University of Saskatchewan
CMPT 355: Theory and Application of Databases

ASSIGNMENT 3

Introduction

The purpose of this assignment is to gain experience creating conceptual database designs, and translating those designs to physical database implementations. The conceptual design could be informed by written requirements, mock-ups of the user interface, or if they have an old system that is being replaced, their old data and reports from their old system.

Scenario

You were hired to design a database for a new Saskachewan-based competitor to Reddit called SaskChat. Reddit is an online discussion forum[1] where users can post and discuss content. Your employer has big plans for SaskChat and wants it to stand out. Therefore they do not want to entirely copy Reddit's format and design, but instead want to start from scratch with a new design.

You were not their first choice to implement the database, but the person they hired prior to you only managed to implement a small part of the database. Additionally, they scraped reddit for Saskatchewan-related discussions and compiled a `.csv` file containing that data. They want to import this into their new database to make their platform look like it has people using it already.

Provided Materials

You are provided with the following materials:

1. A written description of the system.
2. A mock-up showing the desired look and feel of the system.
3. A partially-completed database.
4. A `.csv` file containing data that they want to migrate to the new database — the "data dump".

[1] `https://en.wikipedia.org/wiki/Internet_forum`

1. Written Description

The discussion forum that your employer envisions is made up primarily of user-provided posts grouped into threads. Each thread contains a hierarchical organization of posts, as users can create new posts that either reference no other post or a previously existing post. Each post belongs to exactly one user, but if that user is deleted (such as if they are a troll), their posts should remain. Each post belongs to exactly one thread; if that thread is deleted, then so too should the posts belonging to that thread be deleted. Trolls can also simply be banned, meaning that they are no longer able to make new posts.

Every thread should have exactly one category, with one exception — administrators and moderators should be able to create threads that are forum-wide announcements and do not belong to any category. Administrators and moderators should also be able to "pin" threads so that they always show up at the top of the thread listings for any particular category. Categories should have unique names as well as a bit of descriptive text.

Threads can be "tagged". Unlike categories, where threads could only have one category, threads can have as many tags as desired or none at all. Tags should be defined separately from thread creation and can be re-used across multiple threads. Only administrators should be allowed to define new tags. Tags should have unique names.

Every post can be rated by every user. After each post has been rated, the combined rating should be re-calculated and stored alongside the post's information. Ratings can be positive (1) or negative (-1). A user can only rate a post one time. If a user mistakenly rates a post, they can re-rate it, or they can clear it, resulting in a rating of 0.

