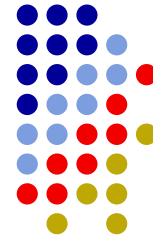


Normalization Strand Map

© Lois Delcambre
Professor
Computer Science Department
Portland State University



1

Normalization based on FDs

- *Normalization* was defined for relations in a relational database (but can be applied in the design of other record or object structures).
- *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.



Strand Map – A Graph of Learning Objectives

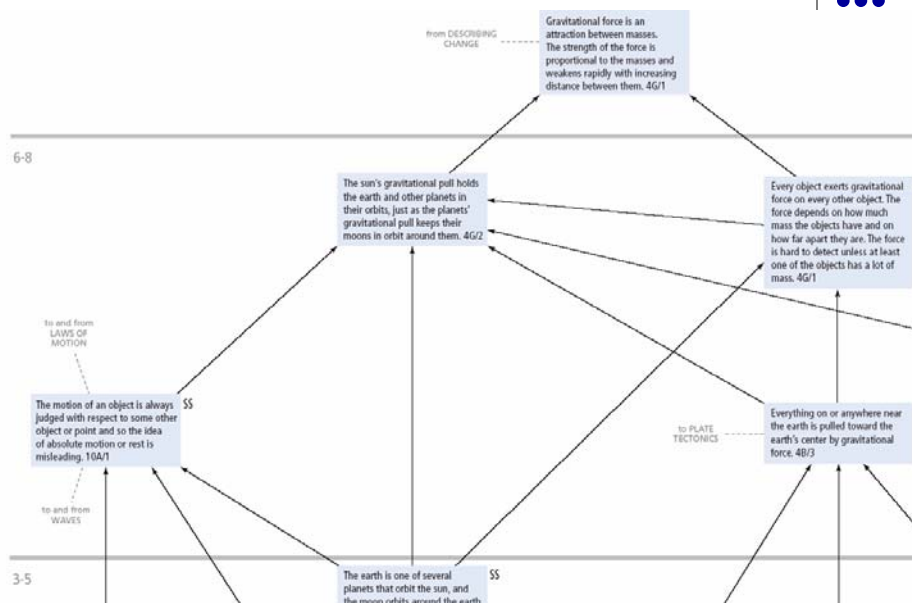


- Developed by the [American Society for the Advancement of Science](#)
- Each box is a “benchmark” or a learning objective
- Connections indicate dependencies: higher-level benchmark above (towards the top of the page), lower-level benchmark below.
- [Sample Strand Map - Gravity \(K-12 Science\)](#)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

3



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

4

Normalization Strand Map



*All computer science students must learn to integrate **theory** and **practice**, to recognize the importance of **abstraction**, and to appreciate the value of **good engineering design**. CC2001 (<http://www.sigcse.org/cc2001/>)*

- This normalization strand map has two strands:
 - the practical concepts and techniques and
 - the formal concepts and results for normalization.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

5

This is an experiment



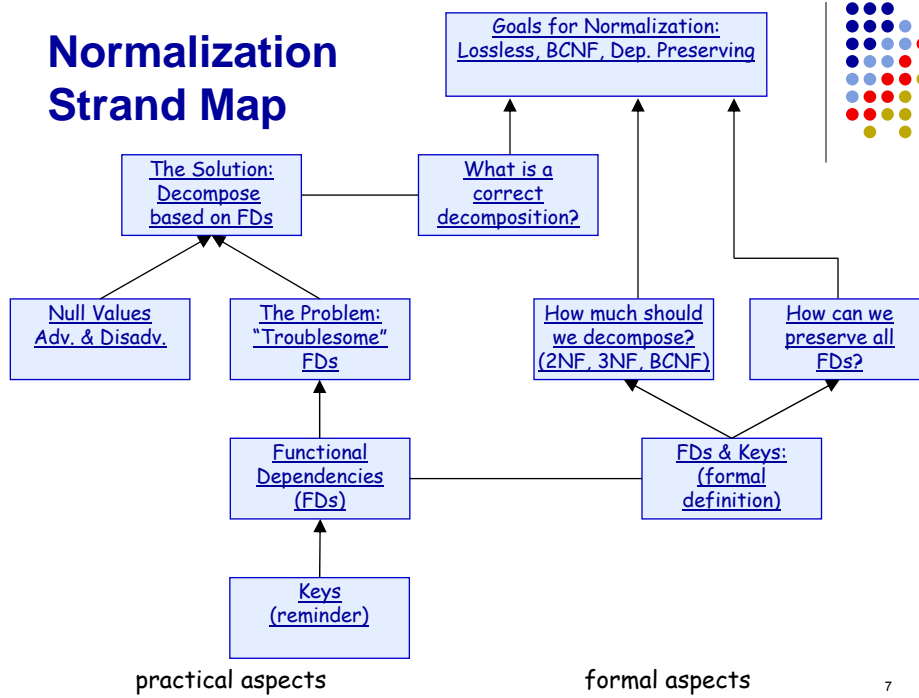
- We are interested in
 - Whether a strand map provides a useful organization for course material.
 - Whether this particular strand map for normalization, with a separation of practical and formal aspects, makes normalization easy to understand.
 - How students like to use the strand map. In particular, we are interested in the order in which students navigate through the strand map.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

