ended)
Events(ename, year)
```

Periods have a starting and ending year (both are inclusive) e.g. ('World war II', 1939, 1945) could be an entry. Events have a year when they occurred e.g.
('SQL became an ISO standard', 1987) could be an entry.

Write relational algebra expressions that solve these tasks:

a) Find the names of all events that occurred during any of the same historical periods as "The Great Collapsing Hrung Disaster" (a fictional event that you may assume is in the Events table). Make sure each such event occurs only once in the result.

To clarify: "The Great Collapsing Hrung Disaster" happened in some year, and that year is during some number of historical periods. Your job is to find all events that occurred during all those periods.

Hint: First write an expression for finding all periods the event is in, and then use it to find all events in those periods.

b) Find the name of the most eventful historical period(s). In other words, the period with the greatest number of events in it. May be more than one period only if there are several periods with the same number of events.

Solution 4:

This is a useful definition for both parts, for every event it lists all the periods it is in (or equivalently: for all periods it lists the events that are in it).

$$EP = \sigma_{year >= started\ AND\ year <= ended}(Events\ X\ Periods)$$

a)

These are the starting/ending years of the relevant periods:

$$Ps = \pi_{started,\ ended}(\sigma_{ename='The\ Great\ Collapsing\ Hrung\ Disaster'}(EP))$$

$$Result=\delta(\pi_{ename}(\sigma_{year >= started\ AND\ year <= ended}(Events\ X\ Ps)))$$

b)

$$ECount = \gamma_{pname,\ COUNT(*) \rightarrow eventCount}(EP)$$

$$MaxEC = \gamma_{MAX(eventCount) \rightarrow topCount}(ECount)$$

$$Result = \pi_{pname}(\sigma_{eventCount=topCount}(ECount\ X\ MaxEC))$$

Question 5: Views, constraints and triggers (11 points, 6+5)

Submit a plain text file called q5.sql (or q5.txt) for this task. The file does not have to be executable.

a) A company wants a simple database for storing messages, with at least the following interface (meaning tables and/or views):

Messages(id, sender, receiver, text, time)
RemovedMessages(id, sender, receiver, text, time)

Additional tables and views can also exist. The column "id" is a unique identifier for messages.

They also want you to write a single SQL statement that removes a message, meaning it disappears from Messages and appears in RemovedMessages. No message should ever appear both in Messages and RemovedMessages.

For this question you should:

- Write all tables and views for this database in SQL. Avoid using triggers if possible.
- Write a single SQL DML statement (So one DELETE, UPDATE or INSERT) that demonstrates how to remove a message, specifically your statement should remove the message with id 0 (if such a message exists).

b) The company further wants to keep at most the 100 latest removed messages for each message receiver, permanently deleting any older messages from the database. Explain in detail how you would solve this, providing enough pseudo-code and text descriptions for a novice SQL developer to finish your solution.

Solutions 5:

a) There are at least three possible solutions that don't use triggers, all of them have both Messages and RemovedMessages as views.

1. Have a single table, like Messages but with a Boolean value for isDeleted, and the views simply filter on that condition or its negation. The removal operation becomes an update that sets it to true.

2. Have a table for all messages, and one for the IDs of removed messages (with a reference to the other table). Messages uses NOT IN and RemovedMessages is just a join on the two tables. The removal operation becomes an insert into the table for removed messages.

3. Like 2 but the other way around, one for all messages and one for "Inbox", i.e. messages that have not been deleted. The removal operation becomes a delete on inbox.

b) Depending on which of the approaches above you use, a trigger after UPDATE, INSERT or DELETE could be used. If using a for each row trigger it should be enough to use an IF-statement that checks if there are more than 100 messages for the same receiver (as OLD or NEW) and delete the oldest row for that receiver if there is.

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

Submit a single plain text file called q6.txt (or q6.json if you prefer) for this question.

A company is using a tool for managing relational database designs as JSON documents. Basically, every document describes a relational schema, including primary keys etc. On the next page is a basic JSON Schema for the JSON Documents the tool accepts. The JSON Schema is also available as a file called q6schema.json on the exam page.

a) Write a JSON-document encoding the following relational schema, valid with respect to the given JSON Schema and containing all information in the way the JSON Schema intends.

