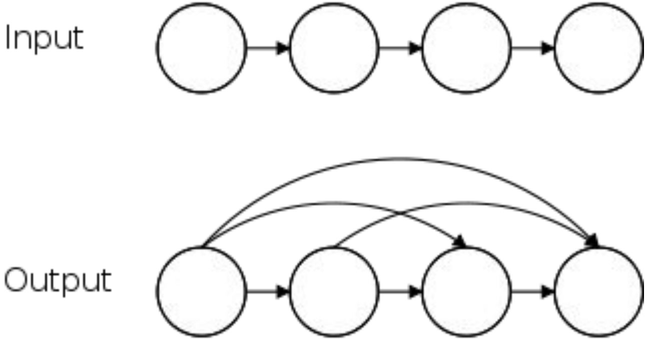
Lecture:
Design Theory

Algebra (reminder)

A => B

B => C, C => D, … X => Y, … (transitive closures)



In this section

1. Normal forms & functional dependencies
2. Finding functional dependencies
3. Closures, superkeys & keys

Example Enrollment table - “v0”

~375 cs145 students

SID	Class	Room	Time	Lat	Lng
4749732	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
2720942	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4823984	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4287594	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2984994	cs 145	Nvidia Aud	T/R 4:30-6	...	...
8472374	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4723663	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2478239	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4763268	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364532	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364573	cs 145	Nvidia Aud	T/R 4:30-6	...	...
3476382	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2347623	cs 145	Nvidia Aud	T/R 4:30-6	...	...
...	...	...	...	...	...
2364579	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
3476343	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
2322232	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W

~300 cs245 students

Problems
Repeats?
Room/time change?
Deletes?

Properties
Class -> Room/time
Room -> Lat, Lng

(more compact)

Example Enrollment table - "v1"

375 cs145 students

300 cs245 students

SID	Class
4749732	cs 145
2720942	cs 145
4823984	cs 145
4287594	cs 145
2984994	cs 145
8472374	cs 145
4723663	cs 145
2478239	cs 145
4763268	cs 145
2364532	cs 145
2364573	cs 145
3476382	cs 145
2347623	cs 145
...	...
2364579	cs 245
3476343	cs 245
2322232	cs 245

Class	Room	Time
cs 145	Nvidia Aud	T/R 4:30-6
cs 245	Nvidia Aud	T/R 3-4:30
cs 246	Nvidia Aud	M/W 3-4:30

Room	Lat	Lng
Nvidia Aud	37.4277° N	122.1742° W

Why Joins? (Recall)

Option 1 (organized tables, with 10s-100s of columns)

Zipcode	Census
94305	
94040	
94041	

Zipcode	Solar
94305	
94040	
94041	

Zipcode	Bikeshare
94305	
94040	
94041	

Zipcode . . .

Option 2 (‘universal table’, with 1000s-millions of columns)

Zipcode	{ Census }	{ Solar }	{ BikeShare }	…….
94305				
94040				
94041				

Option 3 (One table per column, zipcode in each column)

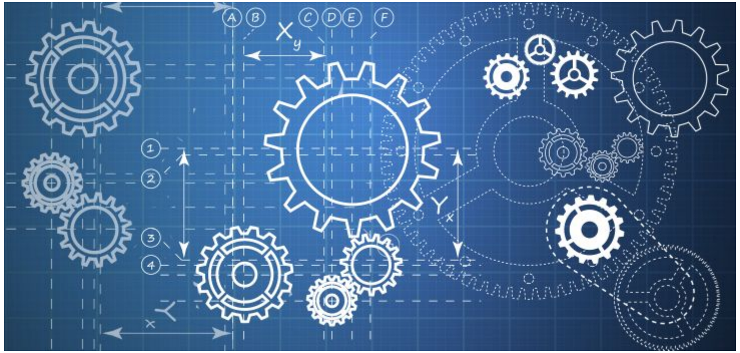
Trade offs?

- Reads? Writes?
- 100s - thousands of applications reading/writing data

Design Theory

- Design theory is about how to represent your data to avoid ***anomalies***.

- Simple algorithms for “best practices”



1. Normal forms & functional dependencies

Normal Forms

- 1st Normal Form (1NF) = All tables are flat
- *2nd Normal Form = disused*
- **Boyce-Codd Normal Form (BCNF)**
- **3rd Normal Form (3NF)**
- *4th and 5th Normal Forms = see text books*

DB designs based on *functional dependencies*, intended to prevent data ***anomalies***

Our focus in this lecture + next one

1^{st} Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…

Violates 1NF.

