Databases Exam

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 along with this 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, 5+5)

a) Your task is to make an ER-diagram for part of the database of a public transportation system, similar (but not identical) to that of Gothenburg.

The database should contain stations, routes and vehicles. Each station has a name and a location. Each route has a number (like line 10 or line 16). Vehicles include busses, trams and boats (the database should store the type of each vehicle) and each vehicle is identified by a serial number. Each vehicle can traffic a single route at any given time, and the database should store which vehicles are currently trafficking which routes (intended to be updated in real time as vehicles start/stop driving).

In addition to the basic information above, the system should store:

- All starting times for each route, and what station each start happens at (can be different for different starts of the same route).
- Which stations each route makes a stop at, along with how many minutes it takes to reach it from the start of the route (you may assume that each route can only stop at any single station once).
- Some stations are ports (can accommodate boat traffic), the database should contain the maximum draught (depth of the boats keel) of these ports.

You do NOT need to enforce that boats only visit ports.

Make reasonable assumptions about key attributes of entities based on the domain description.

b) Translate this (symbolic) ER-diagram into a relational schema, including keys and constraints. **Hint:** May be harder than it looks, think carefully about those keys!
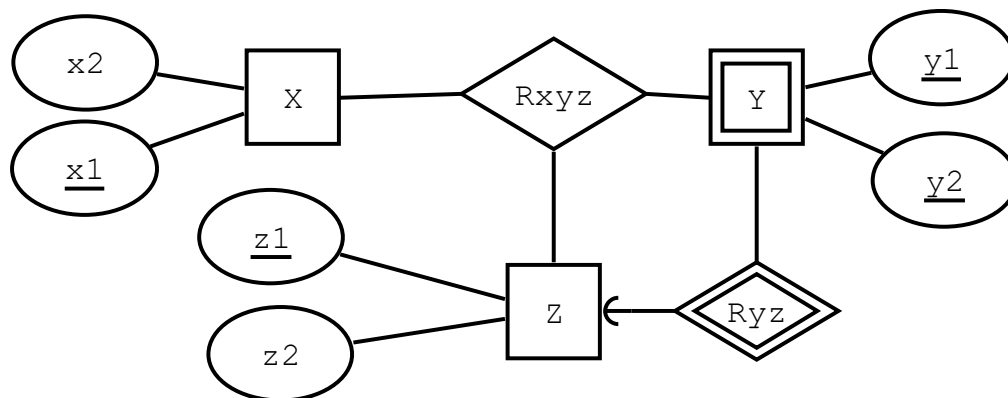
(yes - I was too lazy to come up with another domain for this...)

Question 2: Functional Dependencies, Normal Forms (9 p, 3+3+4)

Here is a table intended to store grades and personal information of students at a school. Each row contains the grade a student got on a course and how many academic credits the course is worth.

student_id	student_name	course_id	grade	credits
1	Hermione	TDA357	Outstanding	7.5
1	Hermione	EDA452	Outstanding	7.5
2	Harry	EDA452	Poor	7.5
3	Ron	TDA666	Troll	10
4	Draco	TDA357	Acceptable	7.5
4	Draco	TDA666	Outstanding	10

a) Give concrete examples of an update anomaly and a deletion anomaly that could happen here. For each example, give a short and clear description of what action is taken, and the unintended consequence of that action.

b) Based on your understanding of the domain, identify at least 3 sensible functional dependencies that should hold on the attributes of the table. If you make any non-obvious assumptions on the domain when choosing your functional dependencies, please briefly describe them.

c) Decompose the relation into BCNF based on the functional dependencies from b), your answer should contain:

- The resulting relational schema (the steps to get it are not needed, adding references is not needed)
- The number of rows each relation would contain to encode the same data as the original table (you do not have to specify the exact contents, just the number of rows).

Question 3: SQL Queries (9 p, 3+3+3)

Below is the schema for a database of users sending messages to each other. Readtime is either a time when the message was read by the receiver, or null if it has not been read yet (no other attributes can be null). Logintime is the last time the user was logged in.

Users(<u>username</u>, email, logintime)

Messages(<u>sender</u>, <u>receiver</u>, content, <u>sendtime</u>, readtime)
  sender -> Users.username
  receiver -> Users.username

a) Write an SQL query that finds sender username, sender email, and content of each unread message sent to the user 'admin', with the oldest message first in the result.

b) Write an SQL query that calculates the average time between reading and sending messages (for messages that have been read). Assume sendtime and readtime are timestamps or similar, so the expression `readtime-sendtime` gives the time between sending and reading (if the message has been read).

c) Write a query that finds for each user, which user they have sent the most messages to, or null in cases of users that have not sent any messages. The result should have two columns, one for the sender and one for the most common receiver. If there are ties, the same sender may occur more than once with different receivers, or you may include any one of the most common receivers.