```
Students(idnr, name)
Grades(student, course, grade)
  student -> Students(idnr)
```

b) Extend the JSON Schema with an additional feature, and an additional constraint. Detailed instructions:

- You should use at least two different features of JSON Schema (e.g. different keywords).
- Your additional feature should enable support for some additional relational schema element (the JSON Schema currently supports attributes, primary keys and references, what else could you have in relational schemas?).
- Your added constraint should prevent encoding of some incorrect relational schemas.
- Your solution should be a copy of the schema on the next page, with a couple of extra JSON Schema keywords added. Add a blank line before and after each section of code you have added to highlight it.
- Write a short explanation (one or two sentences) below your JSON Schema document for each of your additions. The explanation should describe what incorrect relational schemas you prevent (in words and/or an example) and what additional feature you support with an example.

c) Write a JSON Path query for finding the names of all primary key attributes for all tables in a JSON document that validates against the schema.

```
{
  "description": "Each item represents a table/relation",
  "type": "array",
  "items": {
    "type": "object",
    "required": [
      "table",
      "attributes"
    ],
    "properties": {
      "table": {
        "description": "The name of a table",
        "type": "string"
      },
      "attributes": {
        "description": "The list of attributes (a.k.a. columns) for a table",
        "type": "array",
        "items": {
          "type": "object",
          "required": [
            "attribute"
          ],
          "properties": {
            "attribute": {
              "description": "The name of an attribute",
              "type": "string"
            },
            "primaryKey": {
              "description": "true means this attribute is part of the primary key.",
              "type": "boolean"
            }
          }
        }
      },
      "references": {
        "description": "List of reference constraints (Foreign keys) on this table",
        "type": "array",
        "items": {
          "type": "object",
          "required": [
            "fromAtts",
            "toTable",
            "toAtts"
          ],
          "properties": {
            "fromAtts": {
              "description": "The local attributes of a reference (left side of ->)",
              "type": "array",
              "items": {
                "type": "string"
              }
            },
            "toTable": {
              "description": "The name of the table the reference refers to",
              "type": "string"
            },
            "toAtts": {
              "description": "The name of the attributes in the referenced tables.",
              "type": "array",
              "items": {
                "type": "string"
              }
            }
          }
        }
      }
    }
  }
}
```

Solution 6:

a) This is the "minimal" solution, that omits "primaryKey" and "references" (the latter assumed to mean having no references, same as []). Including them is also fine.

```
[{
    "table": "Students",
    "attributes": [{
      "attribute": "idnr",
      "primaryKey": true
    }, {
      "attribute": "name"
    }]
  },
  {
    "table": "Grades",
    "attributes": [{
        "attribute": "student",
        "primaryKey": true
      }, {
        "attribute": "course",
        "primaryKey": true
      }, {
        "attribute": "grade"
      }

    ],
    "references": [{
      "fromAtts": ["student"],
      "toTable": "Students",
      "toAtts": ["idnr"]
    }]
  }
]
```

b) The easiest constraint to add is probably that a table needs at least one attribute (using array length requirements) or similarly that a name needs to have at least one character. More difficult things could be using regular expressions to check that names are valid (does not give extra points). Some things like making sure every table has a primary key or that attributes in references match up might not be possible without significant rewrites (which was not really the intention but can be accepted if the result works well).

The easiest relational schema feature to add is probably uniqueness constraints, it would basically be a list of lists of strings for a property in table objects (so [[“A”,”B”],[“C”]] would mean (A,B) unique and C unique). Things like check constraints also work, but the conditions would presumably just be strings (no way to check that they are correct logic expressions). Nullability is another super-easy one, it's just like "primaryKey" (a Boolean value for each attribute).

c) something like $[*].attributes[*]?(@.primaryKey).attribute