Normalization based on Functional Dependencies (FDs)

- *Normalization* works on relations in a relational database.
- *Normalization* based on FDs has been formalized and the theory has been completely worked out.
- *Normalization* makes it easier to update a database but may degrade query performance – an engineering tradeoff.

Thanks to Lois Delcambre

Keys for a Table (reminder)

The key(s) for a table must have unique values and the key(s) for a table help us understand what the table is "about."

Notice … only one value for non-key attributes (for each key value)

Employee	ssn	name	salary	job-code
	111111111	John Smith	40,000	15
	123456789	Mary Smith	50,000	22
	~~123456789~~	~~Marie Jones~~	~~50,000~~	~~24~~

1. NOT allowed because SSN is key!

2. Only one name (and one salary and one job-code) for each row.

For one particular SSN value, 123-45-6789, there is only ONE name because

1. there is only one tuple and
2. we assume that attributes values are atomic.

Functional Dependencies (FDs) generalize keys

Functional dependencies (FDs) for relational tables are a generalization of the notion of key for a table.

Functional Dependencies

An FD, X → Y, where X is a set of attributes and Y is a set of attributes

It is a statement that if two tuples agree on attributes X they must agree on attributes Y

For each X value there is ONLY one Y value.

Functional Dependencies (from semantics of the application)

Likely functional dependencies:

ssn → *employee-name*

course-number → *course-title*

Unlikely functional dependencies

course-number ↛ *book*

birthdate ↛ *ssn*

Will FDs be enforced?

Consider this table:

Emp(ssn, name, phone, dnum, dept-name)

Suppose there is an FD from *dnum* → *dept-name*

But *ssn* is the key for this table.

What will prevent two names for one dept?

Will this FD be enforced? Let's try it.

Consider this table:

Emp(ssn, name, phone, dnum, dept-name)

Employee	ssn	Name	Phone	Dnum	Dept-name
	111111111	John	555-1234	12	Sales
	222222222	Mary	555-7890	12	Marketing
	…				

Can we put these two rows in this table?

Yes, it doesn't violate the key constraint.

But, the FD from dept to dept-name is violated! We shouldn't haven't two different names for dnum 12!

Functional Dependencies

For an FD

$$X \rightarrow Y$$

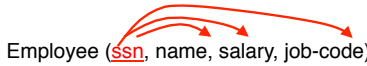
We say that X determines Y

We want to know if it is always true in the application.

We can then use FDs to figure out the keys for tables and to normalize the tables.

Every key implies a set of FDs

Each key implies a set of functional dependencies (FDs) from the key to the non-key attributes.



FDs implied by the key:
ssn → name
ssn → salary
ssn → job-code

But, some FDs are NOT implied by the key.

Emp(ssn, name, phone, dnum, dept-name)

There is an FD from *dnum → dept-name*

The Problem: "Troublesome" FDs

"Troublesome" FDs (FD where the left-hand-side of the FD is NOT a key for the table where its attributes appear) cause redundancy and update anomalies.

Advantages & Disadvantages of Redundancy

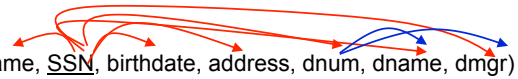
- Disadvantage: Any time information is stored more than once, it has the possibility of being inconsistent.
  - Phone numbers in your laptop
  - Phone numbers in your cell phone
  - Phone numbers in your address book

If someone changes his or her phone number, do you remember to change it in every place?

- Advantage: Redundant information may improve retrieval speed

Sometimes Redundancy is Caused by FDs

Consider this table:



EMP(name, SSN, birthdate, address, dnum, dname, dmgr)

Then *dname* and *dmgr* are stored redundantly – whenever there are multiple employees in a department.

This redundancy is caused by what we informally call "troublesome" FDs. The FDs shown in blue are "troublesome".

Redundancy Caused by Troublesome FD – Sample Data

EMP(name, SSN, birthdate, address, dnum, dname, dmgr)

John	111	June 3	123 St.	D1	sales	222
Sue	222	May 15	455 St.	D1	sales	222
Max	333	Mar. 5	678 St.	D2	research	333
Wei	444	May 2	999 St.	D2	research	333
Tom	555	June 22	888 St.	D2	research	333

We have the department name and manager twice for D1 and three times for D2!

What's wrong?

EMP(name, SSN, birthdate, address, dnum, dname, dmgr)

The name and the manager of the department is repeated, for each employee that works in that department.

