Michael Nguyen

8/4/2023

MET CS669SC1 REG

Professor Matthews

Track Social Media Connections

I would like to create a database that would 'Track Social Media Connections'. Creating real-time application of storage of user's profiles, posts, comments, likes, and connections between users. It also has the option to export it into excel to track between real life friends and the ones in the social media platform.

1. The person visiting this website or app store will installs the application.
2. The application will help create the users an account.
3. The users enter their information, and the account is created within the database.
4. The application will install the installation of the app.

From the database perspective, the use case requires storing information of users' data, e.g., step 2 and step 3. Step 1 and 4 gives the application the current information when it is linked to the social media platform.

Field	**What it Stores**	**Why it's Needed**
Users_Information	Summary of the User's information.	Selection of accounts if users has more than one account.
First_Name	First name of the account holder.	Person's first name when addressing them.
Last_Name	Last name of the account holder.	Peron's last name when addressing them.
Customer_Account_Creation	The date the account was created.	To determine when the account was created for retention ratio if necessary.

Another useful information would be to automatically update connections via background.

1. Update connections.
2. Monitor in real-time what connections is real-life friends and which is within the social media platform.
3. Every end-user has a unique excel file.

Field	**What it Stores**	**Why it's Needed**
Connection_Date	This is the date when the connection was established.	This is useful so the person knows what date they met a person on social media.
Connection_Location	This is the location of connection.	This is useful to know where the location is.
Meta_Data	Summarize basic information of end-user	This is useful for tracking and working with specific data within the application.

The database would make use of the field presented above and automatically store user's data within the application. This use case may help us understand more in-dept how communication work within the platform of social media. Moreover, it will export an excel file to store the user's social media information as a backup for the end-user to keep track when making connections with another person. In addition, and if the person wants to remain in contact or connected with the that person or not. In summary, this iteration will create a database design excel file to help keep track of user's connections within the social media platform and update it.

1. User Entity:

- Each user must have a unique user ID.
- Usernames must be unique.
- Usernames must meet a minimum length requirement.
- User passwords should be stored securely.
- User profiles can have optional fields such as name, bio, location, and profile picture.

2. Post Entity:

- Each post must have a unique post ID.
- Posts must be associated with a user who created them.
- Posts should have a timestamp indicating the date and time of creation.
- Posts can contain text, images, videos, or a combination.
- Posts can have optional fields such as location, tags, and privacy settings.

3. Friendship Relationship:

- Users can have a friendship relationship with other users.
- A friendship is a bidirectional relationship between two users.
- Friendship relationships can have attributes like date of connection and friendship status (pending, accepted, blocked).

4. Like Relationship:

- Users can like posts created by other users.
- A like relationship is a unidirectional relationship from a user to a post.
- Each user can like a post only once.
- Likes can be stored with additional metadata like timestamp and user's reaction (e.g., thumbs up, heart).

5. Comment Entity:

- Each comment must have a unique comment ID.
- Comments must be associated with a user who created them.
- Comments must be associated with the post they are commenting on.
- Comments should have a timestamp indicating the date and time of creation.
- Comments can contain text, images, or a combination.

6. Follow Relationship:

- Users can follow other users.
- A follow relationship is a unidirectional relationship from one user to another.
- Each user can follow another user only once.
- Follow relationships can have attributes like date of connection and notification settings.

7. Group Entity:

- Each group must have a unique group ID.
- Groups must have an owner who created them.
- Groups can have optional fields such as name, description, and privacy settings.
- Group members can have different roles (e.g., admin, moderator, member).

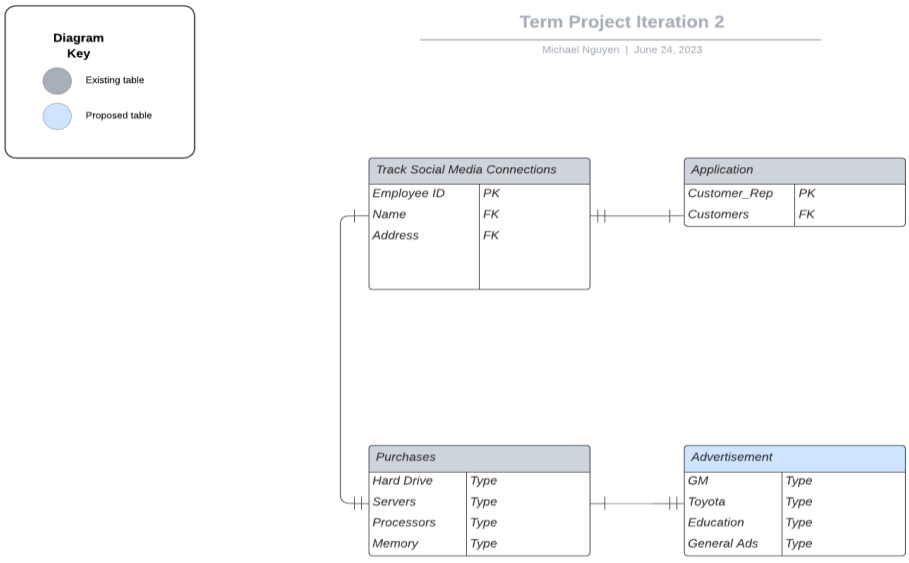
8. Membership Relationship:

