Lecture 10 — Normalization

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

August 11, 2023

Normalization

So far, we learned the syntax to create tables, and how to turn modelling diagrams into tables.

The modelling diagrams give us some guidance and tell us about wrong ways to represent the data.

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.

Normalization Example

Combine instructor (having the attributes id, name, department, salary) with department (having an id, faculty, budget).

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?

Normalization Example

Intuition might tell you no, but think about why.

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.

Normalization Example

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 formal description is a functional dependency.

It is written $dept\_name \rightarrow budget$.

Functional Dependency

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.

Silly Example is Silly

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.

There is an algorithm for that which will be covered soon enough.

Decompose ALL THE THINGS!

The relation is *employee(ID, name, street, city, province, postalcode)*.

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.

Do It Properly At Least

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.

Do It Properly At Least

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.

Atomic Domains and 1NF

Our E-R model allows attributes to be non-atomic, that is to say, divisible.

If a course code is “ECE356”, that attribute is non-atomic because it can be divided into two parts, the “ECE” and “356” components.

If the domain is indivisible units, e.g., an integer, it is atomic.

1NF

A relation *R* is said to be in the first normal form (abbreviated as 1NF) if the domain of all attributes of *R* are atomic.

If some attribute is not atomic, then we could subdivide it to put it into 1NF.

Instead of course being “ECE356” we could split it up into department “ECE” and number “356”.

The primary key could still be composed of those two elements, mind you.

1NF and Varchar

Does this mean that a string (varchar) attribute can never be atomic, as one may take a subset?

No, what really matters is how they are used in the database.

If the application logic requires breaking up the attribute for some reason, then it is non-atomic.

Course ECE356 vs Employee ID AA1234.

Set Valued Attributes

More than this, though, we don't like set-valued attributes.

We don't want the faculty tuple to have a very long varchar attribute called departments which then contains multiple elements, separated by spaces...

For example, Engineering is a faculty, and the list of departments would be ECE, MME, SYDE, etc.

Decomposition with Functional Dependencies

The schema that we create is supposed to model entities and their relationships in the real world.

The real world understanding tells us information about how data should be modelled: students have a student ID number; the student ID number is unique.

If the database does not represent that information in some way, we have done something wrong.

The real world constraints may be represented as a key or as a functional dependency.

Decomposition with Functional Dependencies

The schema is just the data model, and we care if the data in the tables conforms to the constraints.

If the data in the table meets all the real-world constraints, that instance of the relation is called a legal instance.

If all tables in the database are a legal instance of that relation, then we can say the instance of the database is a legal instance of the database schema.

Supergir... Superkey!

A superkey is a subset K of a relation R if, in any legal instance of R, for all pairs, of tuples t_1 and t_2 in the instance of r, if $t_1 \neq t_2$ then $t_1[K] \neq t_2[K]$.

This is to say that no two tuples in a legal instance of the relation may have the same values the subset of attributes K.

Or if you prefer, the attributes in K allow a tuple in the relation to be uniquely identified.

Superkey

Suppose that attributes α and β are part of a relation r.

A functional dependency $\alpha \rightarrow \beta$ is satisfied if for all pairs of tuples t_1 and t_2 such that $t_1[\alpha] = t_2[\alpha]$ it is the case that $t_1[\beta] = t_2[\beta]$.

If this functional dependency holds on every legal instance of the relation, we can say the functional dependency holds on the schema.

Superkey

We can say that K is a superkey of R if the functional dependency $K \rightarrow R$ holds on this instance of the relation.

That means, formally, that if $t_1[K] = t_2[K]$ then it means $t_1 = t_2$ because a superkey identifies a tuple uniquely.

Use of Functional Dependencies

Functional dependencies are used for both design and verification purposes.

We use them in the design process to identify and record the constraints that our database schema should be designed around.

They will be used as a way to decide what entities should be formed and how their relationships to others appear.

And when we have a set of functional dependencies, we can use them to verify if an instance of the database is legal.

If the answer is yes, then we say the set of functional dependencies holds.

Use of Functional Dependencies

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

What can we observe in this data?

Trivial Functional Dependencies

There are also trivial functional dependencies because they are always true.

The functional dependency $A \rightarrow A$ is an example of a trivial dependency.

