Database Design Blueprint

Presented by: John Morris

1. Object Model

We first need to talk about developing an object model before we can discuss how to structure your database, because your database structure depends on the object model you develop.

Of course, if you're not sure what an object model is, then it's hard to understand how to go about building one. So...

WHAT Is An Object Model?

First, consider that every application you build will ultimately be a collection of "objects" that have certain properties, can perform certain actions, and have certain relationships with other objects.

For example, let's look at WordPress...

Object	Properties	Actions	Relationships
POST	Title Content Date Status Type Author Etc...		Category Tag Etc..
USER	Login Password Name Email Etc...	Create POST Edit POST Delete Post View POST Etc...	POST
CATEGORY	ID Name Slug Etc...		POST

Of course, the example above is simple, but what it demonstrates is that your application will ultimately be a collection of objects that have certain properties, those objects will have certain relationships with each other, and certain objects will be able to perform certain actions on or with other objects.

Your job is to define what those objects are, what properties they possesses, what actions each can take, and how they might be inter-related.

That is object modeling.

So, now it's your turn. Use the table below to lay out the object model for the application you're looking to build:

Object	Properties	Actions	Relationships

2. Database Structure

Now, that your object model is built structuring your database become easy because it flows naturally from your object model.

Here's how to map your object model to your database structure:

1. Create a Table For Every Object

In our WordPress example, we'd create three tables: a) posts, b) users, and c) categories. Which, by the way, if you look at the WordPress database you'll see those tables represented (categories is a bit different because of the use of a term taxonomy... but that's for another lesson).

Of course, you can prefix them based on your application if you'd like. Again, using our WordPress example that would be a) wp_posts, b) wp_users, and c) wp_categories.

2. Turn Your Object Properties Into Your Table Columns

Using our example, the fields/columns we'd create in our posts table would be:

- Title
- Content
- Date
- Status
- Type
- Author

Also, keeping in mind that every table should have an ID field that auto-increments. The key here is to pay close attention to whether the field/column you're creating is truly a property OF that object or a relationship BETWEEN it and another object.

For example, it might be tempting to think of a category as a property of the post object. But, if you think about it... it's really not.

Instead, a category is its own object to which a post can belong... which is a type of relationship and why we use a relationships table to track what posts belong to what category.

3. Create a Meta Table For Each Object

This isn't readily apparent when you create your object model, but it's absolutely critical for creating a flexible database design.

The idea here is that you know what properties are critical to your object, but you also don't know every property an object could ever have.

You need a way to be able to add non-essential, but useful, properties to your objects without cluttering up your object table.

That's exactly what a meta table is. It allows you to associate certain metadata with an instance of an object in a flexible manner.

Using our example, you'd have:

1. postmeta
2. usermeta
3. categorymeta

The columns/fields in meta tables are usually:

1. ID
2. object_id
3. meta_key
4. meta_value

This allows you to attach any custom named property to an object and gives you the flexibility to account for situations that may arise that you can't predict right now.

4. Create a Relationships Table For Each Relationship

Relationships tables allow you to "tie" to objects together in a fluid manner. Keep in mind that the table itself doesn't necessarily define the nature of the relationship, but more just THAT there is some kind of relationship.

Your application will define a context that gives those relationships meaning.

I say that because you might be confused by the simplicity of a relationships table. Generally, a relationships table will only require two columns fields:

1. object_1_id
2. object_2_id

Using our example, that would be:

1. post_id
2. category_id

That's because, we only need to know that these two objects are somehow related. The rest of

our code and the purpose of our application will give us the meaning of the relationship... e.g that a category is a way of organizing posts and that "the post with an ID of 1 belongs to the category with an ID of 3"... and so on.

Of course, you'll want to build one of these relationships tables for every relationship you defined in your object model.

In our example, we might have the following tables:

1. post_categories
2. post_tags

And, that's it. If you go back over everything then, our example database might have these tables:

- posts
- post_meta
- post_categories
- users
- user_meta
- categories
- category_meta

Of course, for your application this will look different, but walking through this process will help you get clear on how your application will function and help you set up your database in a way that provides maximum flexibility and scalibility.