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1^{st} NF

1NF Constraint: Types must be atomic!

Data Anomalies & Constraints

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

If every course is in only one room, contains ***redundant*** information!

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	C12
Sam	CS145	B01
..	..	..

If we update the room number for one tuple, we get inconsistent data = an ***update anomaly***

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	Course	Room
..	..	..

If everyone drops the class, we lose what room the class is in! = a ***delete* anomaly**

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

...	CS229	C12

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Similarly, we can't reserve a room without students = an ***insert* anomaly**

Constraints Prevent (some) Anomalies in the Data

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Today: develop theory to understand why this design may be better **and** how to find this *decomposition*…

Functional Dependencies

Functional Dependency

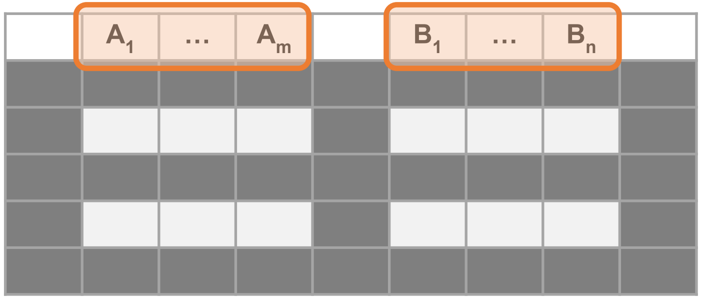
Def: Let A,B be *sets* of attributes
We write A → B or say A ***functionally determines*** B if, for any tuples t_1 and t_2:

$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$

and we call A → B a **<u>functional dependency</u>**

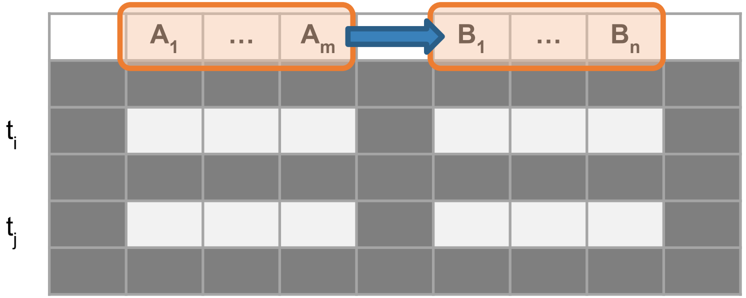
A->B means that
“whenever two tuples agree on A then they agree on B.”

A Picture Of FDs



Defn (again):
Given attribute sets **A={A$_1$,…,A$_m$}** and **B = {B$_1$,…B$_n$}** in **R**,

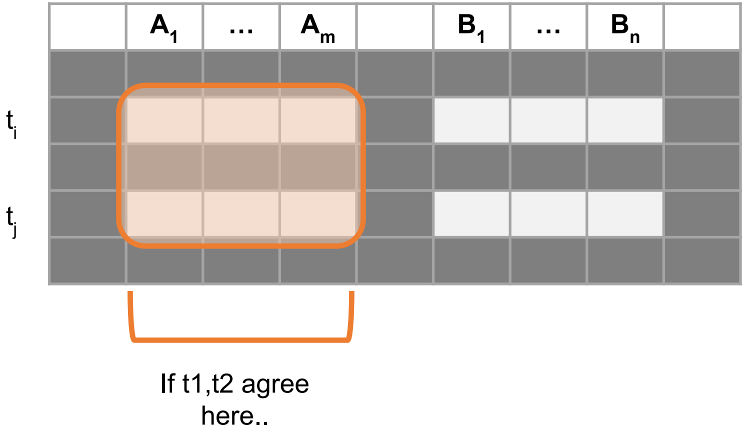
A Picture Of FDs



<u>Defn (again):</u>
Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R,**

The ***functional dependency* A→ B on R** holds if for ***any*** t_i,t_j in R:

A Picture Of FDs

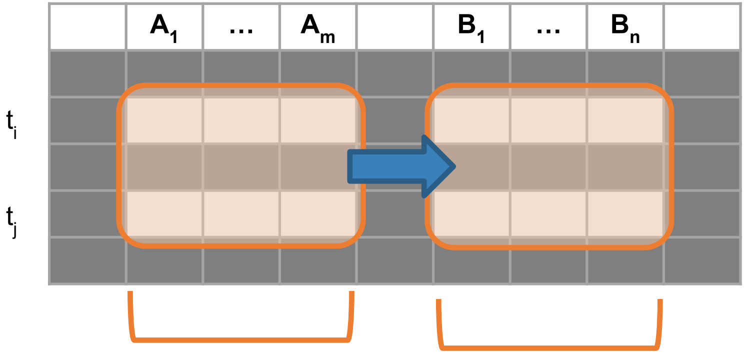


<u>Defn (again):</u>
Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots B_n\}}$ in **R,**

The ***functional dependency* A→ B on R** holds if for ***any*** t_i,t_j in R:

$t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

A Picture Of FDs



If t1,t2 agree here..

…they also agree here!

<u>Defn (again):</u>
Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots B_n\}}$ in **R,**

The ***functional dependency* A→ B on R** holds if for ***any*** t_i,t_j in R:

<u>if</u> $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

<u>then</u> $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

FDs for Relational Schema Design

High-level idea: **why do we care about FDs?**

1. Start with some relational *schema*

2. Find out its *functional dependencies (FDs)*

3. Use these to *design a better schema*
   One which minimizes the possibility of anomalies

Functional Dependencies as Constraints

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Note: The FD {Course} -> {Room} ***holds on this table instance***

However, cannot *prove* that the FD {Course} -> {Room} is ***part of the schema***

Functional Dependencies as Constraints

Note that:

- You can check if an FD is **violated** by examining a single instance;
- However, you **cannot prove** that an FD is part of the schema by examining a single instance.
  - *This would require checking every valid instance*

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

More Examples

An FD is a constraint which holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

More Examples

EmpID	Name	Phone		Position
E0045	Smith	1234		Clerk
E3542	Mike	9876	←	Salesrep
E1111	Smith	9876	←	Salesrep
E9999	Mary	1234		Lawyer

{Position} → {Phone}

More Examples

EmpID	Name	Phone		Position
E0045	Smith	1234	→	Clerk
E3542	Mike	9876		Salesrep
E1111	Smith	9876		Salesrep
E9999	Mary	1234	→	Lawyer

but *not* {Phone} → {Position}

ACTIVITY

A	B	C	D	E
1	2	4	3	6
3	2	5	1	8
1	4	4	5	7
1	2	4	3	6
3	2	5	1	8

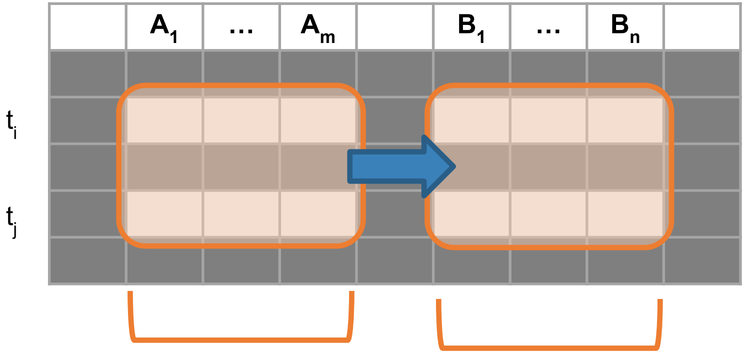
Find at least *three* FDs which are violated on this instance:

$\{ \quad \} \rightarrow \{ \quad \}$
$\{ \quad \} \rightarrow \{ \quad \}$
$\{ \quad \} \rightarrow \{ \quad \}$

2. Finding functional dependencies

A Picture Of FDs [recall]

A→ B on R



If t1,t2 agree here..

…they also agree here!

<u>Defn (again):</u>
Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R,**

The ***functional dependency* A→ B on R** holds if for ***any*** t_i,t_j in R:

<u>**if**</u> $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

<u>**then**</u> $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

Example (mega) Enrollment table - "v0"

~375 cs145 students

~300 cs245 students

SID	Class	Room	Time	Lat	Lng
4749732	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
2720942	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4823984	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4287594	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2984994	cs 145	Nvidia Aud	T/R 4:30-6	...	...
8472374	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4723663	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2478239	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4763268	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364532	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364573	cs 145	Nvidia Aud	T/R 4:30-6	...	...
3476382	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2347623	cs 145	Nvidia Aud	T/R 4:30-6	...	...
...	...	...	...	...	...
2364579	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
3476343	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
2322232	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W

Problems

Repeats?

