Schemas and Constraints

Photobook Database

Users Table

The users table stores registered users as well as visitors in the Photobook database. It stores a ***user_id*** which is a set as the Primary Key for this table.

Another constraint in this table is the ***gender*** column. We only allow 3 types of characters – 'M', 'F', 'O' – as the value of this column. This is ensured using the **CHECK** keyword. In case this check fails, the database will return an error.

One interesting constraint we have on this table is demonstrated by the following SQL:

```
CONSTRAINT chk_visitor CHECK (
        (is_visitor AND fname IS NULL AND lname IS NULL AND email IS NULL AND
dob IS NULL AND hometown IS NULL AND gender IS NULL AND password IS NULL)
        OR
        (NOT is_visitor AND fname IS NOT NULL AND lname IS NOT NULL AND email
IS NOT NULL AND dob IS NOT NULL AND hometown IS NOT NULL AND gender IS
NOT NULL AND password IS NOT NULL)
        )
```

This constraint is here to help with the ***is_visitor*** column. Our design choice of having an ***is_visitor*** column is to distinguish between a visitor and an authentic user of the Photobook app. The value of this column (***is_visitor***) is true if the user is just a visitor, and has not registered/signed up on the system. This helps us have a unique ***user_id*** for that visitor in order to process further operations such as adding likes and comments. In summary, if ***is_visitor*** is true, other columns can be **NULL**, otherwise **NULL** values are not allowed (except for the ***dob*** column).

Albums Table

The ***album_id*** column serves as the primary key for the table, ensuring each album has a unique identifier. The ***user_id*** column has a foreign key constraint (***fk_user_id***) referencing the ***user_id*** column in the users table. This constraint maintains referential integrity, ensuring that each album is associated with a valid user. If a user is deleted from the users table, all albums associated with that user will be deleted (**ON DELETE CASCADE**).

There is a trigger (***trigger_check_legit_user***) and a corresponding function (***check_legit_user()***) to ensure that only non-visitor users can create albums. The trigger fires before inserting or updating a record in the albums table, invoking the function to check if the user creating the album is not a visitor.

Photos Table

The ***photo_id*** column serves as the primary key for the table, ensuring each photo has a unique identifier. The ***album_id*** column has a foreign key constraint (***fk_album_id***) referencing the album_id column in the albums table. This constraint ensures that each photo is associated with a valid album. If an album is deleted, all photos associated with that album will be deleted (**ON DELETE CASCADE**).

We are storing photos as binary strings (The **BYTEA** data type allows storage of binary strings).

Friends Table

The combination of ***user_id*** and ***friend_id*** columns forms a composite primary key (***pk_friends***), ensuring that each friend relationship is unique. Both the ***user_id*** and ***friend_id*** columns have foreign key constraints (***fk_user_id*** and ***fk_friend_id***, respectively) referencing the ***user_id*** column in the users table. These constraints maintain referential integrity, ensuring that both users involved in a friend relationship exist in the users table. If a user is deleted, all friend relationships associated with that user will be deleted (**ON DELETE CASCADE**).

We encountered an interesting edge case that we wanted to address. What if the ***user_id*** of one of the two friends is of a visitor and not of a registered user of the system? In this case, we will need to show an error.

To address this problem, we created a trigger (***trigger_check_legit_friendship***) and a corresponding function (***check_legit_friend_relationship()***) to ensure that only non-visitor users can be friends. The trigger fires before inserting or updating a record in the friends table, invoking the function to check if both users involved in the friend relationship are non-visitors.

Tags Table

Both ***photo_id*** and ***tag_name*** are used as a combination to create a composite primary key. This is because in our design of the system we want to easily query tags based on ***photo_id***s. The ***photo_id*** column is a foreign key that references the primary key of the Photos table.

Likes Table

This table manages the likes users give to photos. We felt the need to add this table although it wasn't explicitly mentioned in the assignment description.

Each like is uniquely identified by a combination of ***photo_id*** and ***user_id*** through a composite primary key (***pk_likes***), ensuring that a user can like a specific photo only once. Both ***photo_id*** and ***user_id*** are equipped with foreign key constraints (***fk_photo_id*** and ***fk_user_id***, respectively), which link back to the photos and users tables to maintain referential integrity. The **ON DELETE CASCADE** rule ensures that if a photo or user is deleted, all associated likes are automatically removed.

Comments Table

This table stores comments made by users on photos. Each comment is identified by a unique ***comment_id*** and includes references to both the photo and the user through ***photo_id*** and ***user_id***. These columns are bound by foreign key constraints (***fk_photo_id*** and ***fk_user_id***), referencing the primary keys in the photos and users tables respectively, which helps maintain data integrity. The **ON DELETE CASCADE** option for these foreign keys ensures that any comment is deleted if the associated photo or user is removed.

ER Diagram Overview

The ER Diagram provides a visual representation of the database structure, illustrating entities, attributes, relationships, and constraints.

Entities:

We have 6 entities (photos,tags, comments, likes, albums and users). Each entity with its attributes are listed below.

- **Users:**
  - Attributes: user_id (Primary key), fname, lname, dob, email, hometown, gender, password, is_visitor.
- **Albums:**
  - Attributes: album_id (Primary key), album_name, user_id (Foreign key), datetime.
- **Photos:**
  - Attributes: photo_id (Primary key), album_id (Foreign key), caption, data, datetime.
- **Comments:**
  - Attributes: comment_id, photo_id (Foreign key), user_id (Foreign key), text, datetime.
- **Likes:**
  - Attributes: photo_id (Foreign key), user_id (Foreign key), datetime.
- **Tags:**
  - Attributes: photo_id (Primary key, Foreign key), tag_name (Primary key).

Relationships:

- **Own (Users to Albums):**
  - One-to-Many: A user can own many albums, but an album belongs to only one user.
- **Store (Albums to Photos):**

    - One-to-Many: An album can store multiple photos, but a photo belongs to only one album.
- **Has (Photos to Comments, Likes):**
    - One-to-Many: A photo can have multiple comments and likes, but each comment and like belongs to only one photo.
- **Associate (Photos to Tags):**
    - Many-to-Many: A photo can have multiple tags, and a tag can be associated with multiple photos.
- **Friend (Users to Users):**
    - Self-Relationship: Descriptive attributes include user_id and friend_id.

Constraints:

- Directed bold arrows indicate constraints such as ownership relationships (e.g., one user owning an album).
- Other constraints, including primary keys and foreign keys, are implied within the entity descriptions and relationship cardinalities.