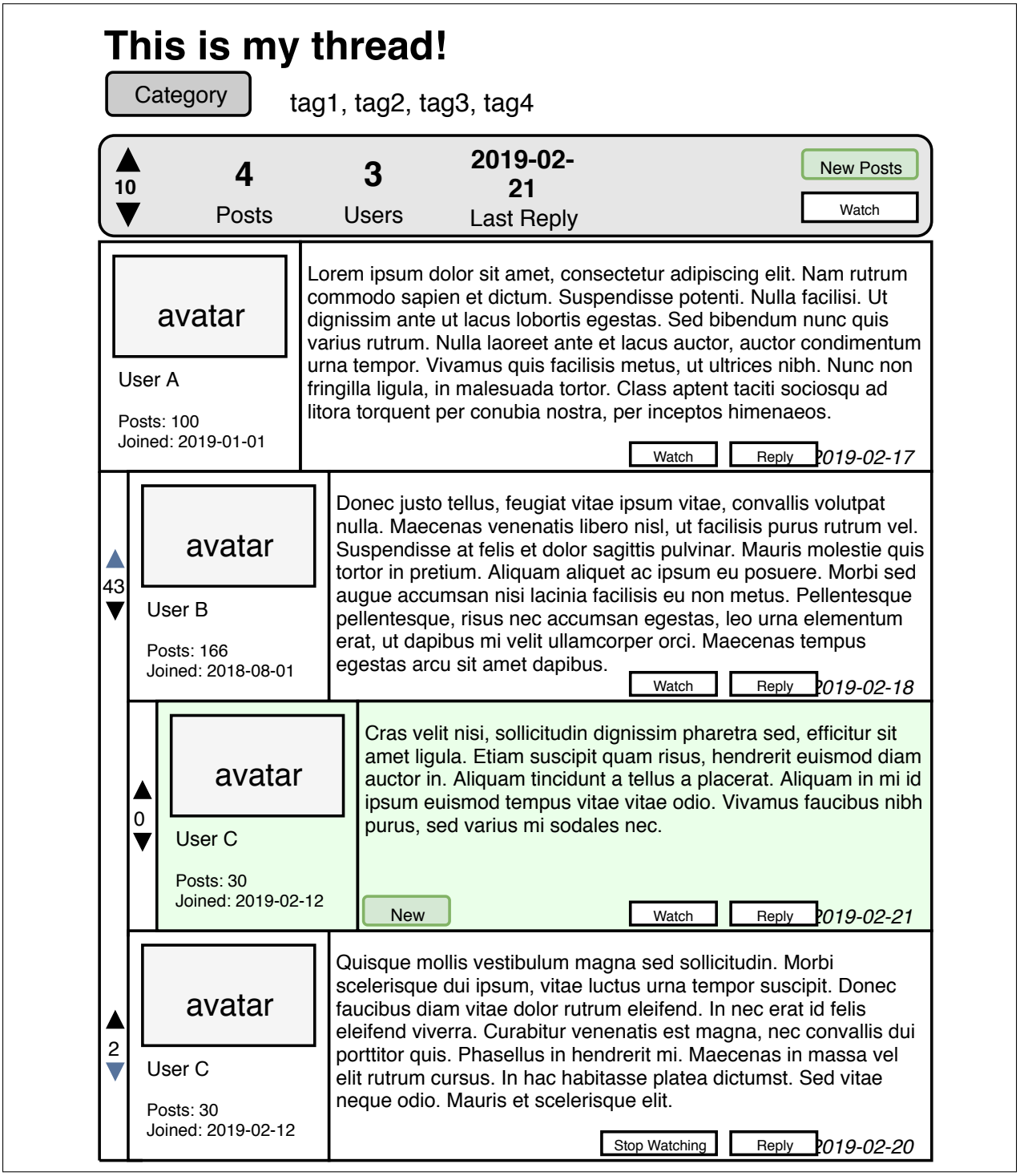
Users should be able to optionally "watch" both threads and posts. When a thread is watched, then any time a new post is added to that thread an email should be sent to the user notifying them of the new post. Watching a post is similar, except that only postings that are direct replies of the watched post should result in email notifications. In both cases, it should be possible for the user to view a listing of new postings relative to the time when they first started to watch the post or thread.

To keep track of whether or not there have been new replies to a thread since a user last visited, a log of the user's activity should be created. The log should keep track of the user, the thread, and the time that they last viewed it on. If the user views the thread that they have visited previously and there are new posts, those posts should be highlighted in green.

If any thread or post is deleted, then all of the ratings, activity, and watch statuses for that thread or post should also get deleted. Additionally, Categories cannot be deleted if they contain any threads at all — all of the threads in that category should be migrated to another category before deletion.

2. Mock-up

Their mock-up shows what a user might see when they are viewing a thread in the online discussion forum.



