# CSC343 Assignment 1 Solution

June 15, 2020

## Warm Up

1. What does this integrity constraint mean? $\sigma_{follower=followed} Follows = \emptyset$

   Means there is no result in a query where user follow himself/herself

2. Would it be a good idea to define the Follows relation like this? Follows()<u>follower</u>, <u>followed</u>, start)

   Omitted for now

3. Can the database represent a single post that has multiple comments?

   Yes.

   Create two relations Comment(<u>pid, commenter, when</u>, text) and Post(<u>pid</u>, uid, when, location, caption). Set *pid* in comment to be the pid of Post.

   Since primary key of Comment is *pid, commenter, when*, comments can have multiple entires with the same *pid*.

   On the other hand, since *pid* in Post is the only primary key, it's value must be non-repeating.

4. Can the database represent multiple comments from the same user on one post?

   Yes.

   The PRIMARY KEY of Comment(<u>pid, commenter, when</u>, text) depends on <u>pid, commenter, when</u>.

   By definition of PRIMARY KEY, the entry with the same pid, commenter, and when

cannot exist.

Since pid and commenter are the same in this case, as long as when is different, the PRIMARY KEY condition will remain valid.

5. How does the schema allow *any* number of photos or videos to be inclyded in one story, but restrict the user to having only one profile photo?

The user is restricted to having one profile photo since the attribute *photo* contains only the URL of user photo.

On the other hand, a story is allowed to have multiple videos or photos, since its list of video and photo urls are defined in relation SIncludes, and it's PRIMARY KEY depends on two attributes *sid* and *url*.

So, as long as there are no entries in SIncludes with the repeating value of *url*, a story can have as many photos and video urls.

6. Can the database represent that a user likes the same post more than once? (If not, how would you change the schema to allow this?)

No.

To like the same post by one person more than once, the matching values of the attributes *liker* and *pid* need to appear more than once in Relation Likes.

Since, *pid* and *liker* are together set as PRIMARY KEY, this is not allowed.

7. Can the database represent that a user makes two posts at the same time?

Yes.

For a post to be created at the same time, the date-created attribute must not be set as a part of primary key.

And indeed it is so in Post relation.

8. Can the database represent that the same user makes the same comment on two different posts?

Yes.

The problem tells us the comment is made by the same user on two different posts, so the value of *pid* is different and *commenter* is the same.

Since the PRIMARY KEY of Comment relation tells us an entry cannot be created when *pid, commenter*, and *when* are the same, this doesn't create an issue.

9. Can the database represent that the same picture is included in 2 stories?

Yes.

The problem tells us the same picture is included in two different stories, so the value of *sid* are different and *url* is the same.

Since the PRIMARY KEY of SIncludes relation tells us an entry cannot be created when *sid*, and *url* are the same, this doesn't create an issue.

# Part 1

1. **likers_posts(uid, liker)** $\coloneqq \Pi_{liker,uid}(Post \bowtie Likes)$

   **dislikers_posts(uid, liker)** $\coloneqq \Pi_{liker,uid}(\rho_{u2.uid \to liker}(\sigma_{u1.uid \neq u2.uid}($ $\rho_{u1}(User) \times \rho_{u2}(User))) -$ likers_posts(uid, liker))

   **likers_posts_not_follow(uid, liker)** $\coloneqq \Pi_{liker,uid}(Post \bowtie Likes) -$ $\Pi_{liker,uid}(\rho_{followers \to liker, followed \to uid}(Follows))$

   **dislikers_posts_not_follow(uid, liker)** $\coloneqq$ dislikers_posts(liker, uid) $-$ $\Pi_{liker,uid}(\rho_{followers \to liker, followed \to uid}(Follows))$

   **viewers_stories(uid, viewerid)** $\coloneqq \Pi_{uid,viewerid}(Story \bowtie Saw)$

   **no_viewers_stories(uid, viewerid)** $\coloneqq \Pi_{uid,viewerid}(\rho_{u2.uid \to viewerid}(\sigma_{u1.uid \neq u2.uid}($ $\rho_{u1}(User) \times \rho_{u2}(User))) -$ viewers_stories(uid, viewerid))

   **no_viewers_stories_not_follow(uid, viewerid)** $\coloneqq$ no_viewers_stories(uid, viewerid) $- \Pi_{liker,uid}(\rho_{followers \to liker, followed \to uid}(Follows))$

**dislikers_no_viewers_not_follow(uid, viewerid)** :=
no_viewers_stories_not_follow(uid, viewerid) ∪
$\rho_{liker \to viewerid}$(dislikers_posts_not_follow(uid, liker))

**Answer1(uid, about)** := $\Pi_{uid,about}(User \bowtie \rho_{viewerid \to uid}($
$\Pi_{viewerid}$(dislikers_no_viewers_not_follow(uid, viewerid))))

**Answer2(username, description)** :=
$\rho_{uid \to username, about \to description}(Answer1(uid, about))$

**Rough Work:**

- Find all users who liked a post of a user

  **likers_posts(uid, liker)** := $\Pi_{liker,uid}(Post \bowtie Likes)$

- Find all users who never liked a post of a user

  **dislikers_posts(uid, liker)** := $\Pi_{liker,uid}(\rho_{u2.uid \to liker}(\sigma_{u1.uid \neq u2.uid}($
  $\rho_{u1}(User) \times \rho_{u2}(User))) -$ likers_posts(uid, liker))