Redundancy!

If you replicate information, the copies might be inconsistent.

Update Anomalies
caused by "troublesome" FDs
EMP(name, SSN, birthdate, address, dnum, dname, dmgr)

Insertion anomalies:
if you insert an employee with a department
then you need to know the descriptive information for that department.
if you want to insert a department, you can't ... until there is at least one employee.

Deletion anomalies: if you delete an employee, is that dept. gone? Was this the last employee in that dept.?

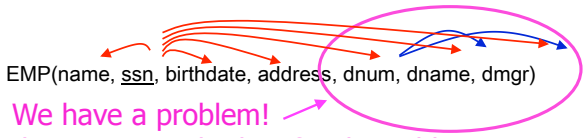
Modification anomalies: If you want to change *dname*, for example, you need to change it everywhere! And you have to find them all first.

Troublesome FDs cause (redundancy and) update anomalies.

The Solution: Lifting "Troublesome" FDs

Normalization by decomposition, based on FDs (where "troublesome" FDs are lifted into a separate table), reduces redundancy and update anomalies.

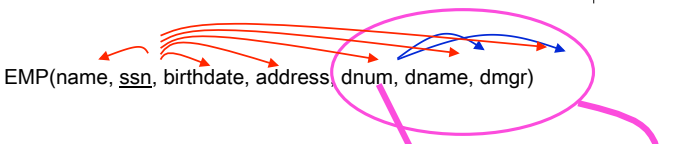
Example: Finding Troublesome FDs



We have a problem!
dnum is NOT the key for this table!

So these blue FDs will not be enforced automatically by the DBMS (using only keys).
And there can be redundancy and update anomalies

Example: Lifting Troublesome FDs



1. Lift the "troublesome" FDs into their own table with dnum as the key. Now they will be enforced.
   Dept(dnum, dname, dmgr)
2. Leave the LHS of the "troublesome" FDs behind. Define a foreign key where New-Emp.dnum REFERENCES Dept.dnum

New-Emp(name, ssn, birthdate, address, dnum)

Table is Split onto New Schemas

New-EMP(name, SSN, birthdate, address, dnum)

John	111	June 3	123 St.	D1
Sue	222	May 15	455 St.	D1
Max	333	Mar. 5	678 St.	D2
Wei	444	May 2	999 St.	D2
Tom	555	June 22	888 St.	D2

Dept(dnum, dname, dmgr)

D1	sales	222
D2	research	333

Less redundancy!

Basic Idea: Normalize based on FDs

- Identify all the (non-trivial) FDs in an application.
  - Identify FDs that are implied by the keys.
  - Identify FDs that are NOT implied by the keys – the "troublesome" ones.
- Decompose a table with a "troublesome" FD into two or more tables by "lifting" each troublesome FD into a table of its own. Note: when there are two or more "troublesome" FDs with the same left side, then they can be lifted, together, into a single table.

Can Define a View to Get Original Table

Emp(name, ssn, birthdate, address, dnum, dname, dmgr)
split into

Dept(dnum, dname, dmgr)

New-Emp(name, ssn, birthdate, address, dnum)

If there are applications that currently query Emp, can define a view:

```
CREATE VIEW Emp AS
  SELECT *
  FROM Dept NATURAL JOIN New-Emp
```

Update statements will require changes in most cases

Advantages of Normalization based on Decomposition

When this table:

Emp(name, ssn, birthdate, address, dnum, dname, dmgr)

is replaced by these two tables:

Dept(dnum, dname, dmgr)

New-Emp(name, ssn, birthdate, address, dnum)

Are there any update anomalies in the new tables?

Let's Check the Update Anomalies

Insertion anomalies:
if you insert an employee with a department
then you need to know the descriptive information for that department. NO – ONLY THE NUMBER
if you want to insert a department, you can't ... until there is at least one employee. NO PROBLEM

Deletion anomalies: if you delete an employee, is that dept. gone? Was this the last employee in that dept.? NO PROBLEM

Modification anomalies: If you want to change *dname*, for example, you need to change it everywhere! And you have to find them all first. dname is only stored once!

Is there any redundancy? Yes – in the foreign key.

2NF, 3NF, BCNF: Normal forms based on FDs

Given a set of FDs and one or more tables, three increasingly stronger normal forms, namely 2NF, 3NF, and BCNF, have been defined.

