Lectures 5 & 6: Design Theory

Reminders

1. Project #1 due Friday
2. Gradiance homeworks

Algebra (preview)

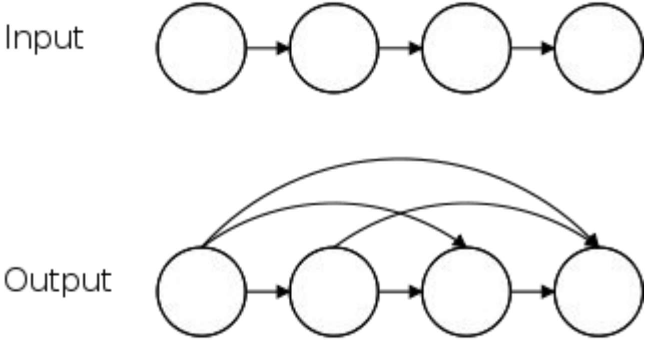
R * S	JOIN
+, -, … ∀ , ∃	Union, intersect...
f (g (x))	Nesting
∑ #	SUM, Count, . . .

A => B

Algebra (reminder)

A => B

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



Today's Lecture

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

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

<u>Problems</u>
Repeats?
Room/time change?
Deletes?

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

(more compact)

Example
Franken tables

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				

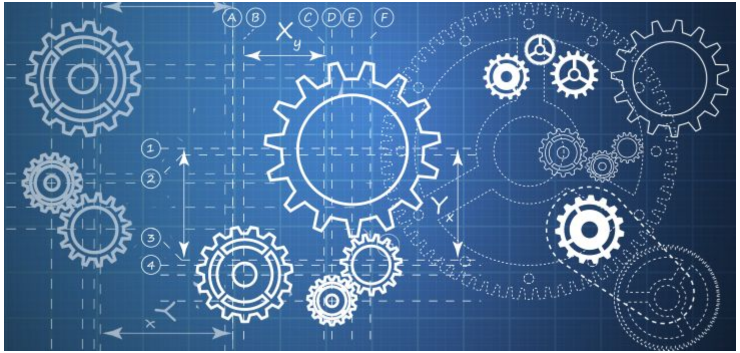
Option 3 (One table per column, zipcode in each column)

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)**

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

- *4th and 5th Normal Forms = see text books*

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 ***<u>redundant</u>*** 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

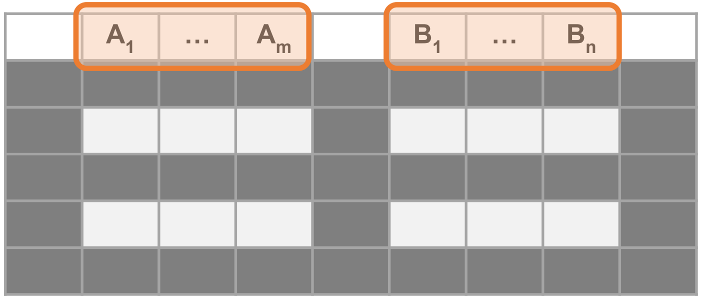
Def: Let A,B be *sets* of attributes
We write A → B or say A ***functionally determines*** B if, for any tuples t_1 and t_2:

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

and we call A → B a **functional dependency**

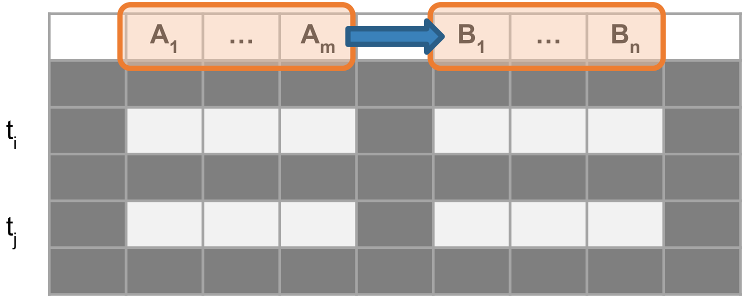
A->B means that
"whenever two tuples agree on A then they agree on B."