- Users can be members of groups.
- A membership relationship is a bidirectional relationship between a user and a group.
- Each user can be a member of a group only once.
- Membership relationships can have attributes like date of joining and membership status.

9. Hashtag Entity:

- Each hashtag must have a unique hashtag ID.
- Hashtags can be associated with multiple posts.
- Hashtags must meet a minimum length requirement.
- Hashtags can be used in posts and comments to categorize content.

These rules provide a foundation for designing the structural database of a social media platform. Depending on the specific requirements and features of your social media platform, additional rules and constraints may be necessary. As I went through creating the structural database rules the use cases model it first needs to focus on what the database must store and what the data must be durable of keeping.



My database is a windows application named ‘Track Social Media Connections’ which records real life friends and acquaintances from the rest of online friends. When a person becomes a friend, they are updated in real-time the person’s name and the majority of the person’s information which export in an excel file to store the user’s social media information as backup. End-users has the option to keep track of these connections with another person to promote to real-life friends or not.

To look over my existing use cases, after some careful thought and considerations, the first peeks of my interest for this purpose would be.

Account Signup/Installations Use Case

5. The person visiting this website or app store will installs the application.
6. The application will help create the users an account.
7. The users enter their information, and the account is created within the database.
8. The application will install the installation of the app.

Something that would be useful to ‘Track Social Media Connections’ would be the update of connections via background. Perhaps it would be useful to offer a free account that has some limitations on the functionality, and a paid account which offers all of the features. I then modify the use case as follows:

Account Signup/Installation Use Case (New)

1. The person visiting this website or app store will installs the application.
2. The application asks them to create either a free or paid account when it first run.

3. The users selects the type of account and enters their information and the account is created in the database.
4. The application is then downloaded and installed in the computer so that their friends' connections may be updated in real-time.

The #2 now mentions free and paid account. I derive a fourth structural database rules to suppose the change to the use case as follows.

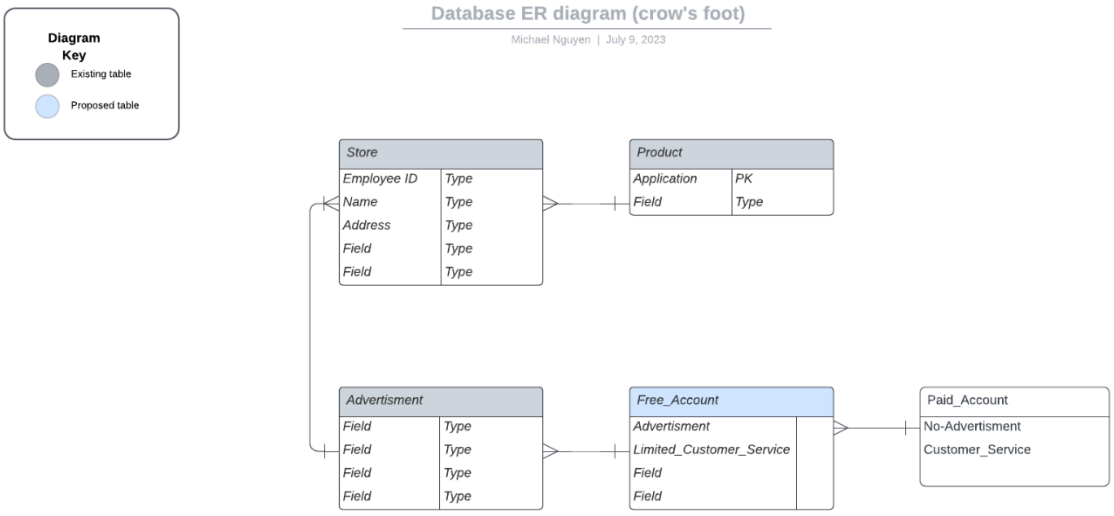
- An account is either a free account or a paid account.

