Lesson 6-Database Design: Part 1

Lesson 6 Concepts

1. Database tables ***model*** real-world ***entities***
2. Each row in a table represents a single, unique entity
3. The columns of a table describe the variable properties of the entity
4. MySQL has defined data types that it accepts
5. A ***schema*** is a blueprint plan for a database that specifies the tables, columns and data types for the database

Database Tables Model Real-world Entities

Part of the skill of database design is knowing what should be contained in a table, and when a new table is required. A general guideline is that each table should represent one "entity" of your application. That is to say, a table should represent one aspect or area of concern for your application. For example, a school database would have a table of teachers, a table of classes, a table of students, a table for non-teaching staff, et cetera, all distinct parts of a real-world institution. An e-commerce site would need (at the very least) a products table and a sales table; quite probably a shipping table as well - what else? Because the table acts as a metaphor for your real-world entity, data structures are often referred to as "models" of the entity, in much the same way that a model car is not an actual car. A model is a reduction of the actual *thing* to the *qualities* of the thing that are relevant to the context of the database.

The Columns of a Table Describe the Variable Properties of the Entity

If a table stores the *qualities* of the thing that are relevant to the context of the database, the columns of the table specify *what* the qualities are that we wish to track. Flashing back to Lesson 1, what are the questions that we need to answer with this data in our web application? For our student model (table), what are the qualities that make one student different from the next that are relevant to our online application? Name, age, grade, address, course… these qualities are the variable properties for the student entity that we need to know about for our web application. Tying in to our discussions on primary and foreign keys from the last lesson, what will the unique identifier column value (primary key) be for each table, and does this table need columns that refer to other tables (foreign keys)? Each column should only store one piece of data. If you find you are trying to store multiple values in one column, you may need a new column or table. Additionally, each column can only store one data type.

Each Row in a Table Represents a Single, Unique Entity

Within the students table, each row would represent one unique student. The same student would not be in the database twice. Even if a student had the same first and last name as another student - or even the same birthday - the first column (typically containing a unique id number) would ensure that no two rows were fully identical. Skillful database design means making sure that you have everything that you need for your application to operate successfully, without unnecessary redundancy or inefficient use of storage space. In each row, we store relevant values for that entity that may vary in value from entity to entity.

MySQL Has Defined Data Types That It Accepts

All programming languages specify which data types they are able to work with. Deciding which data type to use becomes critical in database design because we generally strive to use as little space as possible - but we don't want to lose data because we didn't make enough room for it. In a well-designed database, each column will have one data type that it will accept, and which you will specify when you are creating the database. In MySQL, there are three categories of data types that it uses: **numerical** data types, **string** data types, and **date/time** data types. Below are summaries of the most frequently used data types, however, it is important to know that there are many more, and that they have variations of some types to specify their size. For example, INT (integer) can store a number ranging from -2147483648 to 2147483647. TINYINT can store a number ranging from -128 to 127. There is also SMALLINT and MEDIUMINT with different ranges. Our W3Schools friends have a great reference page here: https://www.w3schools.com/sql/sql_datatypes.asp. These numerous options mean that, in addition to deciding on the data type, consideration needs to be put into the *size* of the values that you wish to store. Again, the balance to strike is making sure that you have all of the information that you need for your application, without wasting storage space.

Numerical Data Types

- **INT** - (short for '**int**eger') is for storing numbers (negative or positive) that do not require a decimal value. You specify the number of characters your integer can have, and you can also specify *unsigned*, which limits the data to whole numbers (includes 0, no negative numbers). e.g. ***`temperature INT(2)`*** With a limit of 2, possible values range -99 to 99.
- **DEC** (short for '**dec**imal') is for numbers that have decimal values. In the parentheses, you specify two parameters: the maximum number of character in the number including the decimal, and the number of decimal places. For storing monetary values: e.g. ***`payment DEC(5,2)`***. This could store -999.99 to 999.99. For scientific values, there are **FLOAT** and **DOUBLE** data types.
- **BOOL** is for Boolean values, however, we don't store "true" or "false", we store the number 1 (true) or 0 (false).

String Data Types

- **CHAR** - (short for '**char**acter') is for storing strings that have a fixed length, that is, when you know the exact amount of character it will be. A country code is two characters, a phone number is 10 characters, and password encryptors will hash a string to a set number of characters. When you create your table, you specify how many characters this column will store, to a maximum of 255 characters: e.g. ***`country CHAR(2)`***
- **VARCHAR** (short for '**var**iable **char**acters') is storing strings that will not have a set number of characters, though you still specify the maximum number of characters. e.g. ***`first_name VARCHAR(30)`***
- **TEXT** is for storing strings that are longer than 255 characters. For example, a blog post, or a product description would be more than 255 characters. e.g. ***`description TEXT`***
- **BLOB** is similar to TEXT, except that it stores a string of binary data. Media files can be stored in this way.

Date/Time Data Types

- **DATE** is for storing a date value in the format **YYYY-MM-DD** (note the dashes).

- **TIME** is for storing a time value in the format **HH:MM:SS** (note the colons).
- **DATETIME** is for storing (wait for it…) both of the above values with a space in the middle: **YYYY-MM-DD HH:MM:SS**.
- **TIMESTAMP** is the same format as DATETIME, but can automatically grab and insert the server date and time when a row is added or updated. e.g. ***`account_created TIMESTAMP`***.

A Schema Is a Blueprint Plan for a Database That Specifies the Tables, Columns and Data Types for the Database

Before creating our database, it is helpful to create a plan for what your database will consist of. A formal plan document is called a schema. The schema does not contain any data, it is simply a blueprint for building the database, specifying all of the tables that will be included, the columns for each table, the data type for each column, along with other structural configurations. Although there are software applications that you will help you create a schema ("entity relationship" software, such as, dbdiagram.io), a Microsoft Excel spreadsheet will do the trick (you can colour-code the keys), and I strongly encourage you to start with good ol' paper and pencil to identify your tables and columns. As Benjamin Franklin said, "Failing to prepare is preparing to fail". While you are designing one table you will (inevitably) discover that you should have added something in an earlier table, or that what you have set as a column should actually be a table. These discoveries are easy to fix in a spreadsheet, but become more difficult to fix once you have created a table - and tricky once you have added data to that database. Figure it all out *before* you open up PHPMyAdmin by creating a schema.

References

Thomson, L., & Welling, L. (2003). *MySQL tutorial*. Sams.
Chapter 3: Database Design Crash Course (pp. 27-32)
Chapter 4: Creating Databases, Tables and Indexes (pp. 51-54)

The above reference is available through the Humber Library (library.humber.ca).

Database entity relationship application: https://dbdiagram.io/home

W3Schools complete list of MySQL data types: https://www.w3schools.com/sql/sql_datatypes.asp