A Picture Of FDs



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

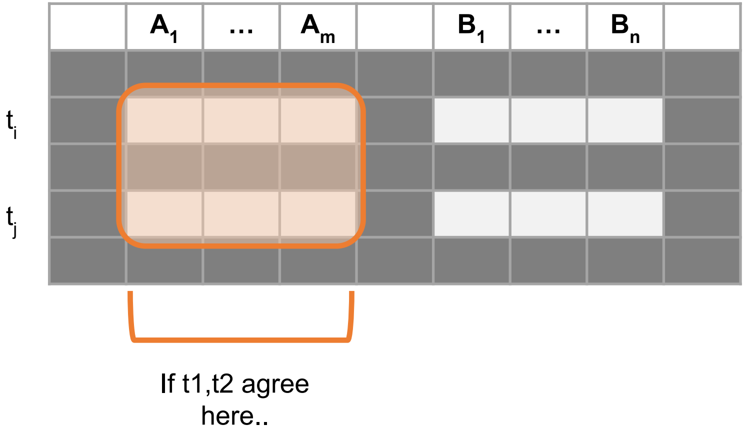
A Picture Of FDs



<u>Defn (again):</u>
Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ 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:

A Picture Of FDs

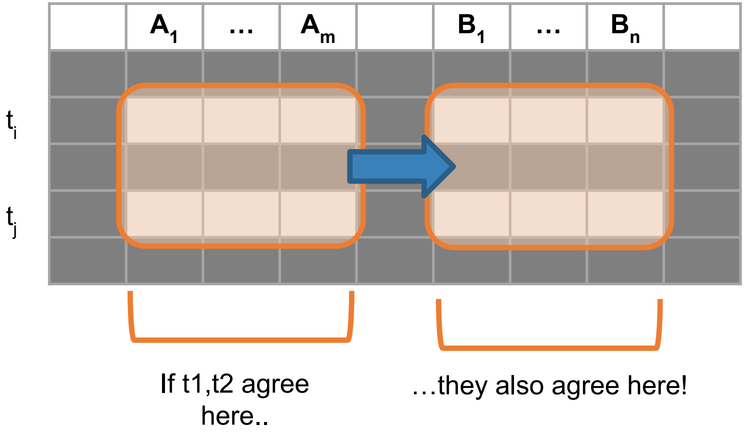


<u>Defn (again):</u>
Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R**,

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

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

A Picture Of FDs



<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*** $\mathbf{A \rightarrow B}$ **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 instance***

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

*Recall: an **instance** of a schema is a multiset of tuples conforming to that schema, **i.e. a table***

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 <u>holds</u>, or <u>does not hold</u> 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:

$\{\quad\} \to \{\quad\}$
$\{\quad\} \to \{\quad\}$
$\{\quad\} \to \{\quad\}$

2. Finding functional dependencies

What you will learn about in this section

1. “Good” vs. “Bad” FDs: Intuition
2. Finding FDs
3. Closures
4. ACTIVITY: Compute the closures

“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

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

<u>Intuitively:</u>

EmpID -> Name, Phone, Position is *“good FD”*

- ***Minimal redundancy, less possibility of anomalies***

"Good" vs. "Bad" FDs

We can start to develop a notion of **good** vs. **bad** FDs:

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

<u>Intuitively:</u>

EmpID -> Name, Phone, Position is *"good FD"*

But Position -> Phone *is a "bad FD"*

- ***Redundancy! Possibility of data anomalies***

“Good” vs. “Bad” FDs

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

Returning to our original example… can you see how the “bad FD” {Course} -> {Room} could lead to an:

- Update Anomaly
- Insert Anomaly
- Delete Anomaly
- …

Given a set of FDs (from user) our goal is to:

1. **Find all FDs, and**
2. **Eliminate the “Bad Ones".**

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)*

   This part can be tricky!
3. Use these to *design a better schema*
   1. One which minimizes possibility of anomalies

Finding Functional Dependencies

- There can be a very **large number** of FDs…
  - *How to find them all efficiently?*

- We can’t necessarily show that any FD will hold **on all instances…**
  - *How to do this?*

We will start with this problem:
Given a set of FDs, F, what other FDs ***must*** hold?

Finding Functional Dependencies

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

Inference problem: How do we decide?