6

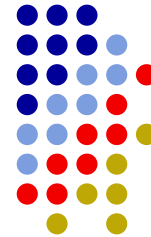
Normalization Strand Map



7

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.”



8

What is the key for each of these tables?



Employee (SSN, Name, Salary, Job-code)

Project (number, title, due-date)

Assigned (ssn, pnum, level-of-effort)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

9

A key serves three purposes:



Employee (SSN, Name, Salary, Job-code)

- The key allows us to access specific records.
- The key tells us what this table represents, what it is “**about**.”
- For one key value (such as one particular SSN), there is only one value for all the other attributes (there is only one name, one salary, and so on). (more later)

[Return to map](#)

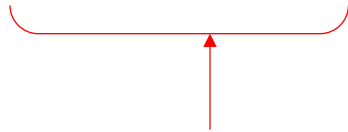
Normalization Strand Map, © Lois Delcambre, 2005-2006.

10

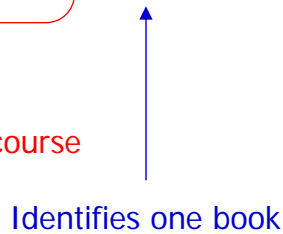
What is the key for this table? What is this table about?



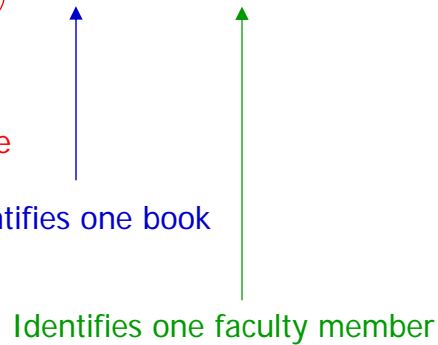
Y(subject, course-num, book-id, faculty-id)



Identifies one course



Identifies one book



Identifies one faculty member

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

11

What is the key for this table?

Y(subject, course-num, book-id, faculty-id)



Option 1:

Y(subject, course-num, book-id, faculty-id)

Table is about
courses!

Option 2:

Y(subject, course-num, book-id, faculty-id)

Table tells
us *who*
teaches which
course!

Option 3:

Y(subject, course-num, book-id, faculty-id)

Table is about
books!

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

12

What is the role of a non-key attribute?



Employee (SSN, Name, Salary, Job-code)

Project (number, title, due-date)

Assigned (ssn, pnum, level-of-effort)

To provide further description of the “thing” or “entity” that this table is about.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

13

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.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

14

If you want each code to have one value, use a table with code as key.



For each subject code, there is only one subject. (These codes are excerpted from the PSU Schedule of Classes.)

For example, the code “G” is used for Geology (and not for Geography for example).

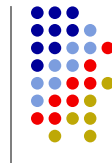
You can always use a table, with the code and description as attributes, with the code as a key, to enforce this situation.

Code	Subject
CS	Computer Science
G	Geology
GEOG	Geography
HST	History
MTH	Mathematics
STAT	Statistics

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

15



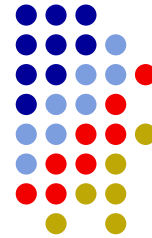
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

16

Functional Dependencies (FDs) generalize keys

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



17

Functional Dependencies



An FD, $A \rightarrow B$, where A is a set of attributes and B is a set of attributes

is a statement that if two tuples agree on attributes A they must agree on attributes B

If the values of the first set of attributes, A , is known, then the value of the second set of attributes, B , is known (i.e., determined).

