Assignment 3

Hien Tu - tun1

October 26, 2021

Part One: The social media features

From the ER-diagram, there is the entity Subscriber with the primary keys username and number. Thus, we will have a table named Subscriber and we will make (username, number) to be the primary key pair. There are also attributes email, hash and salt in the entity Subscriber. We will use large strings VARCHAR(100) for username. For number, INT is used for the domain of attribute. Since both username and number are used as primary keys, two different subscribers can either have the same username or the same number but not both (exclusive or). We would use VARCHAR(320) to store email. Since hash and salt values require 64-byte each, we would use BINARY(64) for each of them. However, since the db2 server does not have BINARY(64), we need to use CHAR(64) instead. Since all the above attributes are required to create a subscriber, all of them need to be not null. The relational schema for Subscriber table is

Subscriber(username: VARCHAR(100),
number: INT,
email: VARCHAR(320),
hash: CHAR(64),
salt: CHAR(64)).

The ER-diagram has a many-to-many relationship Friend_Of between Subscriber and Subscriber. Thus, we will have a table name Friend_Of. The primary keys of the Subscriber will be the primary keys of the Friend_Of table. Since there are two subscribers, we will have a 4-tuple primary key (funame, funum, tuname, tunum). funame is short for from username, which is the username of the subscriber who is following and funum is short for from user number, which is the number of the subscriber who is following. Similarly, tuname and tunum is the username and the user number of the subscriber who is being followed, respectively. Thus, funame and tuname would reference username (column) of the Subscriber table while funum and tunum would reference number (column) of the Subscriber table. Since the domain of the attribute username in the Subscriber table is VARCHAR(100), we need to use VARCHAR(100) to be the domain of the attributes (keys) funame and tuname. Similarly, we need to use

INT for funum and tunum. All of the attributes would be not null as well. We also need to check that a subscriber is not following themselves, that is, we need to check either funame <> tuname or funum <> tunum or both are different (inclusive or). The relational schema for the Friend_Of table is

Friend_Of(funame: VARCHAR(100),
funum: INT,
tuname: VARCHAR(100),
tunum: INT).

We also have the entity Reaction, which will be the table Reaction in the relational schema. Notice that Reaction, ThreadR, and ReviewR has an ISA relationship, thus, we will use the ER method to create the relational schema for this entity-relationship model. Since Reaction has a strict-one-to-many relationship with Subscriber, we can add the primary keys of the table Subscriber to be the attributes of the table Reaction to represent this relationship without having another table. We name these attributes to be uname and unum, referencing the username and number in the table Subscriber with the domain of VARCHAR(100) and INT, respectively. The primary key for Reaction is id with INT as the type. We will automatically increment the id of the Reaction. Furthermore, Reaction has title and content as its attributes. Since the title would be shorter than the content of a reaction, we will use VARCHAR(100) for title and CLOB for content. The relational schema for Reaction is

Reaction(id: INT,
title: VARCHAR(100),
content: CLOB,
uname: VARCHAR(100),
unum: INT).

Notice that ThreadR has strict-one-to-many and ISA relationships with Reaction. As mentioned before, we use the ER method for the ISA relationship. Thus, we will have threadfrom as the primary key of the table ThreadR to reference the id of the current reaction. We also have an attribute threadto to reference the id of the reaction that the current reaction is reacting to. Since both threadfrom and threadto reference the id of the reactions, both have the type INT and must be not null. We only have threadfrom as the primary key since a reaction can only react to exactly one other reaction (the other reaction can be a reaction or review) while a reaction (or review) can be reacted by many reactions. To make sure a reaction is not reacting to itself, we need to check that threadfrom <> threadto (**CHECK**(threadfrom <> threadto)). The table ThreadR is

ThreadR(threadfrom: INT,
threadto: INT).

We also have the table for the entity ReviewR. We will have reactfrom as the primary key for the table ReviewR, referencing the id of the current reaction, whose domain of attribute is INT. Since ReviewR also has a strict-one-to-many relationship with Review entity, we will add the primary (and partial) keys of the entity Review as attributes of the table ReviewR. Thus, we will have reacteduname: VARCHAR(100), referencing the username of the subscriber whose review is being reacted to, reactedunum: INT, referencing the number of the subscriber whose review is being reacted to, reactedrevision: INT, referencing the revision number of the review. We will need to add some more attributes to the table ReviewR since Review is also a weak entity of the entity Film (in Part 2). We will have reactedftitle: VARCHAR(100), reactedfyear: INT, reactedfcreator: INT as attributes, referencing the title, year, creator of the table Film in Part 2. In other words, reacteduname, reactedunum, reactedftitle, reactedfyear, reactedfcreator, and reactedrevision reference the uname, unum, ftitle, fyear and fcreator in the table Review. These attributes will be explained more in the table Review and the table Film in Part 2. The table ReviewR is

ReviewR(reactfrom: INT,
reacteduname: VARCHAR(100),
reactedunum: INT,
reactedftitle: VARCHAR(100),
reactedfyear: INT,
reactedfcreator: INT,
reactedrevision: INT).