Finding Functional Dependencies

<u>Example:</u>

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

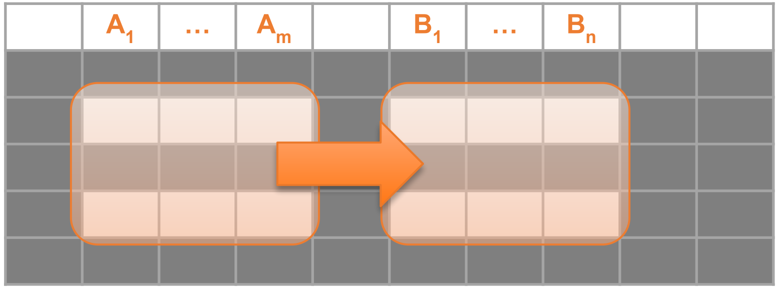
Equivalent to asking: Given a set of FDs, F = $\{f_1,\ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

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

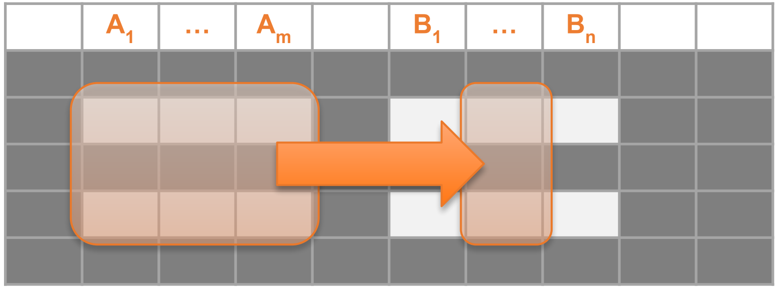
1. **Split/Combine**
2. **Reduction**
3. **Transitivity**

1. Split/Combine



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

1. Split/Combine

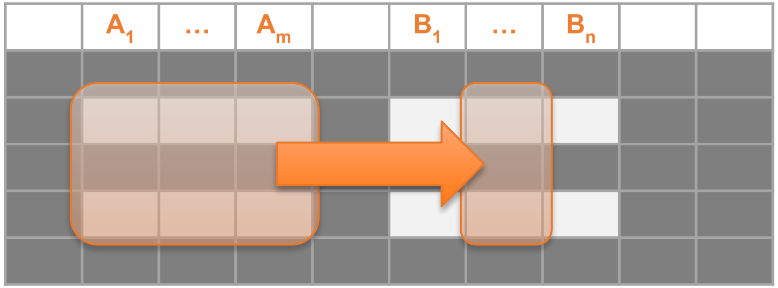


$$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$$

… is equivalent to the following *n* FDs…

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

1. Split/Combine

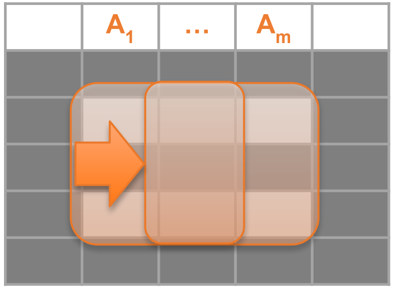


And vice-versa, $A_1,\ldots,A_m \rightarrow B_i$ for $i=1,\ldots,n$

… is equivalent to …

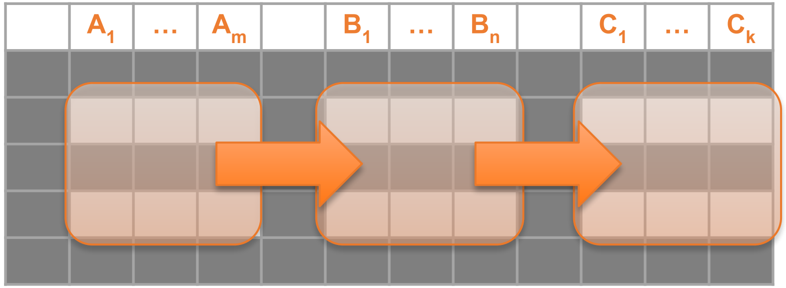
$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