For one A value there is ONLY one B value.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

18

Functional Dependencies (based on the semantics of the application)



Likely **functional dependencies**:

$ssn \rightarrow employee\text{-}name$

$course\text{-}number \rightarrow course\text{-}title$

Unlikely **functional dependencies**

$course\text{-}number \not\rightarrow book$

$course\text{-}number \not\rightarrow car\text{-}color$

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

19

Will FDs be enforced?



Consider this table:

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

There is an FD from $dept \rightarrow dept\text{-}name$

But ssn is the key for this table.

Is it possible for there to be two names for one dept?

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

20

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



Consider this table:
Emp(ssn, name, phone, dept, dept-name)

Employee	<u>ssn</u>	Name	Phone	Dept	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
have two different names for dept 12!
We might say that *dept* "ought to be a key".

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

21

Functional Dependencies



For an FD

$$A \rightarrow B$$

We say that A determines B

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 also to normalize the tables).

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.


22

Every key implies a set of FDs



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

Employee (SSN, Name, Salary, Job-code)



FDs implied by the key:

SSN → Name

SSN → Salary

SSN → Job-code

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

23

But, there can be other FDs that are NOT implied by the key.



(table repeated from slide 20 and 21):

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



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

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

24

Keys and FDs (notation)



Given a table R, with a and b (together) as a key for the table, the following FDs are implied by the key.

R (a, b, c, d, e) then

$ab \rightarrow c$

$ab \rightarrow d$

$ab \rightarrow e$

Note we also write this as: $ab \rightarrow cde$

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

25

Functional Dependencies can Suggest Keys



If we know these FDs:

$SSN \rightarrow name$

$SSN \rightarrow hire-date$

$SSN \rightarrow phone$

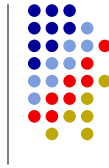
then **SSN** should be the key for a table with these attributes:

Employee (SSN, name, hire-date, phone)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

26



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

27

The Problem: “Troublesome” FDs

“Troublesome” FDs (FDs where the left-hand-side of the FDs are NOT a key for the table where these attributes appear) cause redundancy and update anomalies.



28

Advantages & Disadvantages of Redundancy



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

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

- **Advantage:** **Redundant copies improve retrieval/queries!**

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

29

Sometimes Redundancy is Caused by one or more 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 I informally call “troublesome” FDs. The FDs shown in blue are “troublesome”.

Note: foreign keys always carry redundant values. We can’t get rid of this kind of redundancy.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

30

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!

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

31

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 store information twice (or more) it might be inconsistent! Plus it leads to update anomalies!

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

32

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.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

33

Sometimes redundancy has NOTHING to do with FDs



Suppose we have two tables for employee information.

Employee(ssn, name, salary, birthdate)

Employee2(ssn, name, home-address)

And, we put the name in both tables, for convenience.

Name is stored redundantly.

And, it is possible for the two names to be inconsistent, if you
change it in one place but not in another.

This redundancy is NOT caused by a troublesome FD

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

34



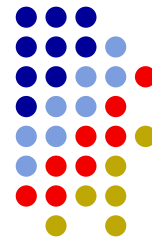
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

35

Null Values: Advantages & Disadvantages

Null values are useful
but they can make it difficult to
understand the semantics of the data
and they can make SQL queries more
complex.



36

Null Values are Useful



- Makes it simpler to insert data, for example
 - before you know the department
 - before you know the spouse name
 - when there is no spouse

For example

Employee(ssn, name, DOB, spouse)

Allowing null values makes the DB more flexible.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

37

Null values cause problems



- may waste space
- may have different meanings:
 - attribute *does not apply* to this tuple
 - attribute value is *unknown*
 - value is *known but absent* (not yet recorded)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

38

Null Values Cause Problems for Aggregate Operators



Employee (ssn, name, salary)

```
SELECT AVG(salary) FROM Employee;
```

```
SELECT SUM(salary) INTO salsum FROM Employee;  
SELECT COUNT(*) INTO total FROM Employee;
```

Salsum/total might be different from first query answer. How could that happen?

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

39

Null Values Complicate Joins



Null values contribute to the need for OUTER joins. And outer joins may be harder to understand (than inner joins).

Student(id, name, advisor) Faculty(ssn, name, office)

where Student.advisor REFERENCES Faculty.ssn

If advisor can be null, then you may wish to use LEFT OUTER JOIN (to find students who do not yet have an advisor).