Room/time change?

Deletes?

FDs

Class -> Room, Time

Room -> Lat, Lng

(more compact)

Table Decomposition

$R(A_1,...,A_n,B_1,...,B_m,C_1,...,C_p)$

$R_1(A_1,...,A_n,B_1,...,B_m)$

$R_2(A_1,...,A_n,C_1,...,C_p)$

R_1 = the *projection* of R on $A_1, ..., A_n, B_1, ..., B_m$

R_2 = the *projection* of R on $A_1, ..., A_n, C_1, ..., C_p$

Conceptual Design

For a “mega” table

- Search for “bad” dependencies
- If any, *keep decomposing the table into sub-tables* until no more bad dependencies
- When done, the database schema is *normalized*

Recall: there are several normal forms…

In this section

1. Finding FDs
   - Closures: How to compute FDs?
   - SuperKeys: One ‘good’ kind of FDs

2. Decomposing mega tables into ‘good’ tables
   - Boyce-Codd Normal Form, 3NF

Finding Functional Dependencies

Example:

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

Given the provided FDs, we can see that {Name, Category} → {Price} must also hold on **any instance**…

Which / how many other FDs do?!?

Finding Functional Dependencies

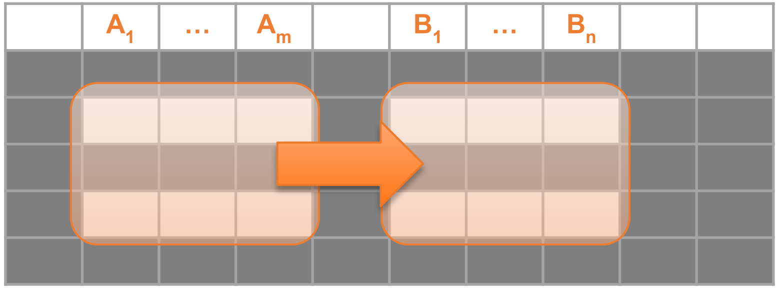
Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

Answer: Three simple rules called **Armstrong's Rules.**

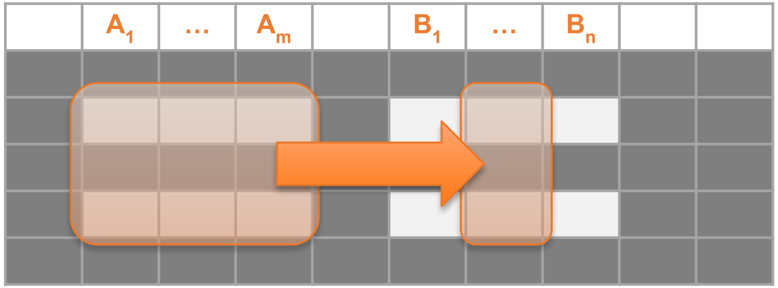
1. **Split/Combine**
2. **Reduction**
3. **Transitivity**