A formal definition of a trivial relation is that $\alpha \rightarrow \beta$ is trivial if β is a contained within α (in set thinking).

Transitive Functional Dependencies

Functional dependencies can be transitive: if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

For each A there can be only one value of B and for that value of B there is only one corresponding C.

If a set of functional dependencies is F, the notation that includes all of these implied dependencies is F^+, the *closure* of F.

The Boyce-Codd Normal Form, known as BCNF, eliminates all redundancy that can be found using functional dependencies.

A schema *R* is in BCNF with respect to a set of functional dependencies in *F*, if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$ either:

(1) the functional dependency is trivial, or

(2) α is a superkey for schema *R*.

Example Callback

Calling back to the earlier example where we looked at merging instructor and relation, we can see that this is not in BCNF fairly easily.

In this relationship a the department name is not a superkey because we had two entries (Sedra, Smith) that were in the same department

Those had the same budget. A department cannot have two budgets.

There should exist a functional dependency. Check it against BCNF conditions.

BCNF Decomposition

If it is not in BCNF there is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ where α is not a superkey for *R*.

Then we need to split up *R* into two relations: (1) $(\alpha \cup \beta)$ and (2) $(R - (\beta - \alpha))$.

We may need to repeat that process on 1 of the new relations until it is in BCNF.

Dependency Preservation

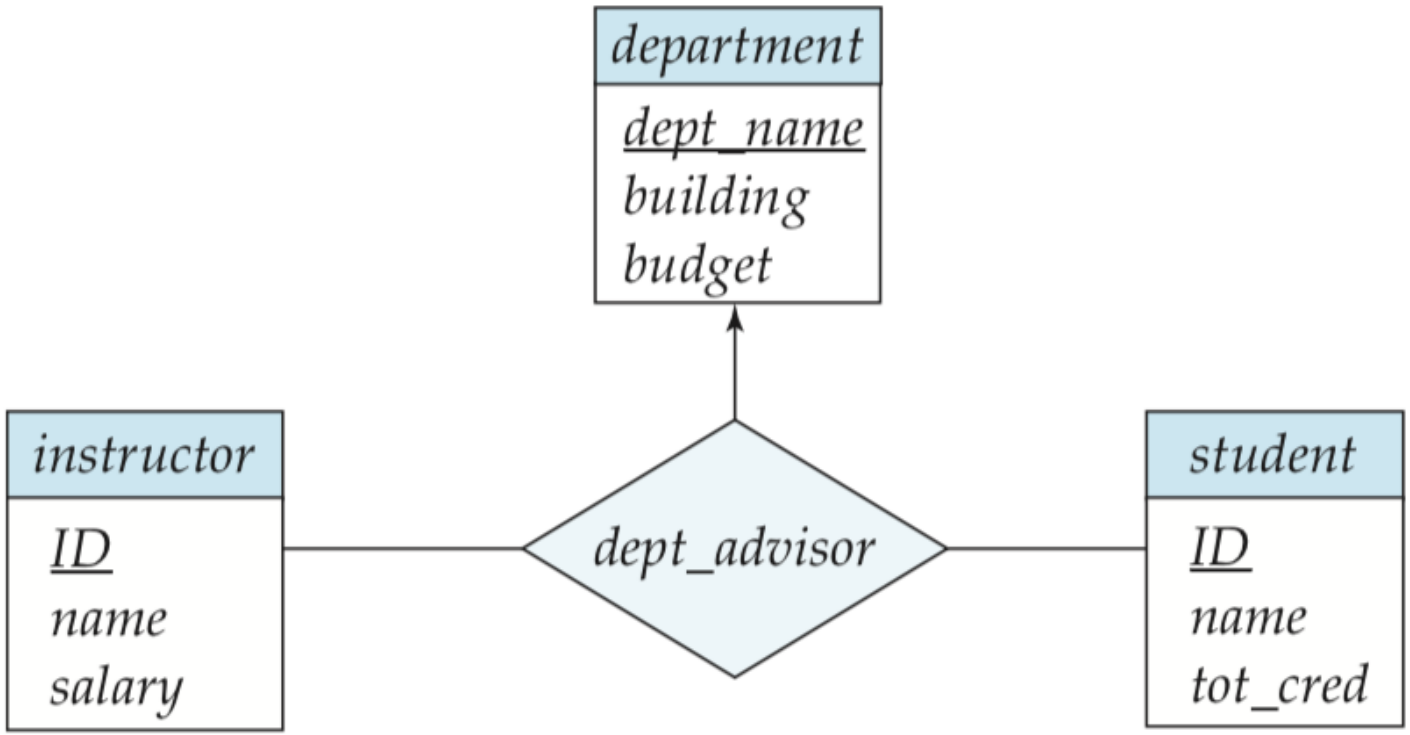
Avoiding duplication of data is not our only goal in a database design.