Reduction/Trivial



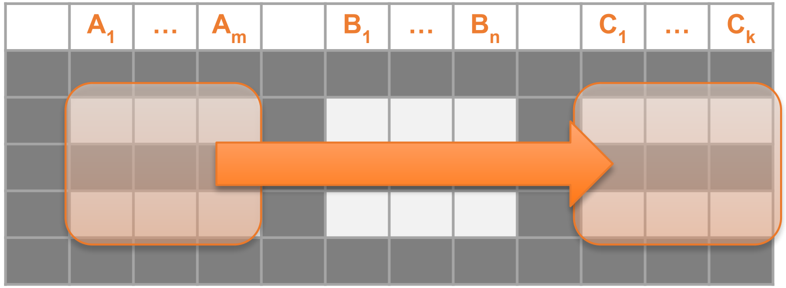
$A_1,\ldots,A_m \rightarrow A_j$ for any $j=1,\ldots,m$

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 FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	?
5. {Name, Category} -> {Color}	?
6. {Name, Category} -> {Category}	?
7. {Name, Category -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Provided FDs:

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

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

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

Closure of a set of Attributes

Given a set of attributes **$A_1, \ldots, A_n$** and a set of FDs **F:**
Then the **closure**, **$\{A_1, \ldots, A_n\}^+$** is the set of attributes **B** s.t. **$\{A_1, \ldots, A_n\} \rightarrow$ B**

Example: F =

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

Example Closures:

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

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change; **do**:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is entailed by F

and $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X.

Return X as X^+

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^+

F =

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

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

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

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

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^+

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

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

F =

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

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

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

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^+

F =

{name} → {color}

{category} → {dept}

{color, category} → {price}

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

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

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

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^+

F =

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

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

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

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

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

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

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

Example

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

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

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

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

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

Example

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

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

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

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

3. Closures, Superkeys & Keys

What you will learn about in this section

1. Closures Pt. II
2. Superkeys & Keys
3. ACTIVITY: The key or a key?

Why Do We Need the Closure?

- With closure we can find all FD's easily

- To check if $X \rightarrow A$

  Check if $A \in X^+$

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 **Split/combine** rule:
$X \rightarrow A_1, \ldots, X \rightarrow A_n$
implies
$X \rightarrow \{A_1, \ldots, A_n\}$

Using Closure to Infer ALL FDs

Example:
Given F =

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

Step 1: 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?

Using Closure to Infer ALL FDs

Example:

Given F =

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

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \varnothing$:

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

Using Closure to Infer ALL FDs

Example:
Given F =

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

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $\boxed{Y \subseteq X^+}$ and $X \cap Y = \varnothing$:

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

"Y is in the closure of X"

Using Closure to Infer ALL FDs

Example:

Given F =

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

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs $X \to Y$, s.t. $Y \subseteq X^+$ and $\boxed{X \cap Y = \varnothing}$:

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

The FD $X \to Y$ is non-trivial

Superkeys and Keys

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

This means that no subset of a key is also a superkey (i.e., dropping any attribute from the key makes it no longer a superkey)

Finding Keys and Superkeys

- 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 of Finding Keys

Product(name, price, category, color)

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

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

Lecture 6:
Design Theory II

Example (mega) Enrollment table - "v0"

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

~375 cs145 students

~300 cs245 students

Problems
Repeats?
Room/time change?
Deletes?

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

(more compact)

Today's Lecture

1. Conceptual design

2. Decomposing tables

3. Boyce-Codd Normal Form, 3NF

3. MVDs

Conceptual Design

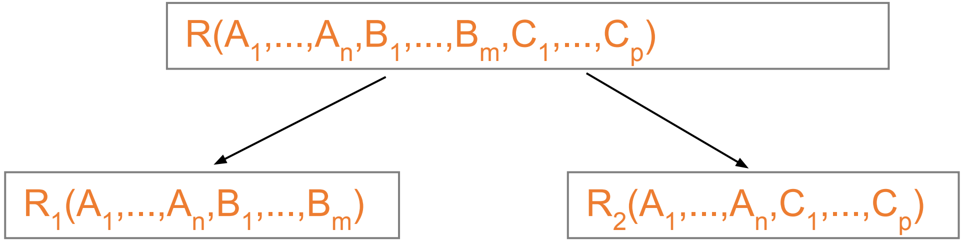
For a “mega” table

- 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…

Decompositions

Decompositions



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

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

Theory 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

Sometimes a decomposition is "correct"