1. Split/Combine



$$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$$

1. Split/Combine

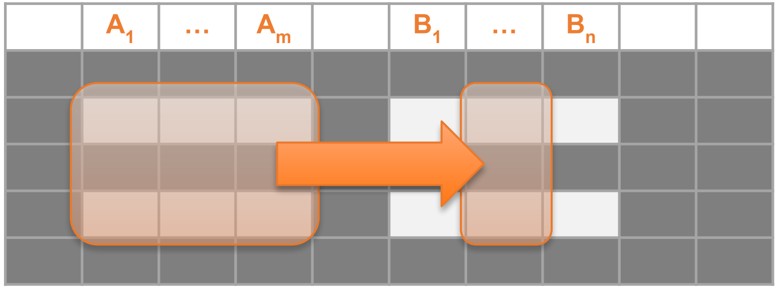


$$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$$

… is equivalent to the following *n* FDs…

$$A_1, \ldots, A_m \rightarrow B_i \text{ for } i=1, \ldots, n$$

1. Split/Combine

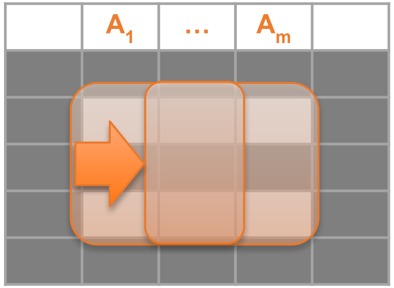


And vice-versa, $A_1,\ldots,A_m \rightarrow B_i$ for $i=1,\ldots,n$

… is equivalent to …

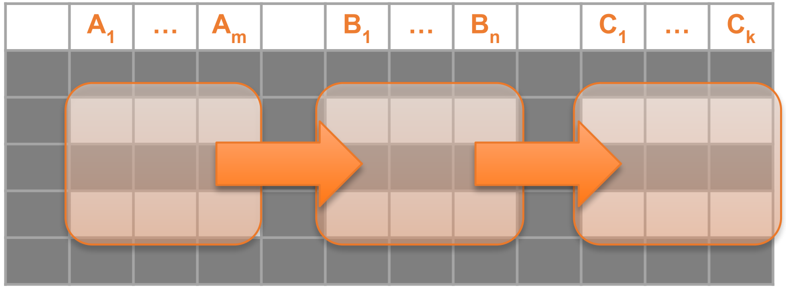
$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

Reduction/Trivial



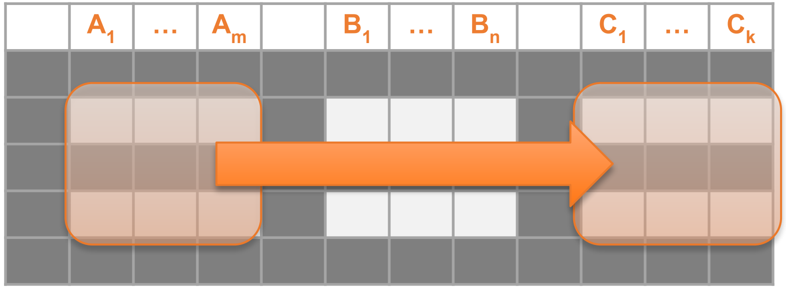
$A_1,\ldots,A_m \rightarrow A_j$ for any $j=1,\ldots,m$

3. Transitive Closure



$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$ and
$B_1,\ldots,B_n \rightarrow C_1,\ldots,C_k$

3. Transitive Closure



$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ and
$B_1, \ldots, B_n \rightarrow C_1, \ldots, C_k$

implies

$A_1, \ldots, A_m \rightarrow C_1, \ldots, C_k$

Finding Functional Dependencies

Example:

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category} -> {Color, Category}	Split/Combine (5 + 6)
8. {Name, Category} -> {Price}	Transitive (7 -> 3)

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

What's an algorithmic way to do this?

Closures & Superkeys

Closure of a set of Attributes

Given a set of attributes **$A_1, \ldots, A_n$** and a set of FDs **F:**

Closure, **$\{A_1, \ldots, A_n\}^+$** is the set of attributes **B** s.t. **$\{A_1, \ldots, A_n\} \rightarrow B$**

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
then add C to X.
Return X as X^+

Example: F =

{name} → {color}
{category} → {department}
{color, category} → {price}

Example Closures:

$\{\text{name}\}^+ = \{\text{name, color}\}$
$\{\text{name, category}\}^+ = \{\text{name, category, color, dept, price}\}$
$\{\text{color}\}^+ = \{\text{color}\}$

Keys and Superkeys

A **superkey** is a set of attributes $\mathbf{A_1, \ldots, A_n}$ s.t.
for *any other* attribute **B** in R,
we have $\{\mathbf{A_1, \ldots, A_n}\} \rightarrow \mathbf{B}$

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

Meaning that no subset of a key is also a superkey

Superkey Algorithm:
For each set of attributes X

1. Compute X^+
2. If X^+ = set of all attributes then X is a **superkey**
3. If X is minimal, then it is a **key**

Example 1

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_n\} \longrightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
then add C to X.
Return X as X^+

$\{name, category\}^+ =$
$\{name, category\}$

F =

$\{name\} \rightarrow \{color\}$

$\{category\} \rightarrow \{dept\}$

$\{color, category\} \rightarrow \{price\}$

Example 1

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

{name} → {color}

{category} → {dept}

{color, category} → {price}

$\{name, category\}^+ = \{name, category\}$

$\{name, category\}^+ = \{name, category, color\}$

Example 1

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_n\} \to C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
then add C to X.
Return X as X^+