For the entity Review, we will have a table Review. Since Review is a weak entity of Subscriber, we will have the primary keys of the table Subscriber as part of the primary keys of the table Review. Hence, we will have uname: VARCHAR(100) and unum: INT for the table Review, referencing the username and number of the table Subscriber. We also have revision as the partial key of the table Review. Since it stores the version of the review, it has INT as the domain. We will also have attributes score of type INT and timestamp whose domain is TIMESTAMP. Notice that Review is also a weak entity of the table Film in Part 2. Thus, we will need to add primary keys of the table Film as part of the primary keys of the table Review. Thus, we will have ftitle: VARCHAR(100), fyear: INT, fcreator: INT as primary keys of the table Review, referencing the title, the year, and the creator of the table Film. The choice of the domain of attributes for Film will be explained in Part 2. Therefore, the primary key of the Review is (uname, unum, ftitle, fyear, fcreator, revision). All of the keys and attributes should be not null since they are necessary to create a review. Furthermore, we need to check that the score should be between 0

and 10 (**CHECK**(0 <= score AND score <= 10)). The table Review is

Review(uname: VARCHAR(100),
unum: INT,
ftitle: VARCHAR(100),
fyear: INT,
fcreator: INT,
revision: INT,
score: INT,
timestamp: TIMESTAMP).

Notice that VideoReview and TextReview have ISA relationships with Review, we will use the ER method to do represent these relationships. Thus, in the table VideoReview, we will have (uname, unum, ftitle, fyear, fcreator, revision) as the primary key, referencing the above keys of the Review table. In addition, we will have one more attribute, named video. Since videos need to be store as binary files and are often large, we will use BLOB for video attribute, this attribute should not be null as well. The table for VideoReview is

VideoReview(uname: VARCHAR(100),
unum: INT,
ftitle: VARCHAR(100),
fyear: INT,
fcreator: INT,
revision: INT,
video: BLOB).

Similarly, the table TextReview has (uname, unum, ftitle, fyear, fcreator, revision) as the primary key and one more attribute description with the domain as CLOB and the attribute should not be null. Since the description of the review of the film could be long, we use CLOB as the domain of attribute.

TextReview(uname: VARCHAR(100),
unum: INT,
ftitle: VARCHAR(100),
fyear: INT,
fcreator: INT,
revision: INT,
description: CLOB).

Part 2: The film information

The table Person has the primary key id. We will let the domain of id to be INT and let it be auto-generated. The name of the person should be stored in

VARCHAR(100) and we will store birthdate in DATE type. While id and name should be not null, birthdate can be null. The relational schema for Person is

Person(id: INT,
name: VARCHAR(100),
birthdate: DATE).

For the table Film, we have the title, year and creator as the primary keys. Since the title of the film is usually short, we will use VARCHAR(100). We will use INT to store the year. Since the creator references the id column in the table Person, we will use INT for creator. We will use DECIMAL to specify the domain of the attribute duration to demonstrate how many hours a film lasts and DECIMAL for the attribute budget since the budget can have part of a dollar (for example, $4,500,000.25). We also need to check that the year should be before or equal to 2021 (current year) (**CHECK** year <= 2021). Unless the cinema wants to store films whose expected release year is later than 2021, then we can skip this check. Since a film should have title, year of release, the creator, the duration and the budget, all the keys and attributes should not be null. The relational schema for Film is

Film(title: VARCHAR(100),
year: INT,
creator: INT,
duration: DECIMAL,
budget: DECIMAL).

The view Film_Info will have the columns title, year, duration, and budget from the table Film and the column creator_name from the column name in table Person. Thus, we will inner join the two tables Film and Person where the creator id is the same as the id in the Person table. The domains of attributes title, year, duration, and budget are similar to the ones in the Film table, which are VARCHAR(100), INT, DECIMAL, and DECIMAL, respectively. The domain of attribute for creator_name is the same as the attribute name in table Person, which is VARCHAR(100).

Film_Info(title: VARCHAR(100),
year: INT,
duration: DECIMAL,
budget: DECIMAL,
creator_name: VARCHAR(100)).

For the table Role, since a person can have multiple role and a film can have multiple people with their roles, we cannot use them as primary keys. Thus, we will have a primary key rid to represent the role number. It will have the type INT and is auto-generated. The table also have pid: INT, referencing

the id in the Person table; the ftitle: VARCHAR(100), fyear: INT, fcreator: INT, referencing the title, year, creator of the Film table, correspondingly. The table Role also has the attribute role to store the role of each person, we will use VARCHAR(100) since the role should not be too long. The attributes in the table Role should not be null as well since most of the attributes reference other not null attributes from other tables, while rid is auto-generated and every person who is in a film needs to have a role in the film. To create a constraint that all film creators have the role of director for that film, we will check, in the table Role, if the pid is equal to fcreator (that is, the id of the person in a role is the same as the id of the film creator), then that person is the creator of the film and the role should be 'director'. Mathematically represent the idea, pid = fcreator $\implies$ role = 'director'. Since SQL does not have $\implies$, and we know that $p \implies q \equiv \neg p \vee q$, we can present the constraint in SQL as **CHECK** (NOT (pid = fcreator) OR (role = 'director')). The relational schema of the table Role is

Role(rid: INT,
pid: INT,
ftitle: VARCHAR(100),
fyear: INT,
fcreator: INT,
role: VARCHAR(100)).

For the SQL part, since we need to have the tables in Part 2 for the tables Review, VideoReview, TextReview, Reaction, ThreadR, ReviewR to use, we need to create the tables in Part 2 first, then create those tables later. Thus, I decided to create the tables (and views) in Part 2 after creating tables Subscriber and Friend_Of, and create the tables Review, VideoReview, TextReview, Reaction, ThreadR, ReviewR after creating the tables (and views) in Part 2.