## Best Practices in Database Design

This section presents a few recommendations that should be taken into account when planning out your database. The database design process doesn't have to be so intimidating. With the following rules in mind, you should be able to achieve the best compromise between speed, occupied disk space, and lack of errors. If the database is poorly designed, and this is discovered late in the development process, recreating it, and all components that use it, will really push up development time.

Here are recommendations:

* Comment your code! Comments are useful as they allow anyone to understand the structure of the database. Comments do not affect the speed of queries executed on a table.
* Although databases allow binary fields where you can store different types of files, it is recommended that you only store the server path to these files in the table. Storing big files in the database will not only increase the disk space occupied, but also make the whole database slower. The files themselves should be kept in folders accessible to the web server.
* When defining the table columns, make sure you correctly identify which fields require data and which do not. Don't set all your fields to not null, if you don't want to force users to enter data for them. On the other hand, defining your constraints very loosely or not defining any constraints at all will make your application vulnerable and ruin your data integrity.
* If a column storing the same data appears or is referenced in more than one table (e.g. a foreign key), it should have the same data type throughout the database.
* When running queries on the database, use only the column names that you need to retrieve. This can reduce network traffic and disk operations.
* When running an insert action query, use the columns list into which to insert instead of the table name (when you want to insert data in all the current columns). This way, if new columns are added to the table, your query will still work.

## Database Naming Convention

A naming convention “standardizes the structure, coding style, and logic of an application”. This means that everyone who works on the database will use the same names, notations and style. Consequently, everyone will be able to understand and alter at any point in time the database, without having to go through a lot of trouble.

**Why use a naming convention?**

A naming convention should be used in the process of working with databases because of several reasons:

* It provides a clear structure, each component using the standard naming rules.
* The database logic can be understood by anyone knowing the naming convention.
* Maintenance work is reduced, as future developers will not have to waste time figuring out what each field is.

Whatever the naming convention, make sure it becomes a standard across the company and is followed by everyone in your department. The following paragraphs presents the conventions recommends for developing web applications. In no way are they imposed, or the only ones possible to use.

**General conventions**

The following conventions apply to all of the elements of a database:

* All names used throughout the database should be lowercase only. This will eliminate errors related to case-sensitivity.
* Separate name parts by underlines, never by spaces. This way, you improve the readability of each name (e.g. product\_name instead of productname). You will not have to use parentheses or quotes to enclose names using spaces as well. The use of spaces in a database name is allowed only on some systems, while the underline is an alphanumeric character, allowed on any platform. Thus, your database will become platform independent.
* Do not use numbers in the names (e.g. product\_attribute1). This is proof of poor design, indicating a badly divided table structure. If you need a many-to-many relation, the best way to achieve it is by using a separate linking table. See how [here](http://www.interaktonline.com/Support/Articles/Details/Design+Your+Database-Databases+By+Example.html?id_art=24&id_asc=222#m2m). Moreover, using numbers to differentiate between two columns that store similar information might be an indication that you need an extra table, storing that information. For instance, having a location1 column in a manufacturers table and a location2 column in a distributors table could be solved by creating a separate table that stores all locations, and that is referenced by both the manufacturers and distributors tables via [foreign keys](http://www.interaktonline.com/Support/Articles/Details/Design+Your+Database-Database+Concepts.html?id_art=24&id_asc=216#foreign).
* Do not use the dot (.) as a separator in names. This way you will avoid problems when trying to perform queries, as the dot is used to identify a field in a specific column. In SQL language, manufacturer\_man.address\_man means the address column from the table that stores information about manufacturers.
* Do not use any of the reserved words as names of database elements. Each database language uses some words as names for internal functions, or as part of the SQL syntax. For instance, using order as the name of a table that stores product orders from an online shop is bad practice, because order is also used in SQL language to sort records (ascending or descending).  
  For a complete list of the reserved words that you should not use when naming the database elements, consult your specific database software manual. See the list of reserved words for MySQL [here](http://dev.mysql.com/doc/mysql/en/reserved-words.html).
* When naming the elements, do not use long or awkward names. Keep them as simple as you can, while maintaining a clear meaning. It's also a good idea to use names which are close to the natural language: product\_description is certainly a better name for a column that stores product descriptions than dscr\_pr or some generic name as field\_1.

**Database names**

Each database must have a name of its own, which should also follow some conventions:

* Use the project name as the name of the database.
* Prefix the database name with the owner name, separated by an underscore. The owner might be a person (e.g. the project manager) or the application for which the database is created. For example, acme\_catalog can be a good name for the database storing  
  the product catalog of the ACME company.

**Table names**

Tables are some of the most common elements used in an application, as they store  
the columns, and as such are mentioned in each query. Therefore, the following conventions should apply to tables:

* Prefix tables that define the same larger entity with a 2 or 3-letter acronym that identifies it. For example, e.g. hr\_applicant\_app, hr\_job\_job and hr\_resume\_rsm are  
  all tables that belong to the Human Resources Department of a large corporation database.
* Do not use generic prefixes, such as tbl\_, or db\_, as they are redundant and useless.
* Use short, unambiguous names for each table, restricted to one word, if possible. This way tables can be distinguished easily.
* Use singular for table names. This way, you avoid errors due to the pluralization of English nouns in the process of database development. For instance, activity becomes activities, box becomes boxes, person becomes people or persons, while data remains data.
* Use clear names. Do not overdo it using abbreviations and acronyms. While using a shorter name might help the developers, it makes the meaning less clear to other members of the team. Using clear names makes the design self-explanatory.
* Prefix lookup tables with the name of the table they relate to. This helps group related tables together (e.g. product\_type, product\_status), and also helps prevent name conflicts between generic lookup tables for different entities. You can have more than one generic lookup table for an existing master table, but which address different properties of the elements in the table.

**Column names**

Columns are attributes of an entity, describing its properties. Therefore, the name they carry should be natural, and as meaningful as possible. The following conventions are recommended:

* All keys are used for indexing and identifying records. Therefore, it's a good practice to put the id particle in their name. This way, you'll know the field is used as a key.
* The primary key is used to uniquely identify each record. That is why its name should be made up of the id particle. For instance, for the table product, the primary key is product\_id.
* The foreign key name is generally the same as primary key. This way, the table being referenced is obvious from the key name, and sql statement can particle natural join statement.