3. Partial Database

The database currently consists of three tables: User, Thread, and Post

```sql
-- User table
CREATE TABLE User (
    UserID          INTEGER  PRIMARY KEY NOT NULL,
    DisplayName     TEXT     NOT NULL,
    Email           TEXT     NOT NULL UNIQUE,
    Password        TEXT     NOT NULL DEFAULT (''),
    IsBanned        BOOLEAN  NOT NULL DEFAULT (0) CHECK (IsBanned IN (0, 1)),
    IsModerator     BOOLEAN  NOT NULL DEFAULT (0) CHECK (IsModerator IN (0, 1)),
    IsAdministrator BOOLEAN  NOT NULL DEFAULT (0) CHECK (IsModerator IN (0, 1)),
    RegisteredOn    DATETIME NOT NULL DEFAULT (datetime(datetime('now', 'localtime'),
     ↪  'utc')),
    PostCount       INTEGER  NOT NULL DEFAULT (0)
);

-- Thread table
CREATE TABLE Thread (
    ThreadID       INTEGER  PRIMARY KEY NOT NULL,
    ThreadName     TEXT     NOT NULL,
    CategoryID     INTEGER  REFERENCES Category (CategoryID) ON DELETE RESTRICT,
    IsPinned       INTEGER  NOT NULL DEFAULT (0) CHECK (IsPinned IN (0, 1)) ,
    IsAnnouncement INTEGER  NOT NULL DEFAULT (0) CHECK (IsAnnouncement IN (0, 1)),
    CreatedOn      DATETIME NOT NULL
                   DEFAULT (datetime(datetime('now', 'localtime'), 'utc')),
    PostCount      INTEGER  NOT NULL DEFAULT (0),
    CHECK (CategoryID IS NOT NULL OR IsAnnouncement = 1)
);

-- Post table
CREATE TABLE Post (
    PostID       INTEGER  PRIMARY KEY NOT NULL,
    PostContent  TEXT     NOT NULL,
    ThreadID     INTEGER  NOT NULL REFERENCES Thread (ThreadID) ON DELETE CASCADE,
    Rating       INTEGER  NOT NULL DEFAULT (0),
    CreatedOn    DATETIME NOT NULL DEFAULT (datetime(datetime('now', 'localtime'),
     ↪  'utc')),
    ModifiedOn   DATETIME NOT NULL DEFAULT (datetime(datetime('now', 'localtime'),
     ↪  'utc')),
    ParentPostID INTEGER  REFERENCES Post (PostID),
    AuthorUserID INTEGER  REFERENCES User (UserID) ON DELETE SET NULL
);
```

See the `'Start.db'` file.

4. Data to Import

See the `'Dump.csv'` file.

Part 1: The Conceptual Design (30 points)

For this part of the assignment, you will need to reference the written description and the mock-up of the system. The columns listed in the data dump (data to import) will have names based on the previous system that may be somewhat different than what you find based on analyzing their requirements and their mock-up. The entities, attributes, and primary key for the three implemented tables are already provided for you.

1. (10 points) Based on the written description of the system and the mock-up, identify all of the entities, attributes, and primary keys you need to make up the database. If there is no natural primary key for an entity, then make a surrogate key named like `EntityID`.

   Provide your answer in the form of a list of entities, with each entity in the form of:

   - Entity (PrimaryKey, Attribute1, Attribute2, ...)

   Note: You do not need to include the provided Thread, Post, and User entities in your solution, but you may prefer to do so for later reference when creating the database.

   Starting Point

   The partially implemented database includes the following entities, attributes, and primary keys.

   - Thread (ThreadID, ThreadName, IsPinned, IsAnnouncement, CreatedOn, PostCount)
   - Post (PostID, CreatedOn, ModifiedOn, Rating)
   - User (UserID, DisplayName, Email, Password, IsBanned, IsModerator, IsAdministrator, RegisteredOn, PostCount)

2. (20 points) Based on the written description and the mock-up, construct a conceptual ER diagram using Crow's Foot notation that demonstrates the relationships between the entities, as well as the minimum and maximum cardinalities.

   Notes: Since you will have listed the attributes and primary keys in question 1, you do not need to include them in your diagram, although you are welcome to, and you may find it helpful when it is time to implement the database. You may also label the relationships if you feel it is helpful (e.g., the relationship between Post and itself might be labelled "parent of" or "child of"), but this is not a requirement.

   Your diagram must...

   - include the relationships between entities.
   - indicate maximum cardinalities.
   - indicate minimum cardinalities.

   If you think that any of these are not clear from the provided materials, then state an assumption in words like in the example below.

