Lecture:
Design Theory

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

Problems
Repeats?
Room/time change?
Deletes?

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

(more compact)

Example Enrollment table - “v1”

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

375 cs145 students

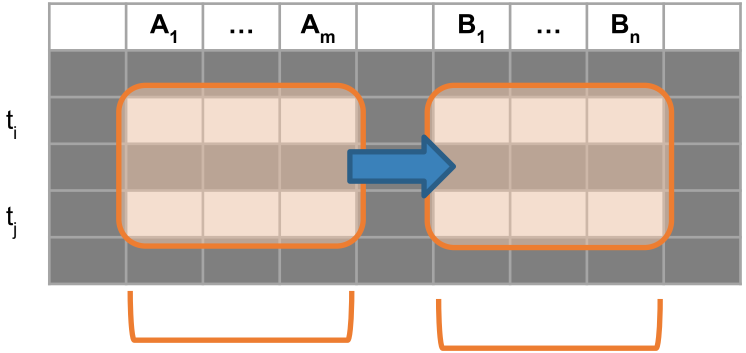
300 cs245 students

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 Picture Of FDs [recall]

A→ B on R



If t1,t2 agree here..

…they also agree here!

Defn (again):
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:

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

then $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

Table Decomposition

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

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$ $\quad$ $R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

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

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

Conceptual Design

For a "mega" table

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

Note: there are several "good" (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

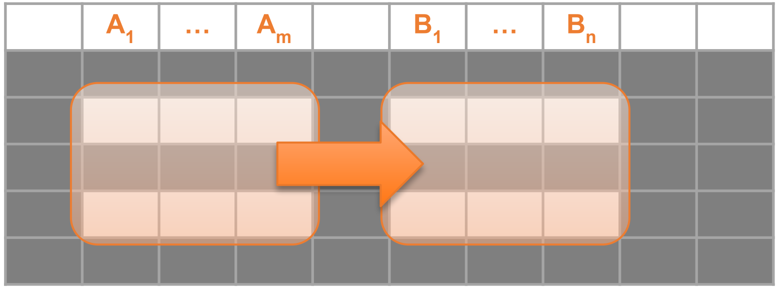
Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

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

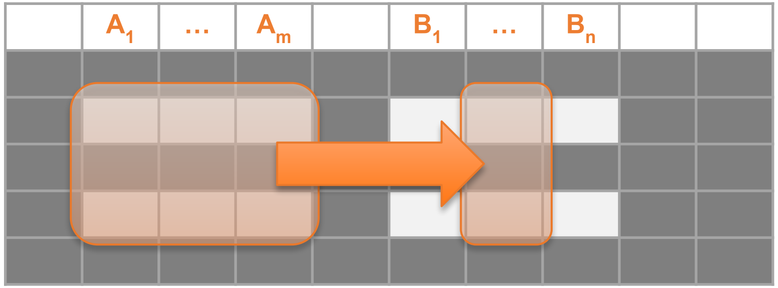
1. **Split/Combine**
2. **Reduction**
3. **Transitivity**