Null values may cause problems for comparators.

Null values may make queries more complex to write.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

40

Decomposition Reduces the Use of Null Values



Use two tables:

Employee (ssn, name, DOB)

Employee-extra (ssn, spouse)

Rather than:

Employee(ssn, name, DOB, spouse)

Generally, it is better to reduce the use of null values, if you can. The first design, above, doesn't require the use of null values for spouse.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

41



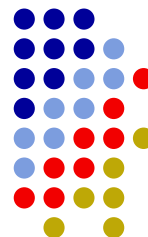
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

42

The Solution: Lifting “Troublesome” FDs

Normalization by decomposition, based on FDs (where “troublesome” FDs are lifted into a separate table), reduces redundancy, the pressure to use null values, and update anomalies.



43

Example: Finding Troublesome FDs



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

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).

We say, informally, that these blue FDs are “troublesome”.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

44

Example: Lifting Troublesome FDs



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

1. Lift the “troublesome” FDs into their own table with dnum as the key. Now they will be enforced.

DEPARTMENT(dnum, dname, dmgr)

2. Leave the LHS of the “troublesome” FDs behind. Define a foreign key where

Employee.dnum REFERENCES Department.dnum

EMPLOYEE(name, ssn, birthdate, address, dnum)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

45

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” the troublesome FDs into a table of their own. Note: when there are two or more “troublesome” FDs with the same LHS, then they can be lifted, together, into a single table.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

46

Decomposition Example



Assigned-to (A-project, A-emp, emp-name, percent)

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.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

47

Decomposition Example



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

Department(dnum, dname, dmgr)

1. Lift the troublesome FD(s) (with the same LHS) 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.

NewEmp (name, ssn, birthdate, address, dnum)

3. Eliminate *dname* and *dmgr* from the *NewEmp* table.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

48

Advantages of Normalization based on Decomposition



When this table:

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

is replaced by these two tables:

`Department(dnum, dname, dmgr)`

`NewEmp (name, ssn, birthdate, address, dnum)`

Are there any update anomalies in the new tables?

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

49

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.**

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

50

Questions about normalization



- How do we know which FDs we have?
Talk to domain experts; identify FDs; use them as the starting point for normalization.
- How do we know if the decomposition is correct?
- How do we know how much to normalize?
How far should we go?
- How do we know if all of the FDs of interest are being enforced – by using keys for a table?
We need the formal definition of FDs to be able to answer these questions.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

51



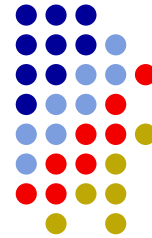
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

52

FDs and Keys: Formal Definition

A functional dependency is formally defined as a functional relationship between two sets of attributes. This leads to the definition of trivial FDs and superkeys.



53

Definition of a function

Remember the definition of a **function**:

x	$f(x)$	x	$g(x)$	x	$h(x)$
1	2	1	2	1	10
1	3	2	2	2	20
2	5	3	5	3	30
3	5				

Which of these are functions?

An FD is a **functional** relationship
(that **must** occur in a relation) among attribute values

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

54

Answer (concerning functions)



Remember the definition of a **function**:

x	$f(x)$	x	$g(x)$	x	$h(x)$
1	2	1	2	1	10
1	3	2	2	2	20
2	5	3	5	3	30
3	5				

f is NOT a function because for an input of “1” there are two answers (“2” and “3”).
 g and h are functions.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

55

Example of an FD – a function



Employee (ssn, name, phone, salary)

Since $ssn \rightarrow name$ is an FD

we know that there is only one name for an ssn.

Thus we know that $ssn \rightarrow name$ is a **function**!

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

56

Another Example



Employee (ssn, name, phone, dept, dept-mgr)



dept → *dept-mgr*

we know that there is only one dept-mgr for a dept.

We know that *dept* → *dept-mgr* is a **function**!

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

57

Trivial FD



We have a trivial FD whenever the attributes on the right side of an FD are a subset of the attributes on the left side of the FD:

$A \rightarrow A$

$AB \rightarrow A$

$ABCD \rightarrow BCD$

A trivial FD represents a function like this: $f(X) = X$

It is definitely a function ... and definitely an FD. But it's not "troublesome" and won't help us decompose a table.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

58

Definition of a Superkey for a relation



A *superkey* is a set of fields from a relation that contains a key.