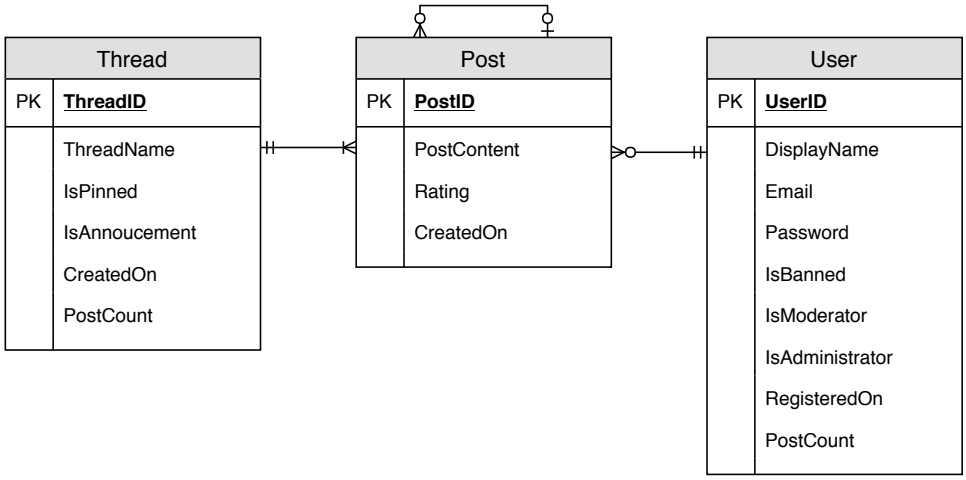
   Example Assumptions

   Considering the relation between a car and its wheels, here are some possible assumptions:

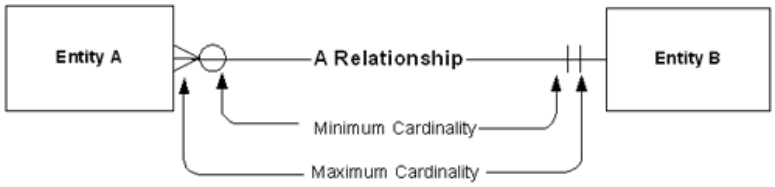
   - Cars have at least one wheel (regarding minimum cardinality).
   - Tires do not have multiple cars; they have only one car (regarding maximum cardinality).

Starting ER Diagram

Since the database is already partially implemented, here is an ER diagram you can use as a starting point.



Crow's Foot Notation



Part 2: Normalizing the Provided Data (25 points)

post_id	topic_id	topic_date	post_date	topic	post	category_id	category_name	user_id	username	email	user_id_rating	rating
19	11	2018-03-16 01:13	2018-03-16 01:13	Fish nothing...	Own phone...	10	Uncategorized	5	katrina62	katrina62@example.com	15	1
19	11	2018-03-16 01:13	2018-03-16 01:13	Fish nothing...	Own phone...	10	Uncategorized	5	katrina62	katrina62@example.com	20	1
19	11	2018-03-16 01:13	2018-03-16 01:13	Fish nothing...	Own phone...	10	Uncategorized	5	katrina62	katrina62@example.com	44	1
20	11	2018-03-16 01:13	2018-03-16 07:25	Fish nothing...	Door world...	10	Uncategorized	5	katrina62	katrina62@example.com	49	1
21	12	2018-03-16 05:09	2018-03-16 05:09	Design dream...	Lead same...	5	Real Estate	5	katrina62	katrina62@example.com	38	1
22	13	2018-03-16 06:29	2018-03-16 06:29	Believe remember...	A major...	7	Movies	3	lisa.beer	lisa.beer@example.com	21	1
22	13	2018-03-16 06:29	2018-03-16 06:29	Believe remember...	A major...	7	Movies	3	lisa.beer	lisa.beer@example.com	28	1
22	13	2018-03-16 06:29	2018-03-16 06:29	Believe remember...	A major...	7	Movies	3	lisa.beer	lisa.beer@example.com	38	1
23	13	2018-03-16 06:29	2018-03-16 11:26	Believe remember...	List different...	7	Movies	2	gayle45	gayle45@example.com	11	1
23	13	2018-03-16 06:29	2018-03-16 11:26	Believe remember...	List different...	7	Movies	2	gayle45	gayle45@example.com	18	1

One of the materials supplied to you was a data dump from an existing discussion forum. You will eventually need to import this data into your new database, but for now your job is to normalize it to reveal relationships between tables as well as tables that may not have been evident from the written description. The provided data has been exported in first normal form (1NF), as this is was a convenient method of extracting data from their old database.

Above is a sample of the data — see the provided `Dump.csv` file for the complete data.

Note: You can use the result of this to double-check part of your work in Part 1 and vice versa. This normalization process may also illuminate some implementation details for you.

