Design of a Database

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the design of databases by discussing:

1. TABLES
2. RELATIONSHIPS

Let’s try designing a database for a photo sharing application. The basic functionality for this application includes users that make posts, showcasing one or many photos per post. Other users can add comments to each post. Believe it or not, just those two sentences of functionality will result in a moderately complex database.

1. TABLES

As was mentioned in the last challenge, there should be a table for each “kind” of record in our database. Tables contain uniform data—so the things that go in each table should all be similar.

Determining the first few tables is usually fairly simple. We need a table for each entity, or object, that was mentioned in the description of the application’s functionality including users, posts, photos, and comments. We’ll need the following tables for each category of users, posts, photos, comments.

Let’s talk about the records we might find in each table.

TABLE 1: USERS

The users table will contain a row, or record, for each user in our application. When someone new registers to user our app, we’ll add a new record in this table. Here are the columns, or fields, that will be present in our users table. We might want to add more fields later, but here’s where we’ll start.

- **User ID**: A unique numeric identifier for each user.
- **Email**: User's email address.
- **Password Hash**: Aversion of the user's password that we can use to verify the user can login to their account.
- **Username**: Name to be associated with the user's account.

TABLE 2: POSTS

This table might seem a little boring at first, but we'll need it later.

- **Post ID**: A unique numeric identifier for each post.
- **Description**: The text that the user will associate with the post.

TABLE 3: PHOTOS

Instead of storing photo information in the posts table, we will create a separate table to store photos. This lets us have a separate record for each photo, rather than having to stuff all the photo information into the post table—too messy!

- **Photo ID**: A unique numeric identifier for the photo.
- **Location**: The place where the photo is stored on our computer.

TABLE 4: COMMENTS

Similar to the photos and posts situation, we don't want to store a photo's comments in the photos table. That would result in a bloated, complicated mess in our photos table. Instead, we'll have a separate record for each comment, all stored in the comments table. Here are the fields for the comments table.

- **Comment ID**: A unique numeric identifier for the comment.
- **Comment Text**: The text of the comment.

Now that there are buckets, or tables, for the data in this photo sharing application, we are able to store data when users interact with our application.

For example, when a user registers, a user record can be added to the user table. When a user adds a post, we are now able to add a post record to the post table, as well as one or several photo records to the photo table. When another user comments on a post, we have a place to store comment data for each comment.

---

2. RELATIONSHIPS

You may have noticed a problem with the tables and fields we've created. According to the fields in our various tables, there is currently no way to determine which post belongs to which user, which photos belong to which post, which comment was made about which post, nor which comment was made by which user.

In order to store this kind of data, we need to find a way to acknowledge and store the relationships between the tables.

Relationship 1: Each post belongs to a single user

Relationship 2: Each photo belongs to a single post

Relationship 3: Each comment was made by a single user

Relationship 4: Each comment is about a single post

These relationships can also be represented like this:

A user has many (at most) posts

A post has many (at most) photos

A post has many (at most) comments

A user has many (at most) comments

Or like this:

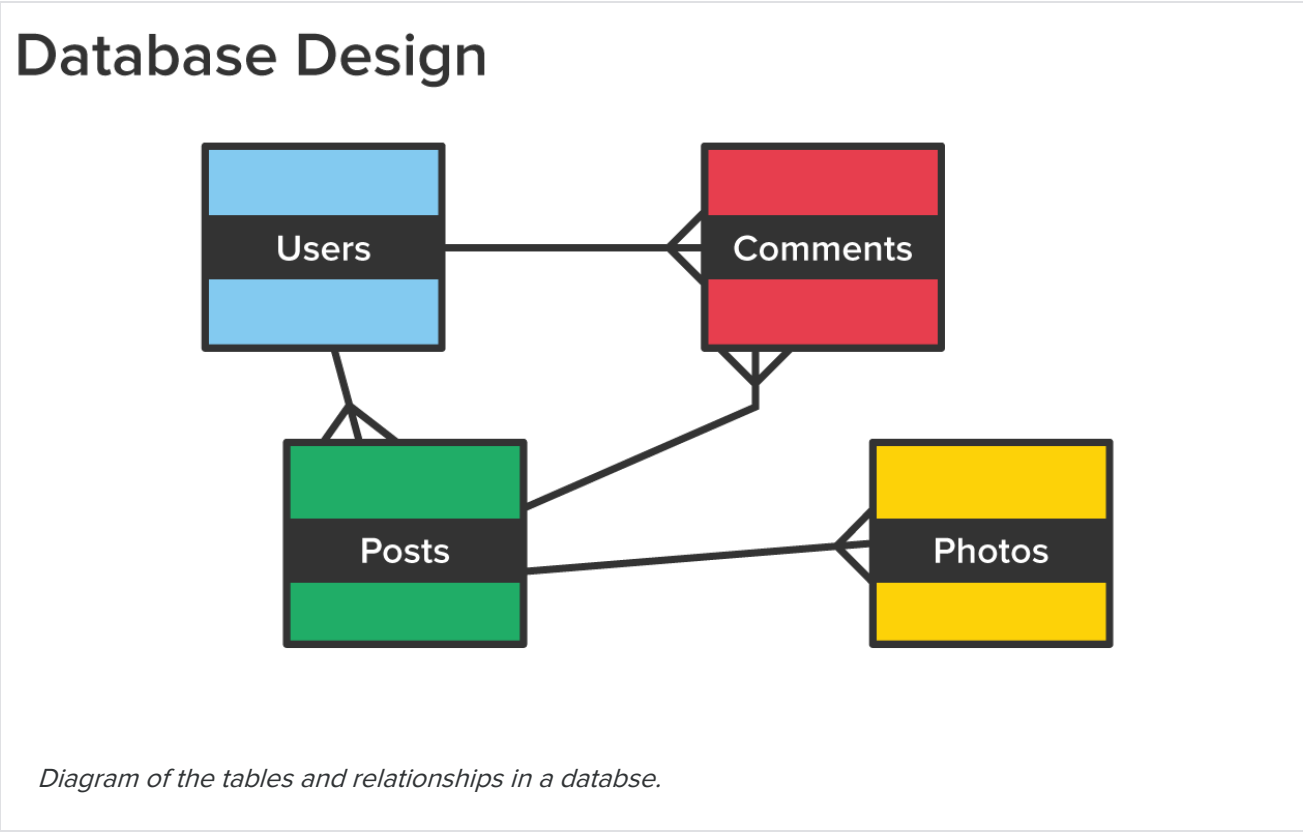

Diagram of the tables and relationships in a databse.

We can store data about the relationships between tables by adding fields to our tables that refer to another record in another table. For example—to represent the relationship that “each post belongs to a single user” we can do the following:

1. Leave the users table as is.

2. Add a field to the post table called User ID. This field will be a place to store the User ID for the user who created a given post!

USERS TABLE

User ID	Email	Password Hash	Username

POSTS TABLE

Post ID	Description	User ID

We'll replicate this for the post/photo relationship as well.

POSTS TABLE

Post ID	Description	User ID

PHOTOS TABLE

Photo ID	Description	Post ID

This process of fields referring to fields in other tables allows the tables to relate data across the database, while still keeping "like" data together, in its own separate table. All photo records can exist in a table where fields are solely about photo data, but we can refer to the Post ID field in the photo table to understand which post a given photo belongs to.

Database design is a highly complex task that takes a lot of planning and evaluating. Determining the right tables, fields, and relationships is not as straightforward that one might assume, but once data is represented correctly within a database, a web application has a solid foundation to build upon.

Hopefully, you've gotten a feel for the kind of critical thinking that's needed to design a database well. It's one of the many tasks for web developers that requires very little coding, and instead, careful thought, planning, and collaboration.