BCNF implies 3NF.

3NF implies 2NF.

(BCNF = Boyce-Codd Normal Form)

Informal Definitions Normal Forms Based on FDs

1NF - all attribute values (domain values) are atomic (part of the definition of the relational model)

2NF - all non-key attributes must depend on a **whole** key (no partial dependencies)

r (A B C D E) B → C violates 2NF

3NF – table is in 2NF and all non-key attributes must depend on **only** a key (no transitive dependencies)

r (A B C D E) C → D violates 3NF

BCNF – every left side of an FD is a key for the table (All FDs are implied by the keys)

r (A B C D E) C → D violates BCNF (but not 3NF)

Examples of Violations

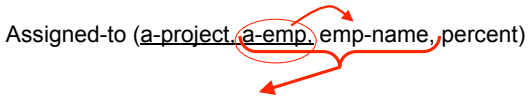
2NF - all non-key attributes must depend on a whole key
Assigned-to (a-project, a-emp, emp-name, percent)

3NF - all non-key attributes must depend on only a key
Employee (SSN, name, address, project, p-title)

BCNF - every determinant (LHS of an FD) is a key for this table (all FDs are implied by the keys) emp-ID → SSN

Assigned-to (emp-ID, a-project, SSN, percent)

Fix violations of Normal Forms by lifting "troublesome" FDs



Employee (a-emp, emp-name)

1. Lift the troublesome FD(s) into a table of their own.
   Key for new table is left hand side of the troublesome FD.

2. Leave the left side of the FD behind in the original table.
   Assigned-to (a-project, a-emp, percent)

3. Eliminate *emp-name* from the *Assigned-to* table.

Formal definition of BCNF

For a table R, every FD X → A that occurs among attributes of R then either:

- A is an element of X (X → A is trivial), or
- X is a superkey of R

For 3NF there is one other option:

- A is part of a key

Decomposition is correct when it is lossless

The decomposition algorithm (based on lifting "troublesome" FDs into a separate table) guarantees that the decomposition of the original table is lossless.

Decompose: Project Operator Recompose: Join Operator

When
Emp(name, SSN, birthdate, address, dnum, dname, dmgr)

is replaced by these two tables:
Department(dnum, dname, dmgr)
NewEmp (name, SSN, birthdate, address, dnum)

We use the project operator to decompose
Department = $\pi_{dnum,dname,dmgr}$Emp
NewEmp = $\pi_{name,SSN,birthdate,address,dnum}$Emp
And we use the join operator to put the pieces together
Emp = Department $\bowtie_{D.dnum=NE.dnum}$ NewEmp

What is a lossless (and a lossy) decomposition?

We want to make sure that we haven't thrown away any information from the original schema.

When table R is decomposed into tables R1 and R2 then the decomposition is lossless (correct) if:

(R1 ⋈ R2) is identical to R — natural join

If it is a lossy decomposition, then R1 ⋈ R2 gives you TOO MANY tuples.

Example: a lossy decomposition

original

Employee(emp-number, name, p-num, p-title)

1	smith	p1	accounting
2	jones	p1	accounting
3	smith	p2	billing

decomposition:

Employee (emp-number, name)

1	smith
2	jones
3	smith

Project (p-num, p-title, name)

p1	account	smith
p1	account	jones
p2	billing	smith

now with natural join: you get at least one extra tuple!

1	smith	p2	billing

One Guarantee for a Lossless Decomposition

Consider a table:

R (a, b, c, d, e) with a troublesome FD d→e.

Decompose it into two tables:

R_1(a, b, c, d)

R_2(d, e)

As long as

the attributes in common are a key for (at least) one of the relations, R_1 or R_2

then we know that the decomposition is lossless.

For this example d is the attribute in common.

And d is a key for R_2, the second table.

Is the Decomposition Algorithm Lossless?

1. Lift the "troublesome" FD(s) (all the FDs with the same LHS) into a table of their own. Key for new table is left hand side of the troublesome FD(s).
2. Leave the left side of the FD behind in the original table.
3. Eliminate the RHS attributes from the original table.

Yes, we are guaranteed that the decomposition is lossless. The attribute in common is definitely a key for the new "lifted" table.

Example of a Lossy Decomposition (revisited)

Employee(emp-number, name, p-num, p-title)

decomposition: Employee (emp-number, name)
Project (p-num, p-title, name)

Notice that the common attribute, name, is not a key for either of these tables.