# Assignment 3

Hien Tu - tun1

October 25, 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 CLOB 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. Since all the above attributes are required to create a subscriber, all of them need to be not null (except for number since it is already automatically incremented). The relational schema for Subscriber table is

**Subscriber**(username: CLOB,
number: INT,
email: VARCHAR(320),
hash: BINARY(64),
salt: BINARY(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. 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 CLOB, we need to use CLOB 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: CLOB,
funum: INT,
tuname: CLOB,
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 reference to the username and number in the table Subscriber with the domain of CLOB 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: CLOB,
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. 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 a 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: CLOB, 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, reactedfrevision: INT as attributes, referencing the title, year, creator of the table Film in Part 2. These will be explained more in Part 2. The table ReviewR is

**ReviewR**(reactfrom: INT,
reacteduname: CLOB,
reactedunum: INT,
reactedrevision: INT,
reactedftitle: VARCHAR(100),
reactedfyear: INT,
reactedfcreator: 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: CLOB 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 store 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 will be explained in Part 2. Therefore, the primary key of the Review is (uname, unum, revision, ftitle, fyear, fcreator, revision). All of the keys and attributes should be not null since they are necessary to create a review. The table Review is

**Review**(uname: CLOB,
unum: INT,
revision: INT,
ftitle: VARCHAR(100),
fyear: INT,
fcreator: 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, revision, ftitle, fyear, fcreator, revision) as the primary key with the explanation as the above. 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: CLOB,
unum: INT,
revision: INT,
ftitle: VARCHAR(100),
fyear: INT,
fcreator: INT,
video: BLOB).

Similarly, the table TextReview has (uname, unum, revision, 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.

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