Table	Columns (primary keys underlined, foreign keys in bold)
Countries	name, code, surface_area, continent, gnp, population, life_expectancy, government, head_of_state, **capital**
Cities	id, name, region, population, **country_code**
CountriesLanguages	**country_code**, language, official, percentage

Table A.5 world database schema version 4 (tweaks)

Normalization

A common mistake many people make (including seasoned web developers) when creating a database is putting too much information in one big table. For example, for the world database, why not combine Countries and CountriesLanguages into a table like in Table A.6?

name	code	surface_area	continent	gnp	population	life_expectancy	government	capital	language	official	percentage
Argentina	ARG	2780400	South America	340238	37032000	75.1	Federal Republic	69	Spanish	T	96.8
Argentina	ARG	2780400	South America	340238	37032000	75.1	Federal Republic	69	Italian	F	1.7
Argentina	ARG	2780400	South America	340238	37032000	75.1	Federal Republic	69	Indian	F	0.3

Table A.6 CountrySpeak combined table (poor design)

The reason this is bad is that every country that speaks many languages (and most do), you will be repeating the name, code, surface area, continent, GNP, population, life expectancy, government, head of state, and capital for the country for as many languages as the country speaks.

Repeating data is bad for two reasons. The first is quite obvious: space. The less you repeat the less space you take in the database. Well, these days large hard drives are pretty cheap so for small or mid-sized databases, maybe this isn't such a compelling argument.

A second reason repetition is bad that is slightly less obvious, but even more important: data consistency. When you have repeated data in a table it is very easy to have inconsistencies when you insert, update, and delete data. For example, you could easily mistype inserting 3703200 as the population for one of the rows above. The difference between 37032000 and 3703200 is huge. Now when an Ayuda user wants to know the population of Argentina, which value should they trust (both seem like viable populations for a country)? An update example: when Argentina elects a new president, an Ayuda user could easily forget to update all rows, again leaving the database in an inconsistent state. Lastly, a deletion example: Ayuda decides they no longer want to keep information about languages spoken in Argentina and so they delete all rows. Unintentionally, they have deleted all information about Argentina even though they meant only to delete information about languages spoken in Argentina. In a world where "Data is King", inconsistency is something to be avoided if at all possible.

How can we minimize data duplication and inconsistencies in our schema? The answer is a technique called normalization. There are various levels of normalization and the higher the level the higher the guarantee of consistency. We'll go through the first three levels of normalization as once you get to the third level you are guaranteed to be free of most update, insert, and deletion errors.

A table is at the first level of normalization, also called First Normal Form, if and only if there are no repeated rows (each row has some unique information) and there are no multi-valued columns. It is not uncommon for those who don't have much experience with relational database to create a database that stores information like in Table A.7.

name	code	surface_area	continent	gnp	population	life_expectancy	government	head_of_state	capital	Languages
Argentina	ARG	2780400	South America	340238	37032000	75.1	Federal Republic	Christina Fernandez	69	Spanish (T, 96.8), Italian (F, 1.7), Indian Languages (F, 0.3)

Table A.7 Design for `CountrySpeak` that is not even in First Normal Form

Fields in tables are not meant to store lists. They are meant to store information about a single, discrete piece of information. Here are a few of the downsides of storing lists in a multi-valued column:

- You might not have anticipated enough space if the list grows too large.
- The basic `INSERT`, `UPDATE`, and `DELETE` statements are not sufficient to manipulate multi-valued columns.
- Web programmers will have to do a lot of string parsing to get information that they need from the list.
- Table name, primary key, and column name do not map to a specific piece of data.

A table is at the second level of normalization, also called Second Normal Form, if and only if it is in First Normal form and the primary key determines all non-key column values. Table A.6 is in First Normal Form, but is not in Second Normal Form because the country code does not determine the value for language, official, or percentage. As discussed above, a table that is not in Second Normal Form is subject to errors on insert, update, and delete.

A table is at the third level of normalization, also called Third Normal Form, if it is in Second Normal Form and all columns are directly dependent on the primary key. A way to remember what Third Normal Form means was given by Bill Kent: every non-key attribute "must provide a fact about the key, the whole key, and nothing but the key so help me Codd" (Codd invented the theoretical basis for relational databases). All tables in the `world` database schema proposed in Table A.5 is in Third Normal Form as all non-key columns depend on the primary key.

An example of a table in Second Normal Form, but not in Third Normal Form would be if we added the head of state's date of birth to the `Countries` table. This is because the date of birth of the head of state relies on the person that is head of state, not on the country. The scenario where this could result in a data inconsistency is if the same person happened to be head of state in two countries at the same time (sounds ridiculous but could be viable if one country invades another), there is nothing to stop the head of state to have two different dates of birth in the two rows. In order to store this additional piece of information and stay in Third Normal Form, we would make a `HeadOfState` table in which we would store the name and date of birth and then `Countries` would link to this table through a foreign key. Table A.8 summarizes the three levels of normalization.

First Normal Form	No duplicate rows and no multi-valued columns (i.e. columns of lists)
Second Normal Form	In First Normal Form and primary key determines all non-key columns
Third Normal Form	In Second Normal Form and all columns are dependent on primary key

Table A.8 Three levels of normalization

Physical Design

At this stage of the database design process, you should have a good idea of what your tables, columns, and keys will be and that the structure of the database is safe from data duplication and inconsistencies. The last step in the design process is to figure out how the database will physically be configured for the hardware on which it runs. This includes choosing the data types of each table