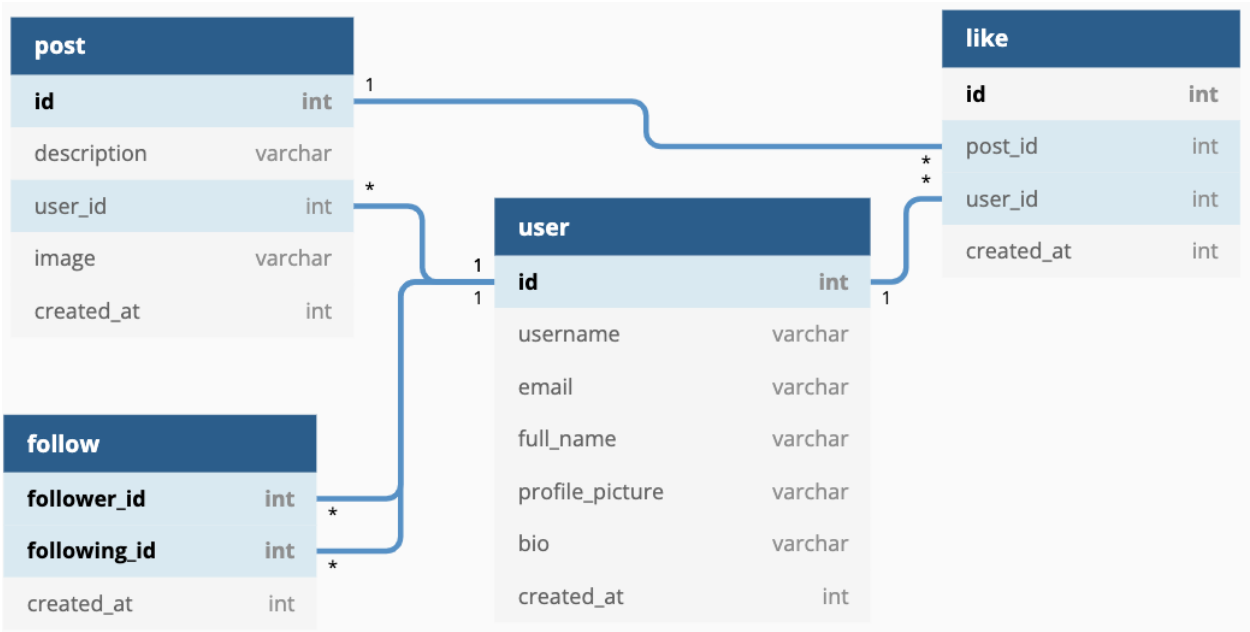
Backend Interview Questions

Imagine we have a social media application. This application has very simple features:

- A user can sign up using a username, email and password.
- Each user has an account containing his username, email, full name, profile picture and bio.
- Each user can follow and unfollow another user, and they can see people who they follow or who follows them.
- Each user can like another user's posts.
- Each user can post a photo which can be seen in his profile in chronological order.

We are using a SQL database to store information. The database diagram is shown below *(For simplicity, authentication information is omitted).*



Q1 - Day-to-day programming (approx. 15 minutes)
We are implementing a simple procedure to get information for a list of posts that might be used in arbitrary places for our project. (Think of this like a random post feed on Instagram.) Write a simple function (signature is given below) to get all information for given post ids.

Note: You don't have to adhere to any language or syntax, just make sure that the execution steps are logically correct.

Data structures that should be returned from the function

```
struct User:
   id: int
   username: string
   full_name: string
   profile_picture: string
   followed: boolean  // whether or not requesting user follows


struct Post:
   id: int
   description: string
   owner: User
   image: string
   created_at: int
   liked: boolean  // whether or not requesting user likes
```

Signature

```
def get_posts(user_id: int, post_ids: List[int]) -> List[Post]: // implement
```

Input parameters

user_id	The requesting user id. Use this to determine *liked* field of *struct Post* and *followed* field of *struct User*
post_ids	List of post ids that are requested

Requirements / Assumptions

- Assume given *post_ids* are unique.
- Procedure should return a list of *struct Post* in the same order as *post_ids*.
- Procedure should place **null** values for non-existing posts in the resulting list.
- You can only read from a single table in each query (no joins are allowed).
- You can use this kind of format for executing SQL queries:

```
db_posts = SELECT * FROM post WHERE id IN post_ids
```

Q2 - Algorithmic design (approx. 30 minutes)

Write a *merge_posts* function (signature is given below) which takes in one parameter *list_of_posts* which is a list of post lists (List[List[Post]]), and returns a list of posts (List[Post]). The function should merge each list in *list_of_posts* to a single list.

Note: You can choose to use a real programming language or a pseudo programming language. Language is not important as long as execution logic is correct.

```
struct Post:
   id: int
   description: string
   image: string
   created_at: int

def merge_posts(list_of_posts: List[List[Post]]) -> List[Post]: // implement
```

Requirements / Assumptions

- You're guaranteed that each element of *list_of_posts* is sorted by *created_at* attribute in ascending order. For the posts that have the same *created_at* value in each element of *list_of_posts*, these posts are sorted by their *id* in ascending order.
- The output of *merge_posts* should be sorted by *created_at* attribute in descending order.
- For posts that have the same *created_at* value, they should be ordered by their *id* in descending order.
- The result should contain unique posts i.e *id* attributes of the result list should be unique (you can assume that if two post has same id, all of their attributes are the same)
- Lists are dynamic-sized arrays so you have index-based access in O(1) time.
- **The time complexity of the function should be at worst O(M*N) where M is the size of *list_of_posts* and N is the sum of size of elements in *list_of_posts.* (a time complexity of O(N*logN) won't be accepted)**