**Design Document**

**Entity Relationship and Database Schema**

**1. Entities**

**User**

Represents a user of the application.

* **Fields**:
  + id (int): Unique identifier
  + username (string): Username.
  + email (string): user Email.

**ProfilePicture**

Represents a profile picture uploaded by a user.

* **Fields**:
  + id (int): Unique identifier.
  + user\_id (int): Reference to the user who uploaded the profile.
  + image\_data (binary): actual image data.
  + is\_active (boolean): Indicates if this is the active profile picture.

**Post**

Represents a post created by a user.

* **Fields**:
  + id (int): Unique identifier.
  + user\_id (int): Reference to the user who created the post.
  + title (string): post title.
  + content (text): post content.
  + category\_id (int): Reference to the category of the post.

**Comment**

Represents a comment made on a post.

* **Fields**:
  + id (int): Unique identifier.
  + post\_id (int): Reference to the post it was made on.
  + user\_id (int): Reference to the user who created the comment.
  + text (text): comment text.

**Category**

Represents a category for posts.

* **Fields**:
  + id (int): Unique identifier.
  + name (string): category name.

**2. Relationships**

**One-to-One Relationships:**

* **User** to **ProfilePicture**: A user can have multiple profile pictures but only one active profile picture (is\_Active=true).

**One-to-Many Relationships:**

* **User** to **Post**: A user can create many posts.
* **User** to **ProfilePicture**: A user can have multiple profile pictures
* **User** to **Comment**: A user can create many comments.
* **Post** to **Comment**: A post can have many comments.
* **Category** to **Post**: A category can include many posts.

**Many-to-One Relationships:**

* **ProfilePicture** to **User**: Each profile picture belongs to one user.
* **Post** to **User**: Each post is created by one user.
* **Comment** to **Post**: Each comment belongs to one post.
* **Comment** to **User**: Each comment is created by one user.
* **Post** to **Category**: Each post belongs to one category.

**Deletion Behavior**

**1. User Deletion**

When a user is deleted:

1. Delete all their profile pictures from **ProfilePicture**.
2. Delete all their posts from **Post**, including all comments associated with those posts.
3. Delete all their comments from **Comment**.
4. Finally, delete the user from **User**.

**Implementation:**

* Use **Cascade Delete** for the user\_id foreign key in **ProfilePicture**, **Post**, and **Comment**.
* Ensure that associated data in dependent tables is deleted automatically.

**2. Post Deletion**

When a post is deleted:

1. Delete all comments associated with the post from **Comment**.
2. Delete the post from **Post**.

**Implementation:**

* Use **Cascade Delete** for the post\_id foreign key in **Comment**.

**3. ProfilePicture Deletion**

When a profile picture is deleted:

1. Remove the profile picture entry from **ProfilePicture**.
2. Ensure that the is\_Active flag is updated if the deleted picture was active.

**Implementation:**

* Use manual logic to update the is\_Active field for the user's remaining profile pictures.

**4. Comment Deletion**

When a comment is deleted:

1. Remove the comment entry from **Comment**.

**Implementation:**

* No cascading rules needed.

**5. Category Deletion**

When a category is deleted:

1. Update all posts associated with the category to have category\_id=NULL or a default value.
2. or Delete all posts associated with the category.
3. Delete the category from **Category**.

**Implementation:**

* Use manual logic to update the category\_id field in **Post**.

**Updated Relationship for ProfilePicture**

To allow users to have multiple profile pictures:

1. Add an is\_active field to **ProfilePicture** to indicate the active profile picture.
2. Ensure that only one profile picture per user can have is\_active=true at any time.

**Logic:**

* When a new profile picture is set as active, update all other profile pictures for the user to set is\_active=false.