**ERD**

**Conventions**

We will be following a set of conventions to create our database. We don't have to follow these conventions, but we recommend our students to follow them for the following reasons:

1. Developers can have a better understanding of your database if you are using a set of industry standards.
2. Developers can create software to automate a lot of the queries if some assumptions can be made. In later chapters, you will learn about Object Relational Mappers (ORM), which are programs that other developers use to make database queries easier by providing some handy functions. These functions will only work if we have followed conventions that ORM author expects, which are primarily based on set industry standards.

**Guidelines**

Down the line, you may find yourself working with a company that has set up their database conventions a little bit differently, but these are the guidelines that we feel are best for this course:

1. **make the table name plural and ALL lowercase**- make it plural (ex. users, leads, sites, clients, chapters, courses, modules)
2. **use "id" as the primary key**- name it *id*(also make it auto-incremented).
3. **name foreign keys with singular\_table\_name\_id**when referencing to a primary key in another table name it [*singular name of the table you're referring to]\_id* (ex. user\_id, lead\_id, site\_id, client\_id, chapter\_id, course\_id, module\_id).
4. **use *created\_at* and*updated\_at*** as columns for the timestamp in EVERY table you create.

When we do things in ORM or in Ruby on Rails, it becomes extremely important that we follow these naming conventions.

Data Types

The following are the data types that you will be using 95% of the time. Although there are quite a few other data types that you can use, focus on these for now.

**Simple Data Types:**

* **VARCHAR**
* Used to store non-numeric values that can be up to 255 characters. It is called a VARCHAR because it can store a variable number of characters and will only use the space required for each record that is stored in the database.
* VARCHAR should be used for values with different character lengths like an email, first\_email, or last\_name.
* **CHAR**
* Also used to store non-numeric values, however, it will use up all space for the set number of characters regardless of what value is added. For instance, if I set CHAR(15), and I try to store the value “Coding”, it will use up the equivalent of 15 characters even though “Coding” is only 6 characters long. Char is good to use for things that will always be given number of characters. Char would work well for something like a state\_abbreviation.
* **INT**
* Used to store integers.
* The columns that you will find mostly using the INT are things like a unique identifier for each table. The majority of rows in a table will not exceed 2.1 billion records. INT is good to use for most normal number values like a phone\_number or a zip\_code.
* **Unsigned**(positive numbers only) – can store numerical values from 0 up to 4294967295
* **Signed**(positive and negative numbers) – can store numerical values from -2147483648 up to 2147483647
* **BIGINT**
* BIGINT would be used for columns that would need to store huge numbers. In most cases, you wouldn’t need BIGINT, but if you wanted to store something like a Facebook id when using Facebook’s API, since they have over a billion users the id will need to be a data type of BIGINT.
* **Unsigned**(again positive numbers only) – can store numerical values from 0 up to 18446744073709551615
* **Signed** (positive and negative numbers) - can store numerical values from 9223372036854775807 to -9223372036854775808.
* **TINYINT**
* TINYINT would be good to use for numbers that will be relatively small. A good example of something that would use a TINYINT is user level identifier(0 – inactive user, 1- active user, 9 – admin).
* **Unsigned –** Can store numerical values from 0 to 255.
* **Signed –** can store numerical values from -128 up to 127.
* **FLOAT**
* Used to store floating point numbers(numbers that need to have decimal places). An example column of this would be like an item\_cost.
* **TEXT**
* Used to store a large amount of text, like a description, message, or coment. Use this for any text that VARCHAR() is too small to handle.
* **DATETIME**
* Used to store a data and time in the format YYYY-MM-DD hh:mm:ss

**MySQL Notes**

SQL – For any relational database like MySQL, you will interact with it using SQL. In the previous chapter, we learned how to design a schema for our data: we set up the collections that we needed, and we set up the relationships among tables. Now we will see the importance of relationships, and how to use SQL to adjust the data in any way we can imagine.

SQL stands for structured query language, which is a programming language designed for managing data in relational databases. SQL statements are used to perform tasks; they can SELECT data, SELECT data WHERE some conditions are true, INSERT data, UPDATE data, DELETE data, and JOIN different tables together. As we go over all of the basic SQL commands, be patient. You will be learning a domain-specific language, unrelated to languages you may have previously seen. However, mastering SQL is the key to mastering the database component of your application.

Database and SQL

We will install and run a MySQL server on our operating system to connect first to MySQL Workbench, and later our web applications. This will be a database server, which will be listening for connections on localhost (just like our Flask web servers). Also like our web servers, our MySQL Workbench via localhost and whichever port our MySQL server is using.

**Inserting Records**

INSERT INTO **table\_name** (column\_name1, column\_name2)

VALUES('column1\_value', 'column2\_value');