Functional Dependencies

One of the key issues in a database is the problem with data redundancy. Let's look at an example of what we mean by that. If we have the following table:

Student ID	Student	Major code	Major name
1000	Bob	CS	Computer Science
1100	Alice	CS	Computer Science
1200	Sandy	BA	Bus. Accounting
1300	Ed	PH	Physics

We are storing for each student their id, their name, their major code, and the name of the major. In this situation, storing the major name with the major code is redundant. Here are some examples of the problems caused by this:

- Redundant storage. Because we have multiple copies of the same information, we are wasting storage space.
- Insertion anomaly: Imagine now we want to perform an insert of the following values:
  - (1400, Susan, CS, Child Services)
  - Does this imply that CS does not equal Computer Science? Is it a mistake? We have created an inconsistency.
- Update anomaly: As with insertion, what does it mean to change Bob's Major code to BD without changing the Major name? Another inconsistency.
- Deletion anomaly: If we delete Ed from the table, is the fact that PH = Physics available anywhere? Or have we just lost that information?

Functional dependencies are a mechanism used to identify these redundancies and eliminate them where appropriate. There may be times where you want to keep redundant data in the system. The most common of these is where we store zip code with a customer's name. Zip code is usually redundant in that it can be derived from the address, city, and state – however we don't want to incur the cost of a lookup every time we print the customer's information, so we allow the redundant storage. For now, though, let's focus on identifying the redundancies and learn how to eliminate them.

A functional dependency is a type of an integrity constraint akin to a key. The form take place like this: Every time I have a value of one field or set of fields (such as zip code), there is a corresponding value in another field. Example: If the value of zip code is 80521, we know that the value of city is Fort Collins and we know that the value of state is Colorado. We illustrate that as follows:

Zipcode -> City, State

And we say it as Zipcode implies City and Zipcode implies State. Using the example table above we can say that StudentID -> Student, Major Code, and Major Name. We can also say that Major code -> Major name. Using this terminology, we can then apply rules to define relationships, etc. The primary key of a table must imply all the other attributes associated with the value in the primary key by definition. This leads us to Armstrong's Axioms.