University of Toronto
CSC343, Winter 2025

Assignment 3

Due: Wednesday, April 2nd by 4:00pm

IMPORTANT: A summary of key clarifications for this assignment will be provided in an FAQ 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,
- identify limitations to the constraints that can be expressed using the DDL,
- appreciate scenarios where the rigidity of the relational model may force an awkward design,
- formally reason about functional dependencies, and
- apply functional dependency theory to database design.

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 meantime, it is instructive to go through the process of designing a schema informally.

The domain

Gotta Love Games (GLG) is a student club at the University of Toronto dedicated to board game enthusiasts. They are seeking your assistance in creating a proof-of-concept version of a database to manage various aspects of their operations, such as tracking their inventory of board games, club members, and events they organize. The information outlined below is what the database needs to store initially. While additional features will be added later, that will not be part of your current task.

- The club needs to maintain name, email-id and level of study (Undergradute, Graduate, Alumni) of all club members.
- Executive members of the club that help run the club have extra data associated to them with their current role in club and the date since they assumed that responsibility. All executive members are also members of the club.
- Board games have titles, category (Strategy, Party, Deck-building, Role-playing, Social-deduction), a minimum and a maximum player limit, publisher and a release year. The club also wants to track when a game was acquired and the physical condition (New, Lightly-used, Worn, Incomplete, Damaged) it is currently in. "Incomplete" games can still be played while "Damaged" ones cannot.

- GLG owns multiple copies of some games and that needs to be tracked.
- GLG holds several events which have a name, location and date.
- Events with the same name can occur multiple times. For example, they hold a weekly event that happens every week and some special events that happen once in a term or year.
- There can be multiple game sessions held at any event each involving a particular board game that was played by some members of the club and is facilitated by one exec member of the club.
- All game sessions are assumed to run for the entire duration of the event. No member can participate in two game sessions happening at the same time.
- No exec member should be facilitating two game sessions at once. They should also not be playing any other game while they are facilitating a game session, they can however be playing the same game as they are facilitating.
- Events have an organizing committee which is comprised of multiple exec members (one of which is a organizing lead) and must always have a lead exec member. The organizing team stays the same for the multiple occurences of any event. For example, all weekly game nights are organied by the same exec team.

Some features above are not realistic, but they simplify your assignment. If we have not constrained something, assume it is unconstrained. For example, if I said houses have windows but didn't constrain it further, you should be prepared that a house may have no windows, 1 window, or many windows.

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. There is no single right answer we are looking for. Instead, we are looking to see how the schema you choose deals with the principles below.

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 schema, unless you can articulate a good reason not to do so.
4. There may be additional constraints that make sense but were not specified in the domain description. You get to decide on whether to enforce any of these in your DDL.

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. Additional requirements:

- Define appropriate constraints, i.e.,
  - 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.
- All constraints associated with a table must be defined either in the table definition itself, or immediately after it.

