There's quite a bit of formalism to relational databases. These notes gloss over the key ideas.

1 Keys, Superkeys, Primary Keys

A *superkey* for relation R is a set of attributes $S \subseteq R$ such that no two different tuples t_1 and t_2 will have $t_1[S] = t_2[S]$. Essentially a superkey is what makes a tuple different from any other tuple.

A *key* is a *minimal* superkey, in a sense that removing any attribute will cause it not to be a superkey.

Any set of attributes that includes the *key*, is also a *superkey*.

A relation can have more than one *key*. These are then called *candidate keys*.

One of the *candidate keys* is picked to be the *primary key*. Every relation must have a primary key.

1.1 Functional Dependencies

A *functional dependency* is a constraint on the values of attributes across tuples. If $S \rightarrow T$ we say that T is functionaly dependent on S. That means that for any two tuples t_1 and t_2, if we have $t_1[S] = t_2[S]$, then we must also have $t_1[T] = t_2[T]$.

An example might be cities and zip codes. If two records have the same zip code, it can be safely assumed that they have the same city. In this case, the city is functionaly dependent on the zip code.

2 Normalization

There is a concept of 'Normal Forms' that is often[1] used to design and judge the quality of design of databases.

There are five[2] normal forms. They're numbered—conveniently enough—one thorough five. There are also a bunch of other intermediate forms named after something or other (usually whoever came up with it).

Higher numbered normal forms have all the goodness qualities of the lower forms. For example, third-normal form database is also in second-normal form and first normal form. Without putting too fine a point on it, we want our databases to be in as higher normal form as possible. Higher is usually better.

Generally, you're only concerned about the first three normal forms—and in this class, we'll also be concerned with Boyce-Codd Normal Form, which is sort of a "3.5 Normal Form".

2.1 First-Normal Form

This is the most basic normal form, and the only requirement is that data is stored in tables. If your data is stored in tables, then you've achieved first-normal form.

[1] Which is to say *never*.

[2] Or more

More formally, it says that you can only have *atomic* values as attributes. For example, tables can store strings, numbers, dates, etc., but cannot store sets, lists.

If a database has columns that contain comma separated values, it's usually[3] a sign that the database is not normalized.

2.2 Second Normal Form

Ok, here it goes:

> "A database is in second-normal form if it is in first-normal form and every attribute is *fully* functionally dependent on the primary key."

And now to explain it: primary keys are fields that uniquely identify a record. Attributes are everything else in the record.

Now, *functionally dependent* means that given a primary key, we can get the value of any attribute. For example, given your student id, we can find your first name and last name: your first and last name are *functionally dependent* on your student id.

The *fully functionally dependent* mostly applies to composite primary keys. It basically means that the attribute needs to be functionally dependent on the *whole* primary key (not one of its parts). For example, some applications use first name, last name, and date of birth as a composite primary key. Every attribute in that record needs to depend on *all* first name, last name, and date of birth.

2.2.1 Design

It is very easy to achieve second-normal form by simply choosing a unique abstract primary key that doesn't depend on the data. i.e.: adding an auto-increment primary key to your tables.

If the above cannot be done, the usual way to achieve $2NF$ is to break the original relation into many smaller relations with smaller keys with proper functional dependencies.

2.3 Third-Normal Form

> "A database is in third-normal form if it is in second-normal form and contains no transitive attribute dependencies."

This one is similar to $2NF$, except we want to avoid transitive functional dependencies. For example, while $X \to Y$ may not be present (ie: $2NF$), we might still have situations like $X \to Z$ and $Z \to Y$, which would cause the relation not to be in $3NF$.

2.4 Boyce-Codd Normal Form

> "A relation R is in Boyce-Codd normal form if its primary key, K, implies all nonkey attributes—$a, b, c, \ldots$—and K is a superkey."

[3] There are always exceptions when applying these things in real life.

Another way of saying the same thing is:

> "A relation R is in BCNF if whenever a nontrivial functional dependency $X \to A$ holds in R, then X is a superkey of R."

Basically if we pick a new unique id for each record (the auto-increment, etc.) and pick fields 'appropriately' then we'll get Boyce-Codd normal form.

The Boyce-Codd normal form doesn't exactly fit into the numbered normal form idea: it starts with normal form (assumes that data is in tables). In fact, it was originally stated as a simplification of $3NF$, and later found to be a stricter. It has been proved[4] that Boyce-Codd is also in third-normal form. So we can consider Boyce-Codd as a 3.5-normal form.

3 Summary

The key (heh) behind all of the above normal forms is: avoid data duplication.

[4] By someone.