- Find all users who liked a post of a user they do not follow

  **likers_posts_not_follow(uid, liker)** := $\Pi_{liker,uid}(Post \bowtie Likes) -$
  $\Pi_{liker,uid}(\rho_{followers \to liker, followed \to uid}(Follows))$

- Find all users who never liked a post of a user they do not follow

  **dislikers_posts_not_follow(uid, liker)** := dislikers_posts(liker, uid) −
  $\Pi_{liker,uid}(\rho_{followers \to liker, followed \to uid}(Follows))$

- Find all users who viewed a story of a user

  **viewers_stories(uid, viewerid)** := $\Pi_{uid,viewerid}(Story \bowtie Saw)$

- Find all users who never viewed a story of a user

  **no_viewers_stories(uid, viewerid)** := $\Pi_{uid,viewerid}(\rho_{u2.uid \to viewerid}(\sigma_{u1.uid \neq u2.uid}($
  $\rho_{u1}(User) \times \rho_{u2}(User))) -$ viewers_stories(uid, viewerid))

- Find all users who never viewed a story of a user they do not follow

**no_viewers_stories_not_follow(uid, viewerid)** :=
no_viewers_stories(uid, viewerid) − $\Pi_{liker,uid}(\rho_{followers\to liker,followed\to uid}(Follows))$

- Find all users who never liked or viewed a post or a story of a user they do not follow

**dislikers_no_viewers_not_follow(uid, viewerid)** :=
no_viewers_stories_not_follow(uid, viewerid) ∪
$\rho_{liker\to viewerid}$(dislikers_posts_not_follow(uid, liker))

- Report their user id and "about" information

**Answer1(uid, about)** := $\Pi_{uid,about}(User \bowtie \rho_{viewerid\to uid}($
$\Pi_{viewerid}$(dislikers_no_viewers_not_follow(uid, viewerid))))

- Put the information into a relation with attributes "username" and "description"

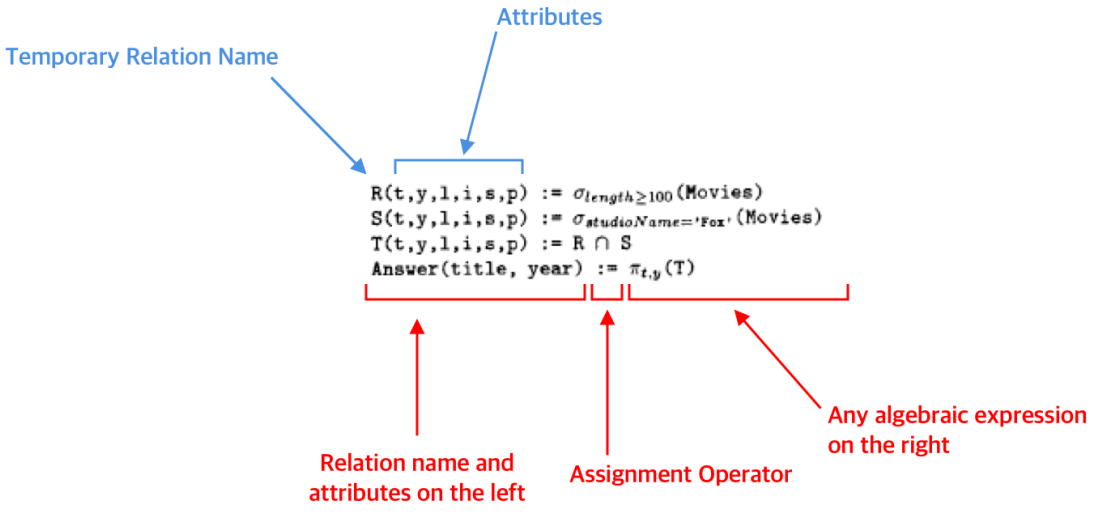
**Answer2(username, description)** := $\rho_{uid\to username,about\to description}(Answer1(uid, about))$

**Notes:**

s

- Renaming
  - **Syntax (Attribute renaming):** $\rho_{\text{oldName}\to\text{newName}}$
  - e.g.

    $\rho_{\text{Father}\to\text{Parent}}$(Paternity)
- Linear Notation for Algebraic Expressions
  - Invents names for the temporary relations
  - Works like variable in mathematical expression
  - The name **Answer** is used for the result of the final step.



- e.g. Report the user name of every student who has never worked with anyone, but has indeed submitted at least one file for at least one assignment

$$had\_groupmates(username) \coloneqq \Pi_{M1.username}(\sigma_{M1.gID=M2.gID \wedge M1.username \neq M2.username}(\rho_{M1}Membership \times \rho_{M2}Membership))$$

$$no\_groupmates(username) \coloneqq \Pi_{username}(USER - had\_groupmates)$$

$$submitted\_some\_files(username) \coloneqq \Pi_{username}(Submission)$$

$$\textbf{Answer(username)} \coloneqq no\_groupmates \cap submitted\_some\_files$$

2. **Notes:**

- Simple example

a) **Hashtag**

| pid | tag |
|---|---|
| 1 | Hello |
| 2 | Hi |
| 3 | Hello |