Question 4: Relational Algebra (9, 3+3+3)

Consider this relational schema for part of the inventory for a clothes store.

Items(<u>idnr</u>, name, quantity, <u>size_number</u>)

Sizes(<u>designation</u>, min_number, max_number)
  min_number ≤ max_number

Items contain the id and name of all items the store sells, quantity is the number of items currently in store, and size_number is a numeric measure of the size of the item.

Sizes contains designations (like 'S', 'M', 'L', 'XL' etc,) of classes of sizes, and the (inclusive) size interval the class includes, so a row like ('S', 36, 38) would say that all numeric sizes from and including 36 to and including 38 would be considered size 'S'. The intervals may overlap, e.g. if the relation includes size designations from different countries, so a single item may belong in several size designations.

a) Write a relational algebra expression for the name and quantity of all items with sizes designated 'L' and/or 'XL'.

b) Write a relational algebra expression that finds the designation of all size classes that are always strictly larger than 'M' (for instance it could give 'L' and 'XL' if all sizes in those classes are larger than all sizes in the 'M'-class).

c) Write a relational algebra expression that calculates the <u>total</u> quantity of available items with idnr 1337 and sizes with designation 'L'. The result should have a single column, for the total quantity and always give a single row.

Question 5: Views, constraints and triggers (12 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. It does not have to be a syntactically correct trigger, but it needs to be close enough that it is clear how you intend for it to work.

Your task: The database contains dots in a two-dimensional plane and connections between those dots. The external interface for queries should look like this (constraints not included):

Dots(x_pos, y_pos, idnr, radix)

Connections(from_idnr, to_idnr)

This interface can include views and tables, and additional views and tables may be created as needed. Each row in `Dots` is a dot, with a position (`x_pos, y_pos` are integer coordinates), an identification number and a radix (see below). Each row in `Connections` represents a line from one dot to another, identified by their id-numbers.

Implement the following 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) There is at most one dot on any position (x,y-pair) and each dot has a unique idnr.
b) Dots can only connect to valid dots (those that are in `Dots`), and dots cannot be connected to themselves. Also there cannot be multiple lines to/from the same dots.
c) The radix of a dot should be equal to the number of dots it is connected to.
d) The radix cannot exceed eight. Attempting to add additional connections should fail.
e) All connections are bi-directional, meaning if there is a connection from point A to point B, there must also be a connection from point B to point A.
f) If a dot is deleted, all its connections should be automatically deleted as well.

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

Read all parts of the question before you start, the answers are not independent.

In this task you will store a log for a meteorological measuring station. The log keeps tracks of two kinds of data: Performed measurements, and requests for information from users.

Here is an example of a log in plain text, containing a single measurement and a single request:

```
Measurement time=1560000000,
            temperature=21,
            windspeed=0,
            rainfall=Error 213 no value found

User Jonas requested temperature, rainfall
    replied with time=1560000000,
                  temperature=21,
                  rainfall=Error 213 no value found
```

A measurement always contains a timestamp, and values for a number of additional parameters, in the example above the additional parameters are temperature, windspeed and rainfall, but they can be any set of text strings. The values are either numbers or errors. Errors contain an error code (213 in the example) and an error text ("no value found" in the example). The order of parameters (including time) is not important. The number and names of parameters can differ between measurements in the log.

A user request contains a username and a list of parameters (order not important). It also contains a reply consisting of parameters and values/errors (note the similarity to measurements). You may assume that replies always contain a time parameter. Your schema does NOT have to enforce that the reply contains all the requested parameters or that values are correct in context.

Each whole log should be a single JSON document containing multiple measurements and served requests (order is important). Each entry must be either a measurement or a request.

a) Encode the example log above in a JSON document.

Regarding syntax: Missing a few quotation marks is not a big deal, but make sure the structure and what type of JSON elements you are using is clear.

b) Write a JSON Schema for logs. Your solution to a) should be valid in your schema and it.

c) Write a JSON Path expression for finding all temperature values in measurements (but not in replies!). The expression should work on any log validated by your schema. The result should be an array containing values/errors.