I.e. it is a **<u>Lossless decomposition</u>**

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

However sometimes it isn't

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

⋈

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

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

A decomposition R to (R1, R2) is **lossless** if $R = R1 \bowtie R2$

Lossless Decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

If $\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}$
Then the decomposition is lossless

Note: don't need
$\{A_1, \ldots, A_n\} \rightarrow \{C_1, \ldots, C_p\}$

Conceptual Design

For a “mega” table

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

Recall: there are several normal forms…

Recap: FDs, keys, closures

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

If X^+ = set of all attributes, then X is a **superkey**

If X is minimal, then it is a **key**

Armstrong's Rules

1. **Split/Combine (rhs of FD)**
2. **Reduction**
3. **Transitivity**

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF)

Main idea: define “good” and “bad” FDs as follows:

- $X \rightarrow A$ is a “*good FD” if X is a (super) key*
  I.e., A is the set of all attributes
- Else, $X \rightarrow A$ is a *“bad FD”*

- We will try to eliminate the “bad” FDs!

Boyce-Codd Normal Form (BCNF)

Why does this definition of “good” and “bad” FDs make sense?

If X is *not* a (super)key, it functionally determines *some* of the attributes; therefore, those other attributes can be duplicated

Recall: this means there is <u>redundancy</u> and can lead to <u>anomalies</u>

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

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is **<u>in BCNF</u>** 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} → {Name,City}

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

What is the key?
{SSN, PhoneNumber}

⟹ **Not** in BCNF

Example decomposition

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

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

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]

  <u>if</u> (not found) **<u>then</u> 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 **$R_1(X^+)$** and **R_2(X ∪ 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 $\mathbf{R_2(X \cup Rest)}$

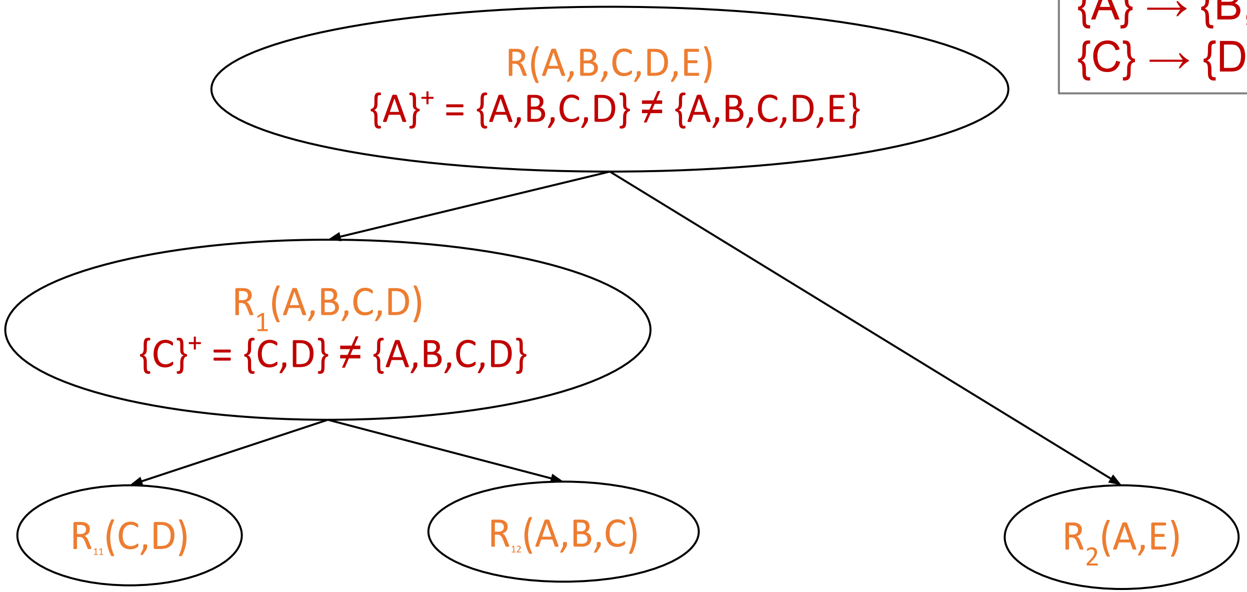
Return BCNFDecomp(R_1), BCNFDecomp(R_2)

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

<u>FDs</u>
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

Key

SID, Class

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
…	…	…

$\{Unit\} \rightarrow \{Company\}$
$\{Company,Product\} \rightarrow \{Unit\}$

Unit	Company
…	…

Unit	Product
…	…

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

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

We lose the FD $\{Company,Product\} \rightarrow \{Unit\}$!!