- 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 A3GLG CASCADE;
CREATE SCHEMA A3GLG;
SET SEARCH_PATH TO A3GLG;
```

You may create IDs, or define additional columns if you feel this is appropriate. Use your judgment.

There may be things we didn't specify that you would like to know. In a real design scenario, you would ask your client or domain experts. For this assignment, make reasonable assumptions. Keep track of these in writing, as we will ask you to articulate them at the top of your DDL file.

Document your choices and assumptions

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

Could not: What constraints from the domain specification could not be enforced without assertions or triggers, if any?

Did not: What constraints from the domain specification could have been enforced without assertions or triggers, but were not enforced, if any? Why not?

Extra constraints: What additional constraints that we didn't mention did you enforce, if any?

Assumptions: What assumptions did you make?

Instance and queries

Once you have defined your schema, create a file called `data.sql` that inserts data into your database. This file should insert the data that is described informally in file `glg-data.txt`. You will also need to insert more data of your own choosing. You may find it instructive to consider this data as you are working on the design.

Then, write queries to do the following:

1. Find the percentage of club members who have participated in at least one game session at each event (only include events with game sessions).
2. For each game in the inventory, report the total number of times it has been played in game sessions.
3. Find the board game that has been facilitated the most by a single exec member.
4. Identify the club member who has participated in the highest number of unique game sessions as a player.
5. For each event, report the average number of participants per game session.

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`. Download file `runner.txt`, which has commands to import each query one at a time. Once all your queries are working, start postgreSQL, import `runner.txt`, and cut and paste your entire interaction with the postgreSQL shell into a plain text file called `demo.txt`. We will assess the correctness of your queries based only on reading demo.txt, so it must show the results of the queries. There is no need to insert into tables (since we are not autotesting).

There will be lots of notices, like: Eg. INSERT 0 1, psql:q2.sql:16: NOTICE: view "blah" This is normal, and we are expecting to see it.

What to hand in for Part 1

Hand in plain text files `schema.ddl`, `data.sql`, and `q1.sql` through `q5.sql`, and `demo.txt`. These must be plain text files.

IMPORTANT: You must include the demo file, and it must show the output of your queries, or you will get zero for this part of the assignment.

How Part 1 will be marked

Part 1 will be graded for design quality, including: whether it can represent the data described above, appropriate enforcement of the constraints described, avoiding redundancy, avoiding unnecessary NULLs, following the priorities given for any tradeoffs that had to be made, and good use of DDL (choice of types, NOT NULL specified wherever appropriate, etc.) Your queries will be assessed for correctness, as evidenced by the `demo.txt`.

Your code will also be assessed for these qualities:

- Names: Is every name well chosen so that it will assist the reader in understanding the code quickly? This includes table, view, and column names.
- Comments:
  Does every table or view have a comment above it specifying clearly exactly what a row means? Together, the comments and the names should tell the reader everything they need to know in order to use a table or view. For views in particular, 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 . . .").
- 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.
  - attribute names are not capitalized.
  - Line breaks and indentation are used to assist the reader in parsing the code.

Part 2: Functional Dependencies, Decompositions, and Normal Forms

In your answers for this part, please list all attributes in final relations and FDs for each part in alphabetical order. This will make it easier for the graders to read your answers. Within each individual FD, this means stating an FD as $XY \rightarrow ABC$, not as $YX \rightarrow BCA$. 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 $WXZ \rightarrow B$. You could also combine FDs with the same LHS in your final answer.

1. Consider a relation R_1 with attributes $ABCDEFGHIJ$ and functional dependencies F_1, where:

$$F_1 = \{AB \rightarrow C, CD \rightarrow E, DE \rightarrow F, B \rightarrow D, EF \rightarrow H, H \rightarrow G, GH \rightarrow I, I \rightarrow J\}$$

   (a) Identify which functional dependencies prevent R_1 from satisfying BCNF.
   - Justify why each violating functional dependency does not satisfy the BCNF condition based on the definition of superkeys.

   (b) Apply the BCNF decomposition algorithm to decompose R_1 into a lossless, redundancy-free set of BCNF relations.
   - Step-by-step decomposition: At each step, identify the violating dependency, determine the new relations, and project the functional dependencies onto the decomposed relations.
   - Ensure the final relations are alphabetically ordered, both in terms of relations and attributes within each relation.

   (c) Does your final schema preserve all functional dependencies? Provide a formal proof using attribute closures. If dependencies are lost, discuss how dependency preservation could be ensured.

   (d) Use the Chase Test to verify that your decomposition is lossless. Clearly demonstrate each step of the test.

   (e) Find a decomposition of R_1 into $n-1$ relations, R_2, R_3, ..., R_n such that:
   - R_i is in BCNF for all $i, 2 \leq i \leq n$
   - R_2, R_3, ..., R_n share at least one attribute in common (i.e., $R_2 \cap R_3 \cap ... \cap R_n$ is non-empty)
   - $R_2 \bowtie R_3 \bowtie ... \bowtie R_n$ does not reconstruct R_1 (i.e., the decomposition is lossy)

2. Consider a relation R_2 with attributes $KLMNOPQRS$ and functional dependencies F_2, where:

$$F_2 = \{KLS \rightarrow M, MN \rightarrow PQ, NP \rightarrow QR, PQ \rightarrow R, RS \rightarrow O, S \rightarrow L\}$$

   (a) Compute a minimal basis for F2.
   - Show the process of removing extraneous attributes, eliminating redundant dependencies, and ensuring minimality.
   - Present the final functional dependencies in alphabetical order, sorting by left-hand side and then right-hand side.

   (b) Compute all candidate keys for R2 using the minimal basis.
   - Show all calculations using attribute closures.
   - Explain why each candidate key qualifies as a minimal superkey.

   (c) Apply the 3NF synthesis algorithm to decompose R2 into a lossless, dependency-preserving collection of 3NF relations.
   - 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) Prove that your decomposition is lossless using the Chase Test. Clearly document each step of the test.

   (e) Does your final schema allow redundancy? If so, explain where redundancy might occur and whether it can be reduced while maintaining dependency preservation.

Show all of your steps so that we can give part marks where appropriate. There are no marks for simply a correct answer. You must justify every shortcut that you take.

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 a3.pdf. Handwritten submissions will not be accepted and will receive a grade of 0.

Final Thoughts

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.

Forming groups: Make sure you have created your group in MarkUs as soon as you decide to work with a partner. Make sure you both see the group as expected. The deadline for getting assistance from us in creating your groups is Tuesday, April 1 (one day before the deadline). We strongly recommend you create your group before then.

Marking: The marking scheme will be approximately this: Part 1 70%, and Part 2 30%.

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. A reminder from the syllabus: "If you choose to work with a partner, we expect that you are working together on all parts of the assignment. If you choose to work with a partner in another way, you are responsible for all issues that arise from splitting up the work."