ECE356: Database Systems Winter 2018

Lecture 9 — Database Design: Normalization

Jeff Zarnett

Database Design

So far, we learned the syntax to create tables, and how to turn modelling diagrams into tables and all the various bits and pieces about how modelling diagrams work. The modelling diagrams give us some guidance and tell us about wrong ways to represent the data, but we don't yet have enough guidance as to what is correct. If we do a good job the relation will be in a *normal form*. Yes, *a* normal form... there are several.

An example from [**?**] gives us an opportunity to do something wrong and show bad design: if we combine instructor (having the attributes id, name, department, salary) with department (having an id, faculty, budget) then our combined table means one larger relation replaces two smaller ones. This isn't total nonsense because instructors do have a relationship with a department, but is this a good way to model the data? Intuition might tell you no, but think about why. In the combined relation, if there are two instructors, say, Sedra and Smith, both members of department of electrical & computer engineering, in both tuples there will be an entry for the budget of the department. The budget data is duplicated and risks becoming inconsistent.

Our intuition may tell us this is bad, but we would like a way to express it formally. Informally the rule is that each value of department name corresponds to at most one budget. The focal description is a *functional dependency* and it is written $dept\_name \rightarrow budget$. A functional dependency specified that if there is a schema that consists just of the attributes for department name and budget, then the department name attribute could be the primary key [**?**]. Combining instructor and department breaks this rule, because we have duplicate entries for department and therefore this can't work.

The functional dependency shows us that data is duplicated and that indicates that we need to split the combined relation in a process called *decomposition*.

The previous example is very egregiously and obviously wrong, making it simple to identify that there is a problem. In reality, a database will have many more tables and it will be harder to find out what the functional dependencies are, but there is an algorithm for that which will be covered soon enough.

Another example from [**?**] shows us that decomposition on its own is not always good: we should split up things that don't belong together, but it is possible to go overboard. If the relation is *employee(ID, name, street, city, province, postalcode)*, we might think that we can decompose this into two schemas: *employee(ID, name), address(name, street, city, province, postcode)*. Does this work? No – two employees could have the same name and that is likely in the real world, as some names are extremely popular.

If instead of defining the address relation to be based on name, we could use the ID instead, and it would work: *employee(ID, name), address(employeeID, street, city, province, postcode)*. This is an acceptable decomposition and we call it *lossless* because no information is lost. The first attempt at this caused a loss of information (if two employees have the same name, there is the possibility of confusion), so it is called a *lossy* decomposition. We do not perform lossy decompositions.

Normalization