So Why is that a Problem?

Unit	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

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

{Unit} → {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 {Company,Product} → {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!

Practical Problem: 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…

Other dependencies
MVDs

Example: Student profile

“Student 4749732 is taking
Classes = {cs145, cs 245, cs 222},
Hobbies = {Surfing, Music, Astronomy}”

SID	Class	Hobby
4749732	cs 145	Surfing
4749732	cs 245	Surfing
4749732	cs 222	Surfing
4749732	cs 145	Music
4749732	cs 245	Music
4749732	cs 222	Music
4749732	cs 145	Astronomy
4749732	cs 245	Astronomy
4749732	cs 222	Astronomy
8472374	cs 145	-
8472374	cs 336	-
...		

<u>Problem</u>
FDs?
Lots of redundancy

<u>Root cause</u>
Conditional independence
given SID: Classes & Hobbies are independent

Multi-Value Dependencies (MVDs)

- A multi-value dependency (MVD) is another type of dependency that could hold in our data, ***which is not captured by FDs***
- Formal definition:
  - Given a relation **R** having attribute set **A**, and two sets of attributes $\mathbf{X}, \boldsymbol{Y} \subseteq \boldsymbol{A}$
  - The ***multi-value dependency (MVD)*** $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ holds on R if
  - ***for any tuples*** $\boldsymbol{t_1}, \boldsymbol{t_2} \in \boldsymbol{R}$ s.t. $\boldsymbol{t_1}[X] = \boldsymbol{t_2}[X]$, there exists a tuple $\boldsymbol{t_3}$ s.t.:
    - $\mathbf{t_1[X] = t_2[X] = t_3[X]}$
    - $\mathbf{t_1[Y] = t_3[Y]}$
    - $\mathbf{t_2[A \backslash Y] = t_3[A \backslash Y]}$
      - *Where A \ B means "elements of set A not in set B"*

Multi-Value Dependencies (MVDs)

- Another way to understand MVDs, in terms of *conditional independence:*
- ***The MVD*** $\boldsymbol{X \twoheadrightarrow Y}$ holds on R if given X, Y is conditionally independent of A \ Y and vice versa…

Activity:
Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

Any FDs?

No…

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

For a given movie theatre…

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

For a given movie theatre…

Given a set of movies and snacks…

MVDs: Movie Theatre Example

Movie_theater	film_name	snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 218	Star Wars: The Boba Fett Prequel	Ramen
Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

For a given movie theatre…

Given a set of movies and snacks…

Any movie / snack combination is possible!

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write **{A}** $\twoheadrightarrow$ **{B}** if for any tuples t_1,t_2 s.t. $t_1[A] = t_2[A]$

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write **{A}** $\twoheadrightarrow$ **{B}** if for any tuples t_1,t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write $\{A\} \twoheadrightarrow \{B\}$ if for any tuples t_1, t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$
- $t_3[B] = t_1[B]$

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

More formally, we write **{A}** $\twoheadrightarrow$ **{B}** if for any tuples t_1, t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$
- $t_3[B] = t_1[B]$
- and $t_3[R\backslash B] = t_2[R\backslash B]$

Where R\B is "R minus B" i.e. the attributes of R not in B.

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
	Rains 216	Star Trek: The Wrath of Kahn	Burrito
t_3	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

Note this also works!

Remember, an MVD holds over *a relation or an instance*, so defn. must hold for every applicable pair…

MVDs: Movie Theatre Example

	Movie_theater (A)	film_name (B)	Snack (C)
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
	Rains 216	Star Trek: The Wrath of Kahn	Burrito
t_3	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 218	Star Wars: The Boba Fett Prequel	Ramen
	Rains 218	Star Wars: The Boba Fett Prequel	Plain Pasta

This expresses a sort of dependency (= data redundancy) that we *can't* express with FDs

Actually, it expresses* ***conditional independence* *(between film and snack given movie theatre)!*

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

THANK YOU!