University of Toronto
CSC343, Fall 2019

Assignment 3

Due: Wednesday, December 4 before 8pm!

IMPORTANT: A summary of key clarifications for this assignment will be provided on Piazza. Check it regularly for updates. Both the FAQ and any Quercus announcements are required reading.

Learning Goals

By the end of this assignment you should be able to:

- identify tradeoffs that must be made when designing a schema in the relational model, and make reasonable choices,
- express a schema in SQL's Data Definition Language,
- formally reason about functional dependencies,
- appreciate differences between what DDL can express, what an XML DTD can express, and what a JSON schema can express, and
- recognize differences in the strengths of the relational model and the semi-structured model for representing a domain.

Part 1: Informal Relational Design

In class, we are in the middle of learning about functional dependencies and how they are used to design relational schemas in a principled fashion. After that, we will learn how to use Entity-Relationship diagrams to model a domain and come up with a draft schema which can be normalized according to those principles. By the end of term you will be ready to put all of this together, but in the meanwhile, it is instructive to go through the process of designing a schema informally.

The domain

Suppose you've been hired by LuxuryRentals.com to build the database back-end for their vacation rental application. Below is the information that they want to be able to store for their first proof-of-concept version of the database. There is much more to be added later, such as availability of properties and payment information, but this is not your responsibility.

Guests rent properties that each have a single host. (A person can act as host for multiple properties, however.) Renting is only by the week, and can be for any number of weeks. We refer to it as one "rental" of the property even if it is for multiple weeks. Each property can have a different price, and its price can be the same for all weeks, but it can also be different.

For all properties, a week is defined to start on a Saturday. But each property has its own number of bedrooms, number of bathrooms, capacity (for example, "sleeps 8"), and address. The capacity of the property must be at least the number of bedrooms; properties that don't meet this requirement cannot be listed on LuxuryRentals.com. Some properties are considered to be city properties, and some are considered to be on the water. It is possible that a property is both a city property and a water property. City properties have a walkability score (0 to 100) and the type of transit that is closest (either bus, LRT, subway, or none). Water properties have a water type (either beach, lake, or pool), and may or may not provide lifeguards at certain times. We only need to store whether or not lifeguarding is ever offered, not the times. A water property can have multiple water types. In that case, we need to record for each whether lifeguarding is available.

Because these are luxury rentals, every property includes one or more luxuries from this list: hot tub, sauna, laundry service, daily cleaning, daily breakfast delivery, and concierge service. Regardless, the price for a week is all-inclusive; there are no extra charges for anything.

When renting, one guest is the "renter" and has responsibility for paying and taking care of the property. They must be at least 18 years old on the first day of the rental. The renter can have 0 or more other guests registered to use the property with them, for example, a group of friends or a family. These guests are registered for the whole rental; they cannot be registered for only parts of it. However, it is possible for a person to make two separate rentals of the same property on adjacent weeks, and to associate different guests with one rental than the other.

The number of guests, including the renter, must not exceed the sleeping capacity of the property. Every guest, whether or not they are the official renter, has a name, address, and date of birth recorded. The renter additionally has to provide a credit card number.

Guests can optionally rate the the property with 0 to 5 stars. Hosts can also be rated, on the same star system, but only the renter can rate the host. All guests may also leave a comment, but only if they have rated the property.

For now, the only thing we are recording about each host is their email address. It is allowed for two different hosts to have the same email address.

Define a schema

Your first task is to construct a relational schema for our domain, expressed in DDL. Write your schema in a file called `schema.ddl`.

As you know, there are many possible schemas that satisfy the description above. We aren't following a formal design process for Part 1, so instead follow as many of these general principles as you can when choosing among options:

1. Avoid redundancy.
2. Avoid designing your schema in such a way that there are attributes that can be null.
3. If a constraint given above in the domain description can be expressed without assertions or triggers, it should be enforced by your relational schema.

You may find there is tension between some of these principles. Where that occurs, prioritize in the order shown above. Use your judgment to make any other decisions. In addition, you should:

- Define a primary key for every table.
- Use `UNIQUE` if appropriate to further constrain the data.
- Define foreign keys as appropriate.
- For each column, add a `NOT NULL` constraint unless there is a good reason not to.

To facilitate repeated importing of the schema as you correct and revise it, begin your DDL file with our standard three lines:

```
drop schema if exists vacationchema cascade;
create schema vacationschema;
set search_path to vacationchema;
```

Document your choices

At the top of your DDL file, include a comment that answers these questions:

1. What constraints from the domain could not be enforced, if any?
2. What constraints that could have been enforced were not enforced, if any? Why not?

Instance and queries

Once you have defined your schema, create a file called `data.sql` that inserts data into your database. You get to decide on how to format and store that data. You may find it instructive to consider this data as you are working on the design. Then, write queries to do the following:

1. For each type of luxury (hot tub etc.), report the number of properties that offer that luxury.
2. A rental is considered to be "at capacity" if it involves a group of guests (including the renter themself) that is big enough to reach the capacity of the property. Report the average property rating for at-capacity rentals, as well as the number of such rentals. Do the same for rentals that were below capacity.
3. Find the host/hosts with the highest average host rating. Report their email address, average host rating, and the price for the most expensive booking week they have every recorded.
4. For each type (city property, water property, and other) report the average number of extra guests (that is, not including the renter themself) for properties of that type. Compute the average across all rentings of that type of property. Each renting should contribute once to the average, even if it is for multiple weeks.
5. For each property, report the highest price ever charged for a week renting that property, the lowest price, and the range. Include a column that has a star in it for any property/properties with the highest range.