F =

{name} → {color}

{category} → {dept}

{color, category} → {price}

$\{name, category\}^+ =$
{name, category}

$\{name, category\}^+ =$
{name, category, color}

$\{name, category\}^+ =$
{name, category, color, dept}

Example 1

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
  if $\{B_1, \ldots, B_n\} \to C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

{name} → {color}

{category} → {dept}

{color, category} → {price}

$\{name, category\}^+ =$
{name, category}

$\{name, category\}^+ =$
{name, category, color}

$\{name, category\}^+ =$
{name, category, color, dept}

$\{name, category\}^+ =$
{name, category, color, dept, price}

Example 2

R(A,B,C,D,E,F)

$\{A,B\} \to \{C\}$
$\{A,D\} \to \{E\}$
$\{B\} \to \{D\}$
$\{A,F\} \to \{B\}$

Compute $\{A,B\}^+ = \{A, B, \qquad \}$

Compute $\{A, F\}^+ = \{A, F, \qquad \}$

Example 2

R(A,B,C,D,E,F)

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

Compute $\{A,B\}^+ = \{A, B, C, D \qquad\qquad \}$

Compute $\{A, F\}^+ = \{A, F, B \qquad\qquad \}$

Example 2

$R(A,B,C,D,E,F)$

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

Compute $\{A,B\}^+ = \{A, B, C, D, E\}$

Compute $\{A, F\}^+ = \{A, B, C, D, E, F\}$

Why Do We Need the Closure?

- With closure we can find all FD's easily

- Is $X \rightarrow A$ true?

  Check if $A \in X^+$

  (i.e., A is in Closure of X)

Note here that **X** is a *set* of attributes, but **A** is a *single* attribute. Why does considering FDs of this form suffice?

Recall the **<u>Split/combine</u>** rule:
$X \rightarrow A_1, \ldots, X \rightarrow A_n$
implies
$X \rightarrow \{A_1, \ldots, A_n\}$

Using Closure to Infer ALL FDs

<u>Example:</u>
Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$
$\{B\}^+ = \{B,D\}$
$\{C\}^+ = \{C\}$
$\{D\}^+ = \{D\}$
$\{A,B\}^+ = \{A,B,C,D\}$
$\{A,C\}^+ = \{A,C\}$
$\{A,D\}^+ = \{A,B,C,D\}$
$\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$, $\{B,C,D\}^+ = \{B,C,D\}$
$\{A,B,C,D\}^+ = \{A,B,C,D\}$

No need to compute all of these- why?

Example of Finding Keys

Product(name, price, category, color)

{name, category} $\rightarrow$ price
{category} $\rightarrow$ color

What is a key?

Example of Keys

Product(name, price, category, color)

$\{name, category\} \rightarrow price$
$\{category\} \rightarrow color$

$\{name, category\}^+ = \{name, price, category, color\}$
= the set of all attributes
$\Rightarrow$ this is a **superkey**
$\Rightarrow$ this is a **key**, since neither name nor category alone is a superkey

In this section

1. Finding FDs
    - Closures: How to compute FDs?
    - SuperKeys: One ‘good’ kind of FDs

2. Decomposing mega tables into ‘good’ tables
    - Boyce-Codd Normal Form, 3NF

Decompositions

Decompositions

$$
\begin{tikzcd}[column sep=small]
& R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p) \arrow[dl] \arrow[dr] & \\
R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) & & R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)
\end{tikzcd}
$$

R_1 = the *projection* of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = the *projection* of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Properties of Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
Gizmo	19.99

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

We need a decomposition to be “correct”

I.e. it is a **Lossless decomposition**

Lossy Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Need to avoid "bad" decompositions

What's wrong here?

Lossy Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

$\bowtie$

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera
OneClick	19.99	Camera
Gizmo	24.99	Camera

Lossless Decompositions

$$
\begin{tikzcd}
& R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p) \arrow[dl] \arrow[dr] & \\
R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) & & R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)
\end{tikzcd}
$$

A decomposition R to (R1, R2) is **lossless** if R = R1 ⋈ R2

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF)

Main idea: define "good" and "bad" FDs as follows:

- X → A is a "*good FD" if X is a (super) key*
  I.e., A is the set of all attributes
- Else, X → A is a *"bad FD"*
  I.e., X functionally determines *some* attributes; other attributes can be duplicate/anomalies

- We will try to eliminate the "bad" FDs!