Every key is (automatically) a superkey.

A superkey is NOT necessarily a key.

Example:

Emp (SSN, name, phone, dept)

SSN is a key for this relation.

(dept, SSN) is a superkey for this relation.

Challenge question: For an arbitrary relation, can you name a superkey?

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

59

Definition of a Key for a Relation



A key is a **minimal** set of attributes in a relation whose values are guaranteed to uniquely identify rows in the relation.

- Two distinct rows have distinct key values
- (minimal) No subset of the fields that comprise a key is a key

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

60

Keys and FDs are Constraints



- We need to know if keys and FDs (always) hold in the application.
- We need to consult a domain expert to find out what the keys and FDs are. The keys and FDs serve as input to the database design process.

That is, we (the DB designers) don't get to decide whether or not an FD or a key holds.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

61



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

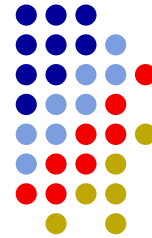
62

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.



63

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)

$r(\underline{A} B C D E) B \rightarrow C$ violates 2NF

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

$r(\underline{A} B C D E)$

BCNF - every **determinant** (LHS of an FD) is a **key** for the table
(All FDs are implied by the keys)

$r(\underline{A} B C D E)$



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)

Assigned-to (Emp-ID, A-Project, SSN, percent)



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

65

Fix violations of 2NF by lifting “troublesome” FDs

(Example repeated from earlier slide)

Assigned-to (A-project, A-emp, emp-name, percent)

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.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

66

Fix violations of 3NF by “lifting” troublesome FDs



(Example repeated from earlier slide)

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

Department(dnum, dname, dmgr)

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.

NewEmp (name, ssn, birthdate, address, dnum)

3. Eliminate *dname* and *dmgr* from the *NewEmp* table.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

67

Fix violations of BCNF by lifting “troublesome” FDs



BCNF - every determinant is a key (all FDs implied by the keys)

Assigned-to (A-emp, A-project, A-ssn, percent)

Take the troublesome FD and put it into a table of it's own.

Assigned-to (A-emp, A-ssn)

Then leave the left side of the troublesome FD in the original table:

Assigned-to (A-emp, A-project, percent)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

68

Formal definition of 3NF (in the textbook)



- For a table R, every FD $X \rightarrow A$ that occurs among attributes of R then either:
 - A is an element of X ($X \rightarrow A$ is trivial)
 - A is part of a key (ignore the “key” attributes)
 - X is a superkey of R
Consider the following 2 cases:
 - X is a key for R (good)
 - X is a superkey for R (and not a key). Then $X \rightarrow A$ is derivable from a key using augmentation. (Stay tuned.)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

69

Formal definition of BCNF (in the textbook)



- For a table R, every FD $X \rightarrow A$ that occurs among attributes of R then either:
 - A is an element of X ($X \rightarrow A$ is trivial)
 - ~~A is part of a key (don't worry about “key” attributes)~~
 - X is a superkey of R
consider the following 2 cases:
 - X is a key for R (good)
 - X is a superkey for R (and not a key). Then $X \rightarrow A$ is derivable from a key using augmentation. (Stay tuned.)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

70

One Goal - BCNF

Whenever we normalize tables, we would like to decompose until all tables are in BCNF.

Then, there are no remaining redundancies (and update anomalies) caused by FDs.

That is, we want all FDs implied by the keys.



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

71



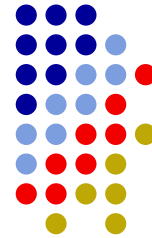
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

72

Dependency Preservation: Using a sound & complete set of inference rules

Dependency preservation requires the use of a sound and complete set of rules of inference to compute F^+ , the closure of a set F of FDs. Given the original set of FDs, F . Then G is the set of FDs that are implied by the keys in the resulting decomposition. Dependency preservation is when $F^+ = G^+$.



73

Example

Consider the following table:

Employee (SSN, name, phone, dept, dept-name)

Original FDs F :

$SSN \rightarrow name$ $SSN \rightarrow phone$ $SSN \rightarrow dept$
 $SSN \rightarrow dept-name$ $dept \rightarrow dept-name$

Employee (SSN, name, phone, dept)

Department (dept, dname)

Resulting FDs G :

$SSN \rightarrow name$ $SSN \rightarrow phone$ $SSN \rightarrow dept$
 $dept \rightarrow dept-name$

What about $SSN \rightarrow dept-name$? Is it lost? Can there be two department names for one SSN?



