**SQL**

Why ERD first?

ERD is the abbreviation for 'Entity Relationship Diagram'. That is just a fancy way of saying that ERDs are essentially visual planes of how your database looks and behaves. ERD and SQL work together very intimately. An ERD is a map of the structure of how we want to store our data, and SQL is the language we use to manipulate the data according to the relationships we define in our ERD. Learning the design of the database first will help us visualize what our relational databases look like, which makes it much easier to choose the actual SQL syntax.

Why ERD?

ERD is a process of designing your tables and establishing relationships between them, making your data relational. Almost all imaginable data can be stored in a relational way, there really isn't anything you can't do using a relational database like MySQL. Later, you will learn non-relational databases where everything is stored in a single table. There are advantages and disadvantages for both, but we find that it is much easier to move from a relational database to a non-relational database.

Main topics for database design

There are many different terms and concepts that you will learn throughout this chapter, but they all point to a very simple concept: Do not repeat data. If you can remember this concept, the rest is to familiarize yourself with the terminology.

Database Relations

One to one

One to many

Many to many

Three (3) forms of normalization

MySQL Workbench

Type of data

What is the point?

When we normalize our tables, we do not repeat data. This means that in the long term, we can use our storage space more efficiently.

There is also another advantage that we obtain by normalizing our tables and establishing relationships between them. Later we will learn that identifiers and foreign keys serve as the glue between our tables. With SQL, we can manipulate our tables and create the custom table we need for the job in question.

By dividing our data into different tables, we make each table good at one thing: store instances or rows of that data. In addition, if we separate our tables, our database becomes more modular. This means that we can create our own custom tables depending on the task in question using SQL.

We will learn this in the next chapter, but it is crucial to understand that we are using the strategy of normalizing our tables and establishing relationships between them because we want to save storage space; and also because it makes our database more modular so we can create more variety of custom tables using SQL.

What is Normalization?

Database normalization is simply a convention for splitting large tables of data into smaller separate tables with the primary goal being to not repeat data. Why is this so important? Let's say that you wear a watch so you can check the time, because it's very important for you to know what the current time is. Would wearing eight watches make it easier? No way! Now we have eight conflicting accounts of what the proper time is. Worse yet, if we ever want to update the time, we'd have to do it for every watch independently. That's not very efficient!

You can apply a similar concept to database design. If we want to store a user's email address, we'd want to store it in only one place. Then, if we ever need to refer to it again, we'd simply use the id. The id will never change, so even if we update the user's email address, none of the other connections we defined in our database will be damaged. Neat!

Below are the three main rules of database normalization. You should use these as a guide for designing your ERDs. Always remember, however, that they are common convention, and not absolute rules. It is possible to take normalization to an extreme. For example, a simple address field. One state can have many cities, one city can have many streets, one street can have many buildings, one building can have many apartments, one apartment can have many residents... and so on. Yikes, that can get really crazy really quick! In the next sections, you'll learn more about why this type of complexity can be inefficient, especially for simple assignments.

First Form

Each Column in your table can only have 1 value.

Ex. You should not have an address column in your table that lists the address, city, state, and zip, all separated by commas.

Second Form

Each Column in your table that is not a key (primary or foreign) must have unique values.

Ex. If you have a movies table with a categories column, you should not have a category repeated more than once.

Third Form

You cannot have a non-key column that is dependent on another non-key column.

Ex. If you have a books table with columns publisher\_name and publisher\_address, the publisher\_address and publisher\_name should be separated into a separate table and linked to books with a foreign key. The publisher\_address is dependent on the publisher\_name and neither column is a key column.

Conventions

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:

make the table name plural and ALL lowercase - make it plural (ex. users, leads, sites, clients, chapters, courses, modules)

use "id" as the primary key - name it id (also make it auto-incremented).

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).

use created\_at and updated\_at as columns for the timestamp in EVERY table you create.

Simple Data Types:

VARCHAR(number of characters)

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\_name, or last\_name.

CHAR(number of characters)

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 a 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 up 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 for this would be like an item\_cost.

TEXT

Used to store a large amount of text, like a description, message, or comment. Use this for any text that VARCHAR() is too small to handle.

DATETIME

used to store a date and time in the format YYYY-MM-DD hh:mm:ss