Problem Set 1: The Relational Model

COMP_SCI 399

Due: Thursday, September 28, 2023

Name: Zhuoyuan Li

1 Background (3 pts)

1. In your own words, briefly answer these questions about the makeup of a database.

   (a) (1 point) In a relational database management systems, the terms "table" and "relation" are sometimes used interchangeably. Speaking formally, what is the difference between a relation and a table?

   relation is a set of tuples, which doesn't allow duplicates.

   table is a set of identically structured records in tabular form. which allows duplicates

   (b) (1 point) What is a transaction? Why do we need them in a database management system?

   A transaction is an unordered list of queries that must either all succeed to take become visible or all fail without affecting the databases.
   we need to use transactions to make sure actions we take is consistent
   transactions ensure the correctness under concurrency

   (c) (1 point) Database management systems are everywhere. Why do we study them as computer scientists?

   Data is a crucial part of our lives, and we need to make sure data is correctly stored, interpreted, and mapped. we need to make sure data is preserved correctly, and we won't lose any information about it.

2 Database Design (3 points)

Recall that when we design our database instance, we often start out with a conceptual schema that lists the table name and its columns. For example, consider the student table described in the Caesar example in Lecture 1. We could describe its conceptual schema as:

student(<u>netid: text</u>, name: text, degree_program: text);

This denotes our three columns and that all contain strings. Moreover, the NetID column is underlined to uniquely identify each student.

Let's design a schema for the following application. We wish to write a system for managing our lists for packing for different types of trips such as vacations, business trips, and so on. We want to describe the following entities:

- Each *packing list* has an unique identifier, a descriptive name, and a suitcase associated with it.
- Each *suitcase* has an unique identifier, a description, its volume (in whole cubic centimeters), and its weight limit (in whole grams).
- Each *item* has a unique identifier, a list ID, a text description, its volume (in cm^3), its weight, and a (t/f) flag indicating whether it has been packed yet. The packed flag is set to false by default for all entries.

We want to keep track of our packing and be able to automatically verify if we have enough room in our bag before adding an item to it. Don't worry about implementing this part!

(a) (1 point) Create a conceptual schema for your packing application.

packing list: <u>plid</u> (text), name(text), sid (text)

suitcase: <u>sid</u> (text), description(text), volume(float), weightLimit(float)

items: <u>iid</u> (text), plid (text), description(text), volume (float), weight(float), isPacked(boolean)

(b) (2 points) Create a schema definition for this using **CREATE TABLE** statements. Make sure to include any integrity constraints. Don't worry about rejecting inputs that will cause a suitcase to go over its volume or weight limits.

This table should be created second

```
CREAT TABLE
      PackingList ( plid text,
                    name text,
                    sid text,
                    PRIMARY KEY (plid),
                    FOREIGN KEY (sid) REFERENCES Suitcase(sid))
```

This table should be created first

```
CREAT TABLE
      Suitcase ( sid text,
                 description text,
                 volume float,
                 weightLimit float,
                 PRIMARY KEY (sid))
```

This table should be created third

```
CREAT TABLE
      Items ( iid text,
              plid text,
              description text,
              volume float,
              weight float,
              isPacked boolean DEFAULT False,
              PRIMARY KEY (iid),
              FOREIGN KEY (plid) REFERENCES PackingList(plid))
```

3 Schema Design

3.1 Schema Normalization (2 pts)

Consider the following (denormalized) schema for describing complaint registers ("CRs") filed by community members against officers in the Chicago Police Department:

allegation(crid, accused_badge_no, allegation_name, incident_datetime, incident_address, investigating_agency, is_officer_complaint, accused_name, accused_birth_year, accused_gender, accused_race, findings, penalty);

We'll be extending this example in the first lab. For this longer example, you may disregard the column types. A few details about this data.

- A given complaint register might name more than one officer. The complainant may accuse a given officer of more than one act of misconduct in the same report. We call each of these accusations an allegation. For example, a CR might allege, "Officer A stole my wallet and Officer B illegally searched my car.". This would be two allegations. A given officer might be named in greater than one allegation.
- The system opens a case when someone (who may or may not be an officer) files a complaint. Depending on the nature of the allegations, either the Civilian Office of Police Accountability (COPA) or the Bureau internal affairs (BIA) will investigate it.
- We can uniquely identify a row in this table using the CRID, badge number and allegation name columns.
- We use badge numbers to uniquely identify the officers for now

We have the following functional dependencies:

1 $crid \Longrightarrow (incident\_datetime, incident\_address, investigating\_agency, is\_officer\_complaint)$

2 $accused\_badge\_no \Longrightarrow (accused\_name, accused\_birth\_year, accused\_gender, accused\_race)$

3 $(crid, accused\_badge\_no, allegation\_name) \Longrightarrow (findings, penalty)$

Normalize this schema by iteratively applying the functional dependencies. Please show your work.

Space for answer:

From ① FD, we can have:
(<u>Crid</u>, incident_datetime, incident_address, investigating_agency, is_officer_Complaint)

From ② FD, we can have:
(<u>accused_badge_no</u>, accused_name, accused_birth_year, accused_gender, accused_race)

From ③ FD, we can have:
(<u>Crid</u>, <u>accused_badge_no</u>, <u>allegation_name</u>, findings, penalty)

now we have a normallzed database.

3.2 Armstrong's Axioms (2 pts)

Let:

$$\begin{aligned} F = \{AB &\implies CD, \text{①} \\ B &\implies DE, \text{②} \\ C &\implies F, \text{③} \\ E &\implies G, \text{④} \\ A &\implies B\} \text{⑤} \end{aligned}$$

Show that F^+ contains $A \implies FG$. You may use Armstrong's Axioms plus the union and decomposition rules introduced in class.

If $B \to DE$, then $B \to D$ and $B \to E$

If $A \to B$, and $B \to E$, then $A \to E$

If $A \to E$, and $E \to G$, then $A \to G$

If $B \to DE$, then $AB \to ADE$

If $AB \to CD$, and $AB \to ADE$, then $AB \to ACDE$

If $AB \to ACDE$, then $B \to CDE$

If $B \to CDE$, then $B \to C$

If $B \to C$, and $C \to F$, then $B \to F$

If $A \to B$, $B \to F$, then $A \to F$

If $A \to G$, and $A \to F$, then $A \to FG$