This specialization-generalization rule turned out to be short, however does capture the intention. My database only has two kinds of accounts – free and paid – and that is the complete list. The relationship is totally complete. The account must be either free or paid, so the relationship is disjoint. I did not put any of the wording such as "several of these" or "none of these" since the rule is totally complete and disjoint.

As I was thinking about specialization-generalization, it became clear to me that I want to treat different kinds of connections differently (real-life friends and acquaintances). I will likely have different data to store for these different connections and different kinds of data for paid subscriptions versus the non-subscription as follows.

Automatic Real-Life Friends Track Social Media Connection Use Case

1. The real-life friends and acquaintances of subscriber versus non-subscribers.
2. The Track Social Media Connections detects real-life friends versus acquaintances and records the relevant information in the database such as calls and video chats.




```
-- Create User table
CREATE TABLE User (
    user_id INT PRIMARY KEY,
```

```
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(100) NOT NULL,
    salt VARCHAR(50) NOT NULL,
    name VARCHAR(100),
    bio VARCHAR(500),
    location VARCHAR(100),
    profile_picture VARCHAR(200)
);

-- Create Post table
CREATE TABLE Post (
    post_id INT PRIMARY KEY,
    user_id INT NOT NULL,
    content VARCHAR(500),
    media VARCHAR(200),
    timestamp TIMESTAMP NOT NULL,
    location VARCHAR(100),
    tags VARCHAR(200),
    privacy_setting VARCHAR(50),
    FOREIGN KEY (user_id) REFERENCES User(user_id)
);

-- Create Friendship table
CREATE TABLE Friendship (
    friendship_id INT PRIMARY KEY,
    user_id1 INT NOT NULL,
    user_id2 INT NOT NULL,
    connection_date DATE,
    friendship_status VARCHAR(50),
    FOREIGN KEY (user_id1) REFERENCES User(user_id),
    FOREIGN KEY (user_id2) REFERENCES User(user_id)
);

-- Create Like table
CREATE TABLE Like (
    like_id INT PRIMARY KEY,
    user_id INT NOT NULL,
    post_id INT NOT NULL,
    timestamp TIMESTAMP NOT NULL,
    reaction VARCHAR(50),
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (post_id) REFERENCES Post(post_id)
);

-- Create Comment table
CREATE TABLE Comment (
    comment_id INT PRIMARY KEY,
    user_id INT NOT NULL,
    post_id INT NOT NULL,
    content VARCHAR(500),
    media VARCHAR(200),
    timestamp TIMESTAMP NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (post_id) REFERENCES Post(post_id)
);

-- Create Follow table
CREATE TABLE Follow (
```

```
    follow_id INT PRIMARY KEY,
    follower_id INT NOT NULL,
    following_id INT NOT NULL,
    connection_date DATE,
    notification_settings BOOLEAN,
    FOREIGN KEY (follower_id) REFERENCES User(user_id),
    FOREIGN KEY (following_id) REFERENCES User(user_id)
);

-- Create Group table
CREATE TABLE Group (
    group_id INT PRIMARY KEY,
    owner_id INT NOT NULL,
    name VARCHAR(100),
    description VARCHAR(500),
    privacy_setting VARCHAR(50),
    FOREIGN KEY (owner_id) REFERENCES User(user_id)
);

-- Create Membership table
CREATE TABLE Membership (
    membership_id INT PRIMARY KEY,
    user_id INT NOT NULL,
    group_id INT NOT NULL,
    role VARCHAR(50),
    join_date DATE,
    membership_status VARCHAR(50),
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (group_id) REFERENCES Group(group_id)
);

-- Create Hashtag table
CREATE TABLE Hashtag (
    hashtag_id INT PRIMARY KEY,
    hashtag_text VARCHAR(100)
);


-- Index for User table
CREATE INDEX idx_user_username ON User (username);
-- This index will improve the performance of lookups based on the 'username' column.

-- Index for Post table
CREATE INDEX idx_post_user_id ON Post (user_id);
-- This index will improve the performance of queries joining 'Post' with 'User' on
'user_id'.

CREATE INDEX idx_post_timestamp ON Post (timestamp);
-- This index will improve the performance of queries that filter or order by
'timestamp'.

CREATE INDEX idx_post_tags ON Post (tags);
-- This index will improve the performance of queries that filter based on hashtags in
'tags' column.

-- Index for Friendship table
CREATE INDEX idx_friendship_user_ids ON Friendship (user_id1, user_id2);
-- This composite index will improve the performance of queries related to friendships.
```

```sql
-- Index for Like table
CREATE INDEX idx_like_user_id ON Like (user_id);
-- This index will improve the performance of queries joining 'Like' with 'User' on
'user_id'.

CREATE INDEX idx_like_post_id ON Like (post_id);
-- This index will improve the performance of queries joining 'Like' with 'Post' on
'post_id'.

-- Index for Comment table
CREATE INDEX idx_comment_user_id ON Comment (user_id);
-- This index will improve the performance of queries joining 'Comment' with 'User' on
'user_id'.

CREATE INDEX idx_comment_post_id ON Comment (post_id);
-- This index will improve the performance of queries joining 'Comment' with 'Post' on
'post_id'.

-- Index for Follow table
CREATE INDEX idx_follow_follower_id ON Follow (follower_id);
-- This index will improve the performance of queries joining 'Follow' with 'User' on
'follower_id'.

CREATE INDEX idx_follow_following_id ON Follow (following_id);
-- This index will improve the performance of queries joining 'Follow' with 'User' on
'following_id'.

-- Index for Membership table
CREATE INDEX idx_membership_user_id ON Membership (user_id);
-- This index will improve the performance of queries joining 'Membership' with 'User' on
'user_id'.

CREATE INDEX idx_membership_group_id ON Membership (group_id);
-- This index will improve the performance of queries joining 'Membership' with 'Group'
on 'group_id'.

-- Index for Hashtag table
CREATE INDEX idx_hashtag_hashtag_text ON Hashtag (hashtag_text);
-- This index will improve the performance of queries based on 'hashtag_text' column.
```

