Jackson Holt

Dr. Moorman

DBMS, CS 3144

13 April 2025

Normal

When presented with an extensive database there are a few steps we should take before being able to use it. The first step would be to sit down and analyze the data, take notes on it and ask clarifying questions. You will develop the ability to ask these questions with time and practice, but here is a general guide on some steps you can take. The best thing to start with is to see which attributes have missing or composite values. Luckily, these are easy to spot. Second, depending on how we answered the first, we should begin to normalize the data into, at the minimum, normal form three. After there is a solid understanding of the data and it is structured into at least third normal form, we can then break down the existing tables into smaller tables to improve the semantic understanding of the database. By analyzing the steps taken with a real data set, we can observe how real and messy data can sometimes have issues when you are trying to work with it.

Real-world datasets are all around us. Examples range from a school registry of students, classes, grades, etc. To big corporations datasets with projects, employees, projects, etc. The dataset we will be observing is a Titanic dataset with the following attributes: Survived, Class, Honorific, First Name, Last Name, Unmarried Name, Age, Siblings/Spouses, Parents/Kids, Ticket Number, Fare, Cabin Number, and Fare. All of these compose our data. Our dataset covers most of the important points of interest of someone who might be on the Titanic.

The first thing to look at is if we are missing any pieces of data within each attribute. As we can observe, there is quite a bit of missing data. So as we start to comb over the data, we can first observe that survived, class, honorific, last name, gender, parent/kid, sibling/spouse, and fare are all attributes that do not have empty columns; the rest do. This might not seem important now, but we will want to pay attention to this when we are determining functional dependencies. Now, we should observe a very important piece of information. The cabin number attribute is composite.

Composite attributes mean our data is not even in first normal form. This means that we have to find a way to separate each cabin into an individual row. If we just did this and duplicated rows with the same information, just with different cabin numbers, while yes this would work, this is redundant. The better way to do this would be to separate the cabin numbers into their own table using a foreign key. Well, that is a grand idea, but let's start with finding some other functional dependencies and assume the cabins do not exist for the time being. That way, when we come back, we might be able to find a natural foreign key or at least not have to add multiple attributes to our table if we do not have to.

So, after observing the data, we have found some possible functional dependencies. The first being, does our ticket number determine our cabin number? Seems simple. I like this idea a lot because it makes since each ticket would correspond to a room or set of rooms. But wait, the one big problem with functional dependencies is that you cannot have a null value on the right side a function dependices. Unfortunately for us, ticket number does have null values. Which knocks this functional dependency out for us. But let's keep in mind that some of these functional dependencies that don’t work out could be important to us later. Next, we can look at whether cabin number determines class. When we are thinking about cabins, it would be telling us the

location you would be staying, and since the nicer rooms are up higher and further away from crew. If your cabin is nicer, it would be higher up on the ship. We know that you would have to be in a higher class. This, unfortunately, does not work out because there are quite a few missing cabins across the data. But again, it is important not to forget the relationships we are perceiving as we go through this process. Now, what if we observed class and embarked together? Would we be able to determine where you embarked from? This would stand to reason because if we know that you bought a ticket for this class and for this length of the trip, we could determine what the cost of that ticket would cost you. This dependency would be really interesting to have because it would allow us to see a pattern and maybe even fill in some of the blanks and give us some prediction power. However, just like the ones before, its left side of the functional dependency has several instances with null values, making it invalid. Another good example of a functional dependency that I thought might work was that last name might determine the number of parents/kids or siblings/spouse relations, observing the data further it is not easy to conlcued someones spouses and siblings based off of name alone and parents and kids can also have different names then their parents or kids. Not being able to use null values is a rather annoying trend that you will find a lot in real-world data. There are several possible ways to get around this problem one way is removing any data that does not contain that field and considering it bad data. I do not prefer this approach and would rather keep as much of my clients' data in their database as possible.

In the efforts to keep as much information as possible still in the database, we must find a solution for normalizing it with no solid functional dependencies and also keeping our database as easy to read as possible. The obvious answer to being able to normalize the database is to create an artificial key. While I know this can be controversial in some circles of database

management, it seems to be an appropriate choice here. Mainly because our database does not possess any unique attributes that do not have null values. Even if we consider combining several attributes, we would have to include so many that it would be almost the entire row.

So, by adding an artificial key, we are able to get into all of the third normal form without having to try very hard. But we still have one big problem. The cabin number is still a composite. Now that we have decided that we are going to use an artificial key, we can also use that as a foreign key to access other tables. So if we break cabins down individually and assign them to each an ID in a separate table, we with the ID being a foreign key and private key we should be good, right? No, because we have to remember that the primary key must be unique, and since the ID will be duplicated over the table several times, we have to make not only the cabin number but also the ID the primary key of our secondary table. This is just to ensure that we follow all of the guidelines of a primary key.

So at this point, we have constructed two tables, one that possesses an ID and a cabin number where the cabin number and ID act as a primary key, and the ID is the foreign key linking us to the other table. This is a simple layout that gets us into the third normal form. At this stage, it does not feel like we have enough for our database to be in the third normal form. And I would agree it does feel off to only have two tables. However, as we can observe, both tables are fully functionally dependent on the whole key and nothing but the key. The other advantage to having an artificial key like this is that no matter what data we run across, we will never have to worry about a weird or outlying circumstance creating a duplicate or null key. So, in a way, we have protected ourselves a little bit more from some ignorant user.

There is one last thing that I think would be really useful to us is to add some semantic understanding to our database. This is a simple step of creating tables that group together data

that is similar or related. So, if you remember some of the functional dependencies ideas we had earlier, we can use some of those perceived relationships to create tables of similar attributes. If we look at the functional dependencies that we used, we can create a table of journey info and passenger info. The passenger info includes first name, last name, unmarried name, honorific, age, gender, sibling and spouse, and parents and kids. Journey info has the attributes: Ticket number, class, embark from, and fare. These split some related attributes into groups that might make it easier to access these things if we only want parts of the data. Obviously we are using the ID as a primary key and foreign key for these tables. We still have an outlying attribute, and that is survival. This we can put in its own table because it is such an important peirce of information it makes since to only be accessed when and if you need it.

This brings our grand total of tables to four when it is all said and done. We have journey info, passenger info, survived, and cabin info. All of these tables are serving some kind of purpose, whether that be for improving the normal form of our database or just for overall semantic understanding. So we can observe that by practcing these steps and critically thinking about a dataset, even real-world and messy data has a way of being organized in a concise and easy-to-understand manner.