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is **in BCNF** if:

if $\{\mathbf{A_1, ..., A_n}\} \rightarrow \mathbf{B}$ is a *non-trivial* FD in R

then $\{\mathbf{A_1, ..., A_n}\}$ **is a superkey** for R

In other words: there are no “bad” FDs

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

$\{SSN\} \rightarrow \{Name, City\}$

This FD is *bad* because it is **not** a superkey

$\Longrightarrow$ **Not** in BCNF

What is the key?
{SSN, PhoneNumber}

Example decomposition

<u>Name</u>	<u>SSN</u>	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

<u>SSN</u>	<u>PhoneNumber</u>
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

{SSN} → {Name,City}

This FD is now *good* because it is the key

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find *a set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

Find a set of attributes X which has non-trivial "bad" FDs, i.e. is not a superkey, using closures

BCNF Decomposition Algorithm

BCNFDecomp(R):
   Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

   if (not found) **then** **Return** R

If no “bad” FDs found, in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):
Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then** **Return** R

decompose R into $\mathbf{R_1(X^+)}$ and $\mathbf{R_2(X \cup Rest)}$

R2: Rest of attributes not in X^+

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) **then** **Return** R

  decompose R into $\mathbf{R_1(X^+)}$ and $\mathbf{R_2(X \cup Rest)}$

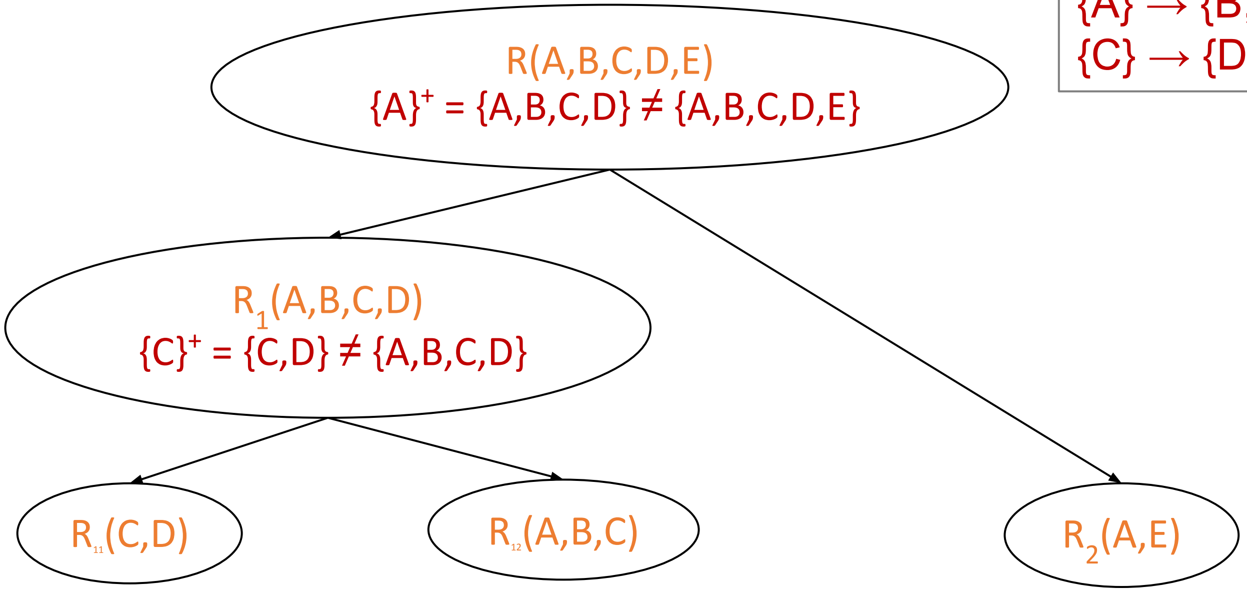
  Return BCNFDecomp(R_1), BCNFDecomp(R_2)

Proceed recursively until no more “bad” FDs!