1. Split/Combine



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

1. Split/Combine

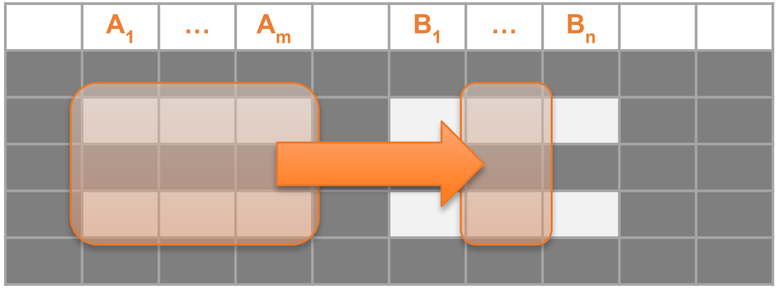


$$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$$

… is equivalent to the following *n* FDs…

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

1. Split/Combine

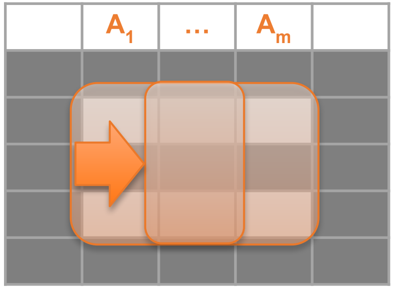


And vice-versa, $A_1,\ldots,A_m \rightarrow B_i$ for $i=1,\ldots,n$

… is equivalent to …

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

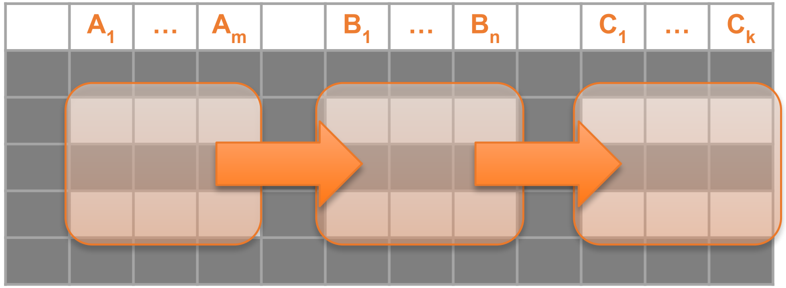
2. Reduction/Trivial



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

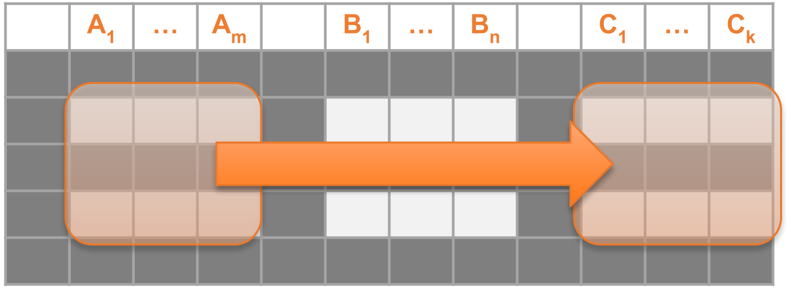
$A_i \rightarrow A_j$ for $i = 1 \ldots m$

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?

Design

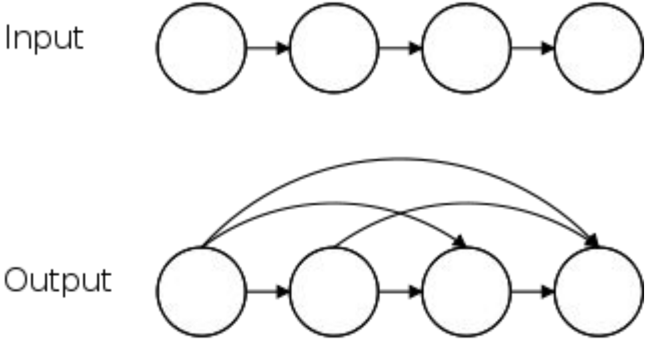
Closures & SuperkeyS

Algebra (reminder)

A => B

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

(Think of Closures as “reachability” in graph)



Closure of a set of Attributes

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

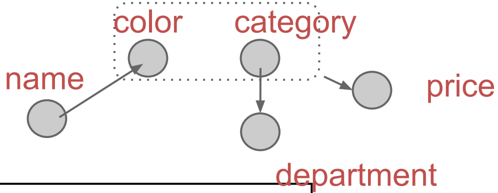
Closure $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. **$\{A_1, \ldots, A_n\} \rightarrow$ B**

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:

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

Example 1

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_n\} \longrightarrow C$ 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}

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

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}

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

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

Example 1

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
  if $\{B_1, \ldots, B_n\} \longrightarrow C$ 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 $\{B_1, \ldots, B_n\} \rightarrow C$ 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}

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

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

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

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

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

Using Closure to Infer ALL FDs

Example:

Given F =

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

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?

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 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)

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

$\{name, category\}^+ = \{name, price, category, color\}$
= the set of all attributes
$\Longrightarrow$ this is a **superkey**
$\Longrightarrow$ 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

(Not tested in Exam)

Decompositions

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

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

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

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

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

Example 1

We need a decomposition to be "correct"

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

Example2: 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?

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

(Lossy ⇒ making up data, "losing' shape of data)

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

$\rightarrow$

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 ⋈ R2

Boyce-Codd Normal Form

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is **<u>in BCNF</u>** if:

if **{A_1, ..., A_n}** → **B** is a FD (*non-trivial*) in R

then **{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

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]

  <u>if</u> (not found) **<u>then</u>** **Return** R

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

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 $\mathbf{R_2(X \cup Rest)}$

