Let’s take a look at the following table:

| **Customer** | **Purchase date** | **Product name** | **Amount** | **Price** | **Total price** |
| --- | --- | --- | --- | --- | --- |
| Joe Smith | 2014-02-14 | Yoga mat | 1 | 80 | 80 |
| Jane Bauer | 2014-02-16 | Yoga block | 2 | 30 | 60 |
| Joe Smith | 2014-02-14 | Yoga block | 2 | 30 | 60 |
| Joe Smith | 2014-02-14 | Yoga strap | 1 | 10 | 10 |
| Thomas Apple | 2014-02-18 | Dumbbells 2kg | 2 | 30 | 60 |
| Jane Bauer | 2014-02-16 | Yoga mat | 1 | 80 | 80 |

What’s wrong with this table? It’s difficult to modify data in it. Upon modification, several anomalies can occur:

**Insert anomalies**

It’s impossible to insert a product into the table if the product hasn’t been bought by a customer yet. Similarly, it’s impossible to insert a customer who hasn’t made a purchase yet.

**Update anomalies**

It’s difficult to update data in the table. If you want to change the name of the product, you have to update all rows where the product is bought. You cannot change the price of the product for all future purchases.

**Delete anomalies**

If you delete the Thomas Apple purchase (say, because the order was cancelled), you will also delete the product “Dumbbells 2kg.”

*how to design tables which have no update anomalies. So here we go!*

Today we begin a series of posts on data normalization. We will talk about **functional dependencies**, a concept that needs to be explained before we dive deeply into database schema normalization.

The subject is rather abstract and theoretical, but I will try to restrain myself from going too deep into mathematics. I will try to keep things simple and down-to-earth. (The operative word being try ;) )

**Simple Example: Person Table**

The data in a table are usually not independent. The values in one column can determine the values in other columns. We’ll explain this using a simple example.

Create Person(

ssn varchar(20),

first\_name varchar(255),

last\_name varchar(255),

date\_of\_birth varchar(255),

address varchar(255),

phone\_number varchar(255))

Example data in the table might be like this.

| **SSN** | **First name** | **Last name** | **Date of birth** | **Address** | **Phone number** |
| --- | --- | --- | --- | --- | --- |
| 123-98-1234 | Cindy | Cry | 15-05-1983 | Los Angeles | 123-456-7891 |
| 121-45-6145 | John | O’Neill | 30-01-1980 | Paris | 568-974-2562 |
| 658-78-2369 | John | Lannoy | 30-01-1980 | Dallas | 963-258-7413 |

Here, the value in the column SSN (Social Security Number) determines the values in columns first\_name, last\_name, date\_of\_birth, address, and phone\_number. This means that if we had two rows with **the same value in the SSN column**, then values in columns first\_name, last\_name, date\_of\_birth, address, and phone\_number would be **equal**. A person with SSN 123-98-1234 is always called Cindy Cry, is born on 15-05-1983, and so on. A situation like this is called **functional dependency**.

The notation for functional dependency:

ssn → first\_name

ssn → last\_name

ssn → date\_of\_birth

ssn → address

ssn → phone\_number

In short, we might write it like this:

ssn → first name, last name, date\_of\_birth, address, phone\_number

On the left-hand side of the arrow we put the name of the column the other is dependent on. On the right-hand side, we put the name of the column that is dependent.

One thing is worth clarifying before we go on: a functional dependency is a constraint in the **database schema**, not in the data. You may notice that in our example all people named John are born on 01-30-2000. But it does not mean that we have a functional dependency: first\_name → date of birth

The constraint is not generally true for all data that might come into our table. It’s only a coincidence in our data. **Coincidences don’t count.** To find a functional dependency you only look for constraints that are generally true, for all possible data.

**Another Simple Example: Car Table**

Create table Car

(vin char(17),

Brand varchar(255),

Model varchar(255),

production\_year int,

price decimal(10,2));

The obvious functional dependency is this:

vin → brand, model, production\_year, price

The column vin determines the brand, model, production\_year, and price columns.

A careful reader might ask: what about a dependency like this:

price → price

Surely the price column determines its own value? You’re right. A dependency like this is called **trivial** functional dependency. You usually omit trivial dependencies if you're writing up all functional dependencies in a table.

**Example: Students Table**

Not all functional dependencies have a single column in the left-hand side. Take a look at the **students**table.

Create table Students

(student varchar(255),

Semester int,

Lecture varchar(255),

teaching\_assistant varchar(255));

And some example data:

| **Student** | **Semester** | **Lecture** | **Teaching assistant** |
| --- | --- | --- | --- |
| Cindy Cry | 6 | Databases 101 | Jack Magpie |
| John Novak | 4 | Databases 101 | Margaret Beettle |
| Allan Smith | 6 | Algorithms | Paul Reason |

The student determines the semester they’re currently at, so we have a functional dependency.

student → semester

But there is one more functional dependency here. Student and lecture **together** determine the name of the teaching assistant tutoring the student. The lecture only is not enough: some lectures have more than one teaching assistant. The student only is not enough either: students attend many different lectures. Without the lecture name we don’t know which class we want to find a teaching assistant for. We note the functional dependency like this:

student, lecture → teaching assistant

And you always have plenty of trivial dependencies:

student, semester → student

student, semester → semester

lecture, semester, teaching assistant → lecture, teaching assistant

### Example: Vertabelo Table

One more example: table with some Vertabelo data. In the table we have the name of the user, the id of the model, the role the user plays in this model (owner, editor, or viewer), and the number of tables in the model.

Create table vertabelo

(user varchar(255),

Model varchar(255),

Role varchar(5),

number\_of\_tables int);

Some functional dependencies we have here:

The number of tables depends only on the model.

model → number of tables

The user role depends on both user and model.

user, model → role

Why do you need all of this normalization stuff? The main goal is to **avoid redundancy** in your data. Redundancy can lead to [various anomalies when you modify your data](http://www.vertabelo.com/blog/update-anomalies). **Every fact should be stored only once** and you should know where to look for each fact. The normalization process brings order to your filing cabinet. You decide to conform to certain rules where each fact is stored.