Example

R(A,B,C,D,E)

$\{A\} \rightarrow \{B,C\}$
$\{C\} \rightarrow \{D\}$



Conceptual Design (recap)

For a “mega” table

- Search for “bad” dependencies
- If any, *keep decomposing (lossless) the table into sub-tables* until no more bad dependencies
- When done, the database schema is ***normalized***

Example Enrollment table - “v0”

~375 cs145 students

~300 cs245 students

SID	Class	Room	Time	Lat	Lng
4749732	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
2720942	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4823984	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4287594	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2984994	cs 145	Nvidia Aud	T/R 4:30-6	...	...
8472374	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4723663	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2478239	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4763268	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364532	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364573	cs 145	Nvidia Aud	T/R 4:30-6	...	...
3476382	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2347623	cs 145	Nvidia Aud	T/R 4:30-6	...	...
...	...	...	...	...	...
2364579	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
3476343	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
2322232	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W

FDs
Class -> Room, Time
Room -> Lat, Lng

(more compact)

Example Enrollment table - "v0"

BCNF decomposition

SID	Class	Room	Time	Lat	Lng
4749732	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
2720942	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4823984	cs 145	Nvidia Aud	T/R 4:30-6	37.4277° N	122.1742° W
4287594	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2984994	cs 145	Nvidia Aud	T/R 4:30-6	...	...
8472374	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4723663	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2478239	cs 145	Nvidia Aud	T/R 4:30-6	...	...
4763268	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364532	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2364573	cs 145	Nvidia Aud	T/R 4:30-6	...	...
3476382	cs 145	Nvidia Aud	T/R 4:30-6	...	...
2347623	cs 145	Nvidia Aud	T/R 4:30-6	...	...
...	...	...	...	...	...
2364579	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
3476343	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W
2322232	cs 245	Nvidia Aud	T/R 3-4:30	37.4277° N	122.1742° W

Schema: SID, Class, Room, Time, Lat, Lng

FDs

Class -> Room,Time
Room -> Lat, Lng

BCNF decomposition

1. Find bad FD #1: Class^+ -> Class, Room,Time, Lat, Lng
   Decomposed: R1(Class, Room, Time, Lat, Lng) and R2(SID, Class)
2. Find bad FD #2: Room^+ -> Room, Lat, Lng
   Decompose R1 into R11(Room, Lat, Lng) and R12(Class, Room, Time)

⇒ BCNF schema: R2(SID, Class), R12(Class, Room, Time), R11(Room, Lat, Lng)

Example Enrollment table - “v1”

375 cs145 students

300 cs245 students

SID	Class
4749732	cs 145
2720942	cs 145
4823984	cs 145
4287594	cs 145
2984994	cs 145
8472374	cs 145
4723663	cs 145
2478239	cs 145
4763268	cs 145
2364532	cs 145
2364573	cs 145
3476382	cs 145
2347623	cs 145
...	...
2364579	cs 245
3476343	cs 245
2322232	cs 245

Class	Room	Time
cs 145	Nvidia Aud	T/R 4:30-6
cs 245	Nvidia Aud	T/R 3-4:30
cs 246	Nvidia Aud	M/W 3-4:30

Room	Lat	Lng
Nvidia Aud	37.4277° N	122.1742° W

A Problem with BCNF

Unit	Company	Product
…	…	…

{Unit} → {Company}
{Company,Product} → {Unit}

Unit	Company
…	…

Unit	Product
…	…

{Unit} → {Company}

We do a BCNF decomposition on a “bad” FD:
$\{Unit\}^+ = \{Unit, Company\}$

We lose the FD {Company,Product} → {Unit}!!

So Why is that a Problem?

<u>Unit</u>	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

No problem so far. All *local* FD's are satisfied.

$\{\text{Unit}\} \rightarrow \{\text{Company}\}$

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

Let's put all the data back into a single table again:

Violates the FD $\{\text{Company,Product}\} \rightarrow \{\text{Unit}\}$!!

The Problem

- We started with a table R and FDs F
- We decomposed R into BCNF tables R_1, R_2, …
  with their own FDs F_1, F_2, …
- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD **across** tables!

<u>Practical Problem</u>: To enforce FD, must reconstruct R—*on each insert!*

Possible Solutions

- Various ways to handle so that decompositions are all lossless / no FDs lost

- Usually a tradeoff between redundancy / data anomalies and FD preservation…

BCNF vs 3NF

BCNF (recap)

- X → A is a "*good FD" if X is a (super) key*

  I.e., A is the set of all attributes

3NF:

- X → A is a "*good FD" if X is a (super) key*
- Or, if A is part of any key

BCNF still most common- with additional steps to keep track of lost FDs…

Summary

- Constraints allow one to reason about **redundancy** in the data
- Normal forms describe how to **remove** this redundancy by **decomposing** relations
  - Elegant—by representing data appropriately certain errors are essentially impossible
  - For FDs, BCNF is the normal form.
- A tradeoff for insert performance: 3NF