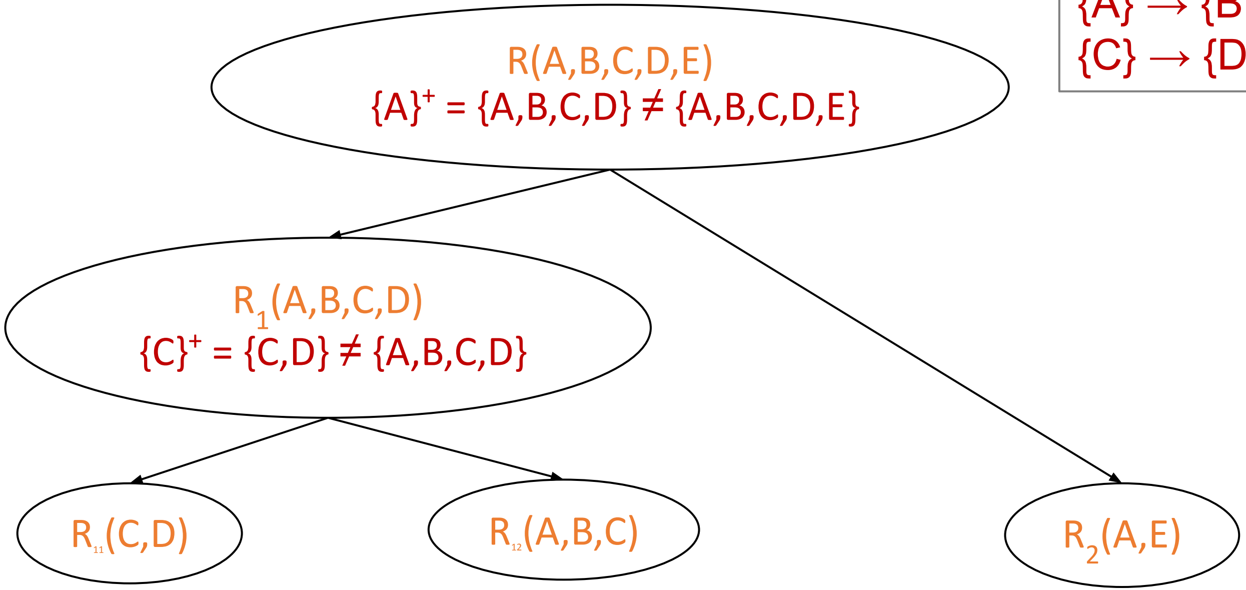
  Return BCNFDecomp(R_1), BCNFDecomp(R_2)

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

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

<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

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

CS145

Goals

Course Summary

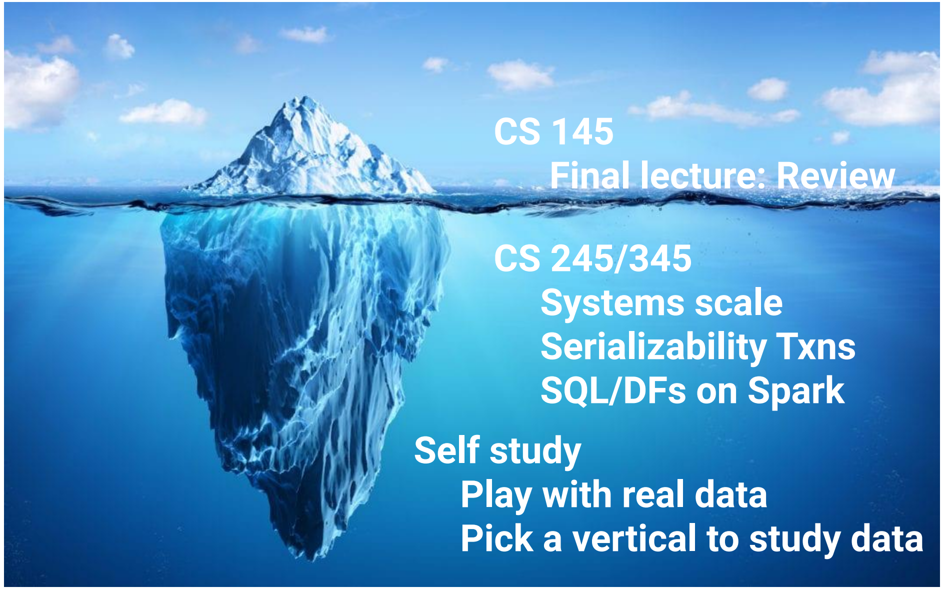
We'll learn How To...

- **Query** over small-med-large data sets with **SQL**? [Weeks 1 and 2]
  - On relational engines, and "big data" engines
- **Scale** for "big queries"? On Clusters? [Weeks 3, 4, 5]
  - OLAP/Analytics, 1st principles of scale
- **Scale** for "big writes"? [Weeks 6, 7, 8]
  - Writes, Transactions, Logging, ACID properties
- **Design** "good" databases? [Weeks 9, 10]
  - Big Schemas, design, functional dependencies, query optimizers

Project: Query-Visualize-Learn on GB/TB scale data sets on a Cloud [sql + python]

Next Steps

Course Summary



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:
$\{\text{Unit}\}^+ = \{\text{Unit, Company}\}$

We lose the FD {Company,Product} → {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}

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

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

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!

<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…

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