3. (5 points) The primary key. What column(s) make up the primary key in the table shown?
4. (10 points) Identifying partial functional dependencies (transforming to 2NF).
   (a) (5 points) Next, you need to identify the non-key columns that are partially functionally dependant on the primary key. For each non-key column, identify what primary key column(s) it directly depends on, using the form:

   $$\text{PrimaryKeyColumn1, PrimaryKeyColumn2, ...} \rightarrow \text{NonKeyColumn1}$$

   (b) (5 points) Based on the distinct primary key column combinations, construct new tables in addition to the original table, moving the non-key columns to the new tables. Describe all of the tables (not just the new ones) in the form:

   $$\text{Table (}\underline{\text{PrimaryKeyColumn1, ...}}\text{, NonKeyColumn1, ...)}$$

5. (10 points) Identifying transitive dependencies (transforming to 3NF).
   (a) (5 points) Next, you need to identify any transitive dependencies that still exist after arriving at the second normal form (2NF). This essentially involves looking for columns that could act as foreign keys, and describe some of the other columns inside the table. For each non-key column, identify any transitive dependencies in the form:

PrimaryKeyColumn1, PrimaryKeyColumn2, ... → NonPKColumn1 → NonKeyColumn1

(b) (5 points) Based on this, the distinct columns in between the two →'s become primary keys for new tables. Take these new primary keys and construct those new tables. Move over the identified non-key columns into those tables, and leave the new primary keys in the original table so that they can serve as foreign keys. Once again describe all of the tables (not just the new ones) in the form:

Table (PrimaryKeyColumn1, ..., NonKeyColumn1, ...)

Part 3: Implementing the Database (20 Points)

Based on the ER diagram you completed in Part 1, you must now implement the database.

6. (20 points) Provide the DDL `CREATE TABLE` syntax to implement each table in your database.

   Notes:

   - Most of the required domain constraints (`CHECK` constraints) were in the provided tables. There is only one additional `CHECK` constraint that you need.
   - Not all of the cardinality constraints listed in your ER diagram are possible to define when you are creating your tables. Implement as many as possible.
   - For data types, choose one of the following: `INTEGER`, `TEXT`, `DATETIME`, `NUMERIC`.
   - Keep an eye out for junction tables that need to be added.

Part 4: Importing Data (10 Points)

Now that you have your database implemented, let's import the provided data. This will help you later on as you can use it to test the triggers you make in the next part.

Note: The assignment has been designed in a way that means you can move forward with Parts 4 and 5 even if your design doesn't entirely match the "solution". There is only one new table involved and it shows up in both Part 1 and 2. Also, if you haven't done so already, run your `CREATE TABLE` queries inside of the provided database so that you have all of the required tables created.

7. (10 points) Import the `.CSV` file as a temporary table named "dump". Then, write a sequence of `INSERT INTO ... SELECT` queries to import the data dump into your database. Use a transaction so that all of the data is imported or none at all is. The order of the `INSERT` queries may be important.

   Note: `DISTINCT` will be your friend here. I recommend writing the `SELECT` queries first. When you import, pay attention to the default values of the provided tables as well as their constraints.

Part 5: Triggers (15 Points)

There are several columns in the database that should be updated or re-calculated based on actions that users make.

Two keep track of the total number of posts — one for the user's posts (User → PostCount) and another for the thread's posts (Thread → PostCount). A third keeps track of combined rating of each post (Post → Rating). A final column keeps track of when posts were edited (Post → ModifiedOn).

8. (5 points) Create a trigger that sets a Post's `ModifiedOn` date and time whenever that post gets updated. You can test your trigger by editing a post. You may want to look at the default value for that column for inspiration.
9. (5 points) Create triggers that set the User's `PostCount` and the Thread's `PostCount` every time post is inserted or deleted. You can test your trigger by inserting or deleting posts.
10. (5 points) Create triggers that set a Post's Rating whenever a rating for that post is inserted, deleted, or updated. You can test your trigger by inserting, deleting, or updating ratings.

To Hand In

- Written responses to Questions 1, 3, 4, and 5.
- The image of your ER diagram for Question 2.
- All of the SQL queries that you wrote for Questions 6, 7, 8, 9, and 10.

This is ideally done in a single `.pdf` file.