Normalization

The process of separating your data into tables and creating primary keys is called normalization. Its main goal is to make sure each piece of information appears in the database only once. Duplicating data is very inefficient, because it makes databases larger than they need to be and therefore slows down access. But, more importantly, the presence of duplicates creates a strong risk that you'll update only one row of duplicated data, creating inconsistencies in a database and potentially causing serious errors.

In the process of splitting a database into multiple tables, it's important not to go too far and create more tables than is necessary, which would also lead to inefficient design and slower access.

First Normal Form

For a database to satisfy the First Normal Form, it must fulfill three requirements:

1. There should be no repeating columns containing the same kind of data.
2. All columns should contain a single value.
3. There should be a primary key to uniquely identify each row.

Second Normal Form

The First Normal Form deals with duplicate data (or redundancy) across multiple columns. The Second Normal Form is all about redundancy across multiple rows. In order to achieve Second Normal Form, your tables must already be in First Normal Form. Once this has been done, Second Normal Form is achieved by identifying columns whose data repeats in different places and then removing them to their own tables.

Third Normal Form

Once you have a database that complies to both the First and Second Normal Forms,it is in pretty good shape and you might not have to modify it any further. However, if you wish to be very strict with your database, you can ensure that it adheres to the Third Normal Form, which requires data that is not directly dependent on the primary key but that is dependent on another value in the table should also be moved into separate tables, according to the dependence.

When Not to Use Normalization

Now that you know all about normalization, I'm going to tell you why you should throw these rules out of the window on high-traffic sites. That's right, you should never fully normalize your tables on sites that will cause MySQL to thrash.
Normalization requires spreading data across multiple tables, and this means making multiple calls to MySQL for each query. On a very popular site, if you have normalized tables, your database access will slow down considerably once you get above a few dozen concurrent users, because they will be creating hundreds of database accesses between them. In fact I would go so far as to say you should denormalize any commonly looked-up data as much as you can.

You see, if you have data duplicated across your tables, you can substantially reduce the number of additional requests that need to be made, because most of the data you want is available in each table. This means that you can simply add an extra column to a query and that field will be available for all matching results.

Of course, you have to deal with the downsides previously mentioned, such as using up large

amounts of disk space, and ensuring that you update every single duplicate copy of data when one of them needs modifying.
Multiple updates can be computerized, though. MySQL provides a feature called triggers that make automatic changes to the database in response to changes you make.

Another way to propagate redundant data is to set up a PHP program to run regularly and keep all copies in sync.
The program reads changes from a “master” table and updates all the others.

However, until you are very experienced with MySQL, I recommend you fully normalize all your tables, as this will instill the habit and put you in good stead. Only when you actually start to see MySQL logjams should you consider looking at
denormalization.