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

74



Example (cont.)

Employee (SSN, name, phone, dept, dept-name)

Original FDs F:

SSN → name SSN → phone SSN → dept

SSN → dept-name dept → dept-name

Employee (SSN, name, phone, dept)

Department (dept, dname)

Resulting FDs G:

SSN → name SSN → phone SSN → dept

dept → dept-name

What about **SSN → dept-name**? Is it lost? Can there be two department names for one SSN?

NO! It's not lost. One SSN has only one dept. And one dept has only one dept-name. So SSN has only one dept-name.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

75

We need to derive all FDs from a given set of FDs. We need rules.



For sets of attribute X and Y

Reflexivity

If Y is a subset of X, then $XY \rightarrow Y$

As an example, for all attributes, $A \rightarrow A$

examples: name → name, gender → gender

Augmentation

for all FDs, $A \rightarrow B$ then $AC \rightarrow BC$, for all C

As an example, augmentation creates superkeys from keys.

Transitivity

for all FDs, if $A \rightarrow B$ and $B \rightarrow C$

then $A \rightarrow C$

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

76

Use the rules to compute closure



Let F be a set of FDs.

F^+ is the set of all FDs **implied** (or **derivable**) from F
using a sound & complete set of inference rules

Reflexivity: If Y is a set of attrs, X subset of Y , then $X \rightarrow Y$

Augmentation: If $X \rightarrow Y$, and Z is a set of attrs, then $ZA \rightarrow ZB$

Transitivity: If $X \rightarrow Y$, $Y \rightarrow Z$, then $X \rightarrow Z$

Compute F^+ by applying rules until no new FDs arise.

F^+ is called the **closure** of F)

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

77

Definition of Dependency Preserving



Suppose F is the original set of FDs.

Compute F^+ .

G is set of FDs from F^+ that are present in individual relations in G .

Compute G^+ .

If **$F^+ = G^+$**

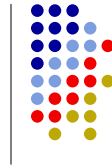
then the decomposition is **dependency preserving**

For a complex design, you may want to implement one of the known algorithms for computing F^+ and G^+ .

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

78



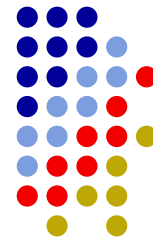
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

79

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**.



80

Decompose: Project Operator Recompose: Join Operator



When

$\text{Emp}(\text{name}, \underline{\text{SSN}}, \text{birthdate}, \text{address}, \text{dnum}, \text{dname}, \text{dmgr})$

is replaced by these two tables:

$\text{Department}(\underline{\text{dnum}}, \text{dname}, \text{dmgr})$

$\text{NewEmp}(\text{name}, \underline{\text{SSN}}, \text{birthdate}, \text{address}, \text{dnum})$

We use the project operator to decompose

$\text{Department} = \pi_{\text{dnum}, \text{dname}, \text{dmgr}} \text{Emp}$

$\text{NewEmp} = \pi_{\text{name}, \text{SSN}, \text{birthdate}, \text{address}, \text{dnum}} \text{Emp}$

And we use the join operator to put the pieces together

$\text{Emp} = \text{Department} \bowtie_{\text{D.SSN}=\text{NewEmp.SSN}} \text{NewEmp}$

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

81

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:

$(R1 \bowtie R2)$ is identical to R natural join

If it is a **lossy** decomposition, then $R1 \bowtie R2$ gives you **TOO MANY** tuples.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

82



Example: a lossy decomposition

original

Employee(SS-number, name, p-num, p-title)				
1	smith	p1	accounting	
2	jones	p1	accounting	
3	smith	p2	billing	

decomposition:

Employee (SS-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

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

83



Example: Test for a Lossless Decomposition

Consider a table:

R(a, b, c, d, e) with a troublesome FD $d \rightarrow e$.

Decompose it into two tables:

R1(a, b, c, d)

R2(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 R2, the second table.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

84

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.