Here is the SQL script for the next iteration of project #5.

To track social media connections and distinguish between free and paid accounts, we can create an SQL script with the necessary tables and relationships.

```sql
-- Create Account Type table
CREATE TABLE AccountType (
    account_type_id INT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL
);

-- Insert default account types (Free and Paid)
INSERT INTO AccountType (account_type_id, type_name) VALUES
(1, 'Free'),
(2, 'Paid');
```

```sql
-- Create User table
CREATE TABLE User (
    user_id INT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(100) NOT NULL,
    salt VARCHAR(50) NOT NULL,
    name VARCHAR(100),
    bio VARCHAR(500),
    location VARCHAR(100),
    profile_picture VARCHAR(200),
    account_type_id INT NOT NULL,
    FOREIGN KEY (account_type_id) REFERENCES AccountType(account_type_id)
);

-- Create Connection table to track social media connections
CREATE TABLE Connection (
    connection_id INT PRIMARY KEY,
    follower_id INT NOT NULL,
    following_id INT NOT NULL,
    connection_date DATE,
    FOREIGN KEY (follower_id) REFERENCES User(user_id),
    FOREIGN KEY (following_id) REFERENCES User(user_id)
);
```

In this schema, we have three tables:

1. AccountType: Stores the types of accounts available, where account_type_id is the primary key and type_name represents the type of the account (e.g., Free or Paid).

2. User: Stores information about users, including their account type. The account_type_id column in the User table is a foreign key referencing the account_type_id in the AccountType table, allowing us to differentiate between free and paid accounts.

3. Connection: This table tracks social media connections between users. Each row represents a connection between two users (follower and following). The follower_id and following_id columns are foreign keys referencing the user_id in the User table.

With this schema, you can now track social media connections and distinguish between free and paid accounts for your social media platform. As users sign up and connect with others, you can add new rows to the Connection table to reflect those connections. Additionally, when a user creates an account, you can specify their account type (Free or Paid) by setting the appropriate account_type_id in the User table.

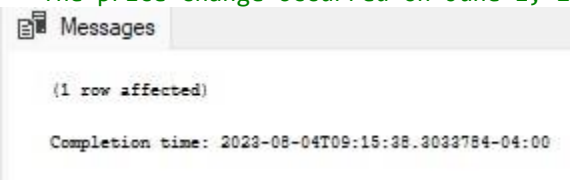
```sql
CREATE TABLE PriceChange (
    PriceChangeID INT PRIMARY KEY,
    OldPrice DECIMAL(10, 2) NOT NULL,
    NewPrice DECIMAL(10, 2) NOT NULL,
    AccountType INT NOT NULL,
    ChangeDate DATE NOT NULL,
    FOREIGN KEY (AccountType) REFERENCES AccountType(account_type_id)
);
```

In this schema, we have added an AccountType column to the PriceChange table, which is a foreign key referencing the account_type_id in the AccountType table to indicate whether the price change applies to free or paid accounts.