Constraints also need to be checked.

Joins are expensive and we would like to avoid them.

Checking if a constraint is satisfied may require a computationally expensive join if the relations are in BCNF.

BCNF Example



The functional dependencies are *instructor_id* $\rightarrow$ *dept_name* (an instructor belongs to one department), and *student_id, dept_name* $\rightarrow$ *instructor_id*.

BCNF Example

This design is not in BCNF, because *instructor_id* is not a superkey.

An instructor ID alone is not enough to identify the relationship, because multiple students can be advised by the same individual.

So we need to perform the decomposition procedure and that will produce two relations: *(student_id, instructor_id)* and *(instructor_id, dept_name)*.

BCNF Example

It is difficult to verify the constraint that requires us to verify that each triple of *(student_id, instructor_id, dept_name*) is unique.

That is because performing this check requires a join on the two relations.

Because it is now hard to check this dependency, the design is said to not be dependency preserving.

Data is not lost, just harder to check and verify.

2NF and 3NF

The Third Normal Form, or 3NF, is weaker than BCNF that allows dependencies to be preserved.

Is BCNF the second normal form? No, it is separate from BCNF.

A table is in the second normal form if it is in the first normal form and all non-key columns are dependent on the table's primary key.

The 3NF rules are a small modification of the BCNF rules.

In particular, we would like to allow some nontrivial functional dependencies whose left side (the α) is not a superkey.

3NF Formally

A schema *R* is in 3NF with respect to a set of functional dependencies in *F*, if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$ one of the following holds:

(1) the functional dependency is trivial, or

(2) α is a superkey for schema *R*, or

(3) Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*.

BCNF vs 3NF

Remember that a schema that is in BCNF would satisfy case 1 or 2 for each of the functional dependencies, so anything in BCNF is already in 3NF.

3NF, however, is slightly less restrictive and allows functional dependencies that are not permitted in BCNF, so something in 3NF may or may not be in BCNF.

Example in 3NF

The relation in the diagram in the previous subsection to represent *dept_advisor* contains three attributes.

All primary keys in the other relations: *(student_id, instructor_id, dept_name)*.

Based on what we know of the functional dependencies, this is already in 3NF.

The step we took to do the decomposition to BCNF is unnecessary here if we can accept 3NF.

How do we know the new definition of *dept_advisor* is in 3NF?

Higher Normal Forms

This isn't even my final form! In addition to the first, second, third, and BCNF normal forms there are more... Fourth, fifth, and beyond.

The fourth normal form is built upon the third normal form, so the first requirement for something being in 4NF is that it is in BCNF.

The new requirement is that a record type cannot have more than one multivalued dependency.

A multivalued dependency exists between two attributes if, for each value of the first attribute α there is more than one value of the second attribute β.

Consider the following table representing students. It lists their department and their research areas.

id	**department**	**research_area**
20000001	ECE	Security
20000002	CS	Software
20000003	ECE	Circuits
20000004	SYDE	HCI
20000005	ECE	Networks
20000006	CS	Security

4NF Decomposition Algorithm

The formal 4NF decomposition algorithm looks something like this:

1. Compute D^+
2. Check all relations to see if they are in 4NF.
3. If all relations are in 4NF with respect to the constraints, the algorithm terminates.
4. Otherwise let $\alpha \rightarrow\rightarrow \beta$ be a multivalued dependency in the relation R_i that has two or more multivalued dependencies.
5. Split the relation into two new ones, $(R_i - \beta)$ and (α, β).
6. Go back to step 2.

Fifth Normal Form

The fifth normal form, as described in the MariaDB documentation, is one where you can't make tables any smaller with different keys.

People have also tried to design sixth normal forms...

There's some talk in the literature about domain key normal form but these are mostly of theoretical or academic research interest only.

So we will focus on BCNF and 3NF mostly.