We will not be autotesting your queries, so you have latitude regarding details like attribute types and output format. Make good choices to ensure that your output is easy to read and understand.

Write your queries in files called `q1.sql` through `q5.sql`. Further instructions about the structure of these query files and how correctness is to be assessed will be provided via an announcement on Quercus.

What to hand in for Part 1

Hand in plain text files `schema.ddl`, `data.sql`, and `q1.sql` through `q5.sql`. The Quercus announcement may specify a few additional files to hand in in order for us to assess correctness of your queries.

How Part 1 will be marked

Part 1 will be graded for design quality, including: whether it can represent the data described above, how well it enforces the constraints described, good justification of choices and tradeoffs made, avoiding redundancy, avoiding unnecessary NULLs, and good use of DDL (choice of types, NOT NULL specified wherever appropriate, etc.) Your queries will be assessed for correctness. More details on this will be announced on Quercus, as noted above.

Your code will also be assessed for these qualities:

- View and column names: Does every view and every column have a name that will assist the reader in understanding the view quickly?
- Comments:
  Does every view have a comment above it specifying clearly exactly what rows get to be in this table? Comments should describe the data (*e.g.*, "The student number of every student who has passed at least 4 courses.") not how to find it (*e.g.*, "Find the student numbers by self-joining ..."). Put comments *before* the

view definition. Together, the comments and the names should tell the reader everything they need to know in order to use the view, *without having to read the code.*

- Formatting according to these rules:
  - An 80-character line limit is used.
  - Keywords are capitalized consistently, either always in uppercase or always in lowercase.
  - Table names begin with a capital letter and if multi-word names, use CamelCase.
  - atttibute names are not capitalized.
  - Line breaks and indentation are used to assist the reader in parsing the code.

Thought questions

These questions will deepen your appreciation of issues concerning design, efficiency, and expressive power. They are for your learning, not for marks. Feel free to discuss them with each other, or with me in class or office hours.

1. Suppose we allowed you to be less strict in following the design principles listed above. Describe one compromise you would make differently.
   (a) How would the schema be different?
   (b) What would be the benefits of this new schema?
   (c) What would be lost in this new schema?
   (d) Why would you make this different compromise?
2. At the end of the course we will discuss the semi-structured data model, and will see at least one example of it: XML and/or JSON. Think about these questions:
   (a) What aspects of the data were awkward to express in SQL? Would they be easier in JSON or in XML?
   (b) Are there constraints that could not be expressed in SQL but that can be expressed in JSON or in and XML DTD?
   (c) Are there constraints that you *could* express in SQL that cannot be expressed in JSON or in and XML DTD?

Part 2: Functional Dependencies, Decompositions, and Normal Forms

1. Consider a relation A with attributes $LMNOPQRS$ with functional dependencies D:

$$D = \{\ L \rightarrow NQ,\quad MNR \rightarrow O,\quad O \rightarrow M,\quad NQ \rightarrow LS,\quad S \rightarrow OPR\ \}$$

   (a) State which of the given FDs violate BCNF.

   (b) Employ the BCNF decomposition algorithm to obtain a lossless and redundancy-preventing decomposition of relation R into a collection of relations that are in BCNF. Make sure it is clear which relations are in the final decomposition, and don't forget to project the dependencies onto each relation in that final decomposition. Because there are choice points in the algorithm, there may be more than one correct answer. List the final relations in alphabetical order (order the attributes alphabetically within a relation, and order the relations alphabetically).

2. Consider a relation P with attributes $ABCDEFGH$ and functional dependencies T.

$$T = \{\ AB \rightarrow C,\quad C \rightarrow ABD,\quad CFD \rightarrow E,\quad E \rightarrow B,\quad BF \rightarrow EC,\quad B \rightarrow DA\ \}$$

   (a) Compute a minimal basis for T. In your final answer, put the FDs into alphabetical order. Within a single FD, this means stating an FD as $XY \rightarrow A$, not as $YX \rightarrow A$. Also, list the FDs in alphabetical order ascending according to the left-hand side, then by the right-hand side. This means, $WX \rightarrow A$ comes before $WXZ \rightarrow A$ which comes before $WXZ \rightarrow B$.

   (b) Using your minimal basis from the last subquestion, compute all keys for P.

   (c) Employ the 3NF synthesis algorithm to obtain a lossless and dependency-preserving decomposition of relation P into a collection of relations that are in 3NF. Do not "over normalize". This means that you should combine all FDs with the same left-hand side to create a single relation. If your schema includes one relation that is a subset of another, remove the smaller one.

   (d) Does your schema allow redundancy? Explain how you know that it does or does not.

Show all of your steps so that we can give part marks where appropriate. There are no marks for simply a correct answer. If you take any shortcuts, you must explain why they are justified.

What to hand in for Part 2

Type your answers up using LaTeX or Word. Hand in your typed answers, in a single pdf file called part2.pdf.

Final Thoughts

Declare your group now: Well before the due date, declare your team (even if you are working solo) on MarkUs. It is impossible to do so during the grace period, even if you have grace points you are going to use or an extension.

Submission: Check that you have submitted the correct version of your files by downloading it from MarkUs; new files will not be accepted after the due date.

Some parting advice: It will be tempting to divide the assignment up with your partner. Remember that both of you probably want to answer all the questions on the final exam.