Now, let's create three example SQL statements for price changes:

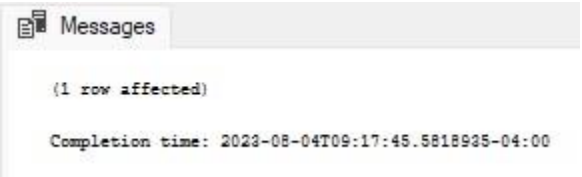
1. Price Change for Free Accounts:

```
INSERT INTO PriceChange (PriceChangeID, OldPrice, NewPrice, AccountType, ChangeDate)
VALUES (1, 0.00, 0.00, 1, '2023-06-01');
-- This represents a price change for free accounts, where both the OldPrice and NewPrice
are 0.00.
-- The price change occurred on June 1, 2023.
```



2. Price Change for Paid Accounts:

```
INSERT INTO PriceChange (PriceChangeID, OldPrice, NewPrice, AccountType, ChangeDate)
VALUES (2, 9.99, 12.99, 2, '2023-07-15');
-- This represents a price change for paid accounts, where the OldPrice was $9.99 and the
NewPrice is $12.99.
-- The price change occurred on July 15, 2023.
```



These SQL statements will add the price changes to the PriceChange table, associating them with the appropriate account types (Free or Paid) based on the AccountType column. By tracking price changes in this way, you can keep a historical record of how prices have evolved for different account types over time.

A trigger is a database object that automatically executes in response to certain events on a table, such as INSERT, UPDATE, or DELETE operations. Triggers are useful for enforcing data integrity, performing auditing, maintaining consistency, and automating certain actions based on specific conditions.

Let's create an example trigger for a hypothetical social media platform to automatically update the user's profile completion status when they update their profile information.

Create the trigger:

```
DELIMITER //
CREATE TRIGGER update_profile_completion AFTER UPDATE ON User
FOR EACH ROW
BEGIN
    IF NEW.name IS NOT NULL AND NEW.bio IS NOT NULL AND NEW.location IS NOT NULL THEN
        UPDATE User
        SET profile_completion = 100
        WHERE user_id = NEW.user_id;
    ELSE
        UPDATE User
        SET profile_completion = 0
        WHERE user_id = NEW.user_id;
    END IF;
END;
//
DELIMITER ;
```

In this example, we assume there is a column named profile_completion in the User table to store the profile completion percentage. When the trigger fires after an update on the User table, it checks if the name, bio, and location fields are not null. If all these fields have non-null values, the profile_completion is set to 100. Otherwise, it is set to 0.

Using the trigger:

Suppose we have the following initial data in the User table:

user_id	name	bio	location	profile_completion
1	Alice	Hi there!	New York	100
2	Bob	NULL	NULL	0
3	NULL	Exploring the world	London	0

Now, let's update the profile information for user_id=2:

```
UPDATE User
SET name = 'Bob Smith', bio = 'Software Engineer', location = 'San Francisco'
WHERE user_id = 2;
```

After this update, the profile_completion for user_id=2 will be automatically changed to 100 by the trigger.

Track Social Media Connections Summary and Reflection

Reflection on social media connections can be beneficial to understand the impact of our online relationships, the nature of our interactions, and their effects on our well-being. The criteria listed below are some of the examples that may or may not be beneficial.

- Types of Connections:

Categorize your connections into different groups (e.g., close friends, family, acquaintances, colleagues, online communities). Reflect on the diversity of your connections and the frequency of interaction with each group.

- Positive Interactions:

Recall positive interactions you had with your connections. How did these interactions make you feel? Did they bring joy, support, or valuable insights into your life?

- Negative Interactions:

Recall any negative interactions you experienced. How did these interactions affect your emotions or well-being?

- Social Media Impact on Mood:

Reflect on how social media interactions affect your overall mood. Do you find yourself feeling happier, stressed, or anxious after spending time on social media?

- Time Spent on Social Media:

Analyze the time you spend on social media daily or weekly. Assess whether it aligns with your personal goals and whether it might be excessive or impacting other aspects of your life.

- Quality vs. Quantity:

Evaluate whether the number of connections or followers matters more than the quality of interactions. Reflect on the value of deeper connections versus superficial ones.

- Unhealthy Habits:

Identify any unhealthy habits related to social media usage, such as comparing yourself to others or seeking external validation. Consider strategies to address and overcome these habits.

- Positive Changes:

Highlight any positive changes or benefits that social media connections bring to your life. Are there any ways you can enhance these positive aspects further?

- Future Intentions:

Set intentions for your future social media interactions. Consider any adjustments you want to make in terms of connection management and online presence.

My database is an application that is named ‘Track Social Media Connections’ which distinguishes the issue of real-life friends and acquaintances or non-real-life friends. Typically, when a person is added to their social media platform, my application will monitor information that is acquired and shared and records it within the application. Users can use that application to export an excel file to track their social media connections.