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

85

Example of a Lossy Decomposition (revisited)



Employee(SS-number, name, project, p-title)

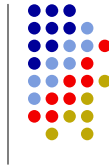
decomposition: Employee (SS-number, name)
Project (p-num, p-title, name)

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

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

86



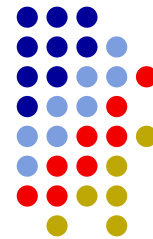
[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

87

Three Goals for Normalization: Lossless, BCNF, Dep. Preserving Decomposition

Given a set of FDs and the original set of relations, the goals for normalization (using lossless decomposition based on FDs) are to have all resulting tables in BCNF and all FDs preserved.



88

Three Goals for Normalization



lossless decomposition
don't throw any information away
be able to reconstruct the original relation

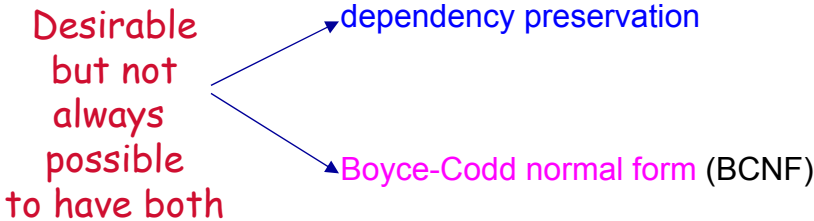
dependency preservation
all of the original, non-trivial FDs can be derived
from FDs implied by the keys of resulting tables

Boyce-Codd normal form (BCNF) - no redundancy
beyond foreign keys; all FDs implied by keys

It is not always possible to have BCNF AND dependency preservation



Required! → **lossless** decomposition



Counterexample

(a table that can't be decomposed into BCNF with dependency preservation)



Original table – a table that holds US addresses

addr(number street city state zip)

The original FDs are:

number street city state -> zip

zip -> state

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

91

Counterexample (cont.)



Based on the FDs:

number street city state -> zip

zip -> state

There are two keys for this table

addr(number street city state zip)

Since all attributes are key attributes, this table is automatically in 3NF and 2NF.

But *zip* → *state* violates BCNF

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

92



Counterexample (cont.)

Let's decompose
 addr(number street city state zip)
 using this "troublesome" FD:
 zip -> state

Addr2 (number, street, city, zip)
 Zip-state (zip, state)

We've lost the FD *number street city state -> zip*

If we put this table back in the design, we are back where we started. And we violate BCNF.

[Return to map](#)

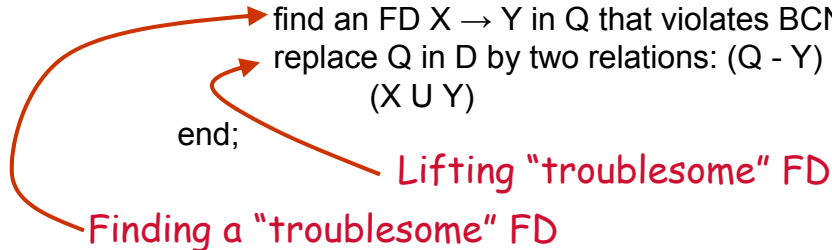
Normalization Strand Map, © Lois Delcambre, 2005-2006.

93

Algorithm for lossless join decomposition into BCNF relations (not necessarily dependency preserving)



1. set $D := \{ R \}$ (the current set of relations)
2. while there is a relation in R that is not in BCNF
 - begin
 - choose a relation Q that is not in BCNF
 - find an FD $X \rightarrow Y$ in Q that violates BCNF
 - replace Q in D by two relations: $(Q - Y)$ and $(X \cup Y)$
 - end;



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

94

Algorithm for dep. preserving, lossless join decomposition into ~~BCNF~~ 3NF relations

(another example of available algorithms)



- dep preserving
- Same as before
1. set $D := \{ R \}$ (the current set of relations)
 2. while there is a relation in R that is not in BCNF
begin
 choose a relation Q that is not in BCNF
 find an FD $X \rightarrow Y$ in Q that violates BCNF
 replace Q in D by two relations: $(Q - Y)$ and $(X \cup Y)$
end;
 3. identify dependencies that are not preserved ($X \rightarrow A$).
 add XA as a table to the set D

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

95

There are a number of other results:

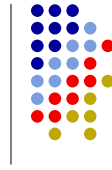


- algorithm to compute F^+
- algorithm to find a minimal cover for a set of FDs
- algorithm for dependency-preserving decomposition into 3NF
- algorithm to synthesize tables, given a set of attributes and a set of FDs

[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

96



[Return to map](#)

Normalization Strand Map, © Lois Delcambre, 2005-2006.

97