Database Systems
Lecture #4 FD

Guifeng Zheng
SYSU

Agenda

- Last time: relational model
- This time:

1. Functional dependencies
    - Keys and superkeys in terms of FDs
    - Finding keys for relations
2. Rules for combining FDs

- Next time: anomalies & normalization

Where are we going, where have we been?

- Goal: manage large amounts of data effectively
  - Use a DBMS
  - must define a schema

- DBMSs use the relational model
  - But initial design is easier in E/R
  - Must design an E/R diagram
  - Must then convert it to rel. model

Where are we going, where have we been?

- At this pt, often find problems – redundancy
  - How to fix?
- Convert the tables to a special “normal” form
  - How to do this?
- First step is: check which FDs there are
  - look at *all true FDs of the table*
  - Then do *decompositions*

Next topic: Functional dependencies

- FDs are *constraints*
  - Logically part of the schema
  - can’t tell from particular relation instances
  - FD may hold for some *instances* “accidentally”

- Finding all FDs is part of DB design
  - Used in normalization

Functional dependencies

- Definition:

If two tuples agree on the attributes

$$\boxed{A_1, A_2, \ldots, A_n}$$

then they must also agree on the attributes

$$\boxed{B_1, B_2, \ldots, B_m}$$

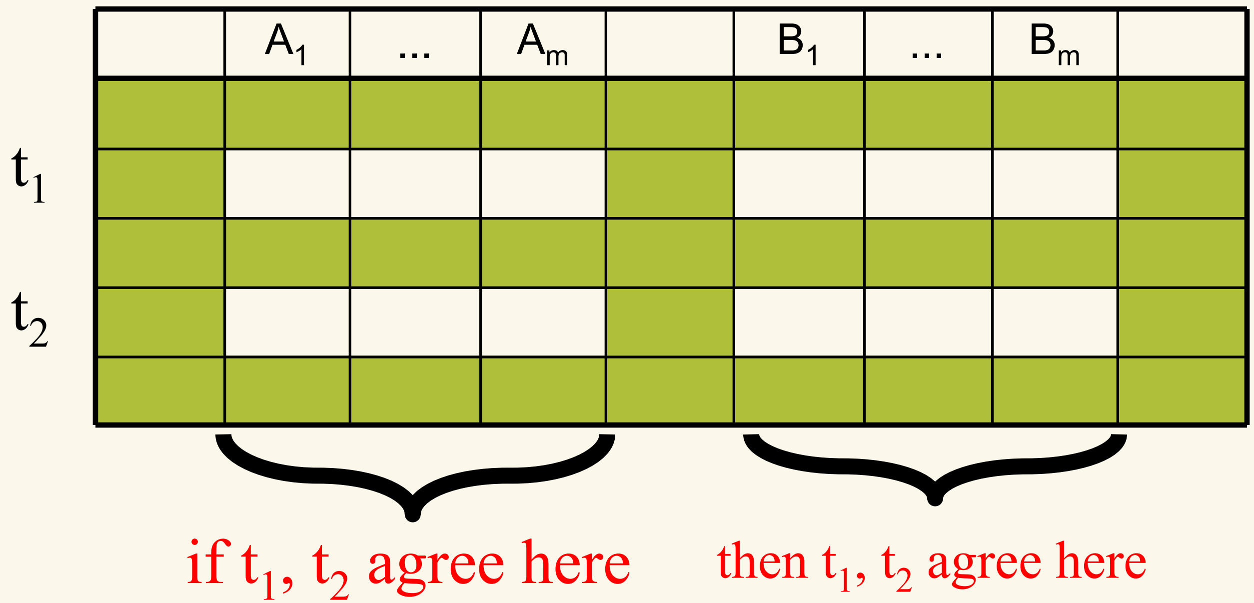
- Notation: $\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$
- Read as: *A_i functionally determines B_j*

Typical Examples of FDs

- Product
  - name → price, manufacturer
- Person
  - ssn → name, age
  - father's/husband's-name → last-name
  - zipcode → state
  - phone → state (*notwithstanding inter-state area codes?*)
- Company
  - name → stockprice, president
  - symbol → name
  - name → symbol

Functional dependencies

- To check A → B, erase all other columns; for all rows t1, t2



- i.e., check if remaining relation is many-one
  - no “divergences”
  - i.e., if A →B is a well-defined function
  - thus, functional dependency

FD example

Product(name, category, color, department, price)

Consider these FDs:

name → color
category → department
color, category → price

What do they say?

FD example

FDs as properties:
- On some instances they hold
- On others they don't

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Does this instance satisfy all the FDs?

FD example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supp.	59

What about this one?

Recognizing FDs

- Q: Is Position→Phone an FD here?

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

- A: It is for this *instance*, but no, presumably not in general
- Others FDs?
- EmpID → Name, Phone, Position
- but Phone $\not\rightarrow$ Position

Keys (candidate key) of relations

- $\{A_1A_2A_3\ldots A_n\}$ is a key for relation R if
  - $A_1A_2A_3\ldots A_n$ functionally determine all other atts
    - Usual notation: $A_1A_2A_3\ldots A_n \rightarrow B_1B_2\ldots B_k$
    - rels = sets $\rightarrow$ distinct rows can't agree on all A_i
  - $A_1A_2A_3\ldots A_n$ is minimal (candidate key)
    - No proper subset of $A_1A_2A_3\ldots A_n$ functionally determines all other attributes of R
- *Primary* key: chosen if there are several possible keys

Keys example

- Relation: Student(ssn,Name, Address, DoB, Email, Credits)
- Which (/why) of the following are keys?
  - SSN
  - Name, Address (on reasonable assumptions)
  - Name, SSN
  - Email, SSN
  - Email

- NB: minimal != smallest

Superkeys

- Df: A set of attributes that contains a key
- Satisfies first condition: determination
- Might not satisfy the second: minimality
  - Some superkey attributes may be superfluous
  - keys are superkeys
- key are special case of superkey
  - superkey set is superset of key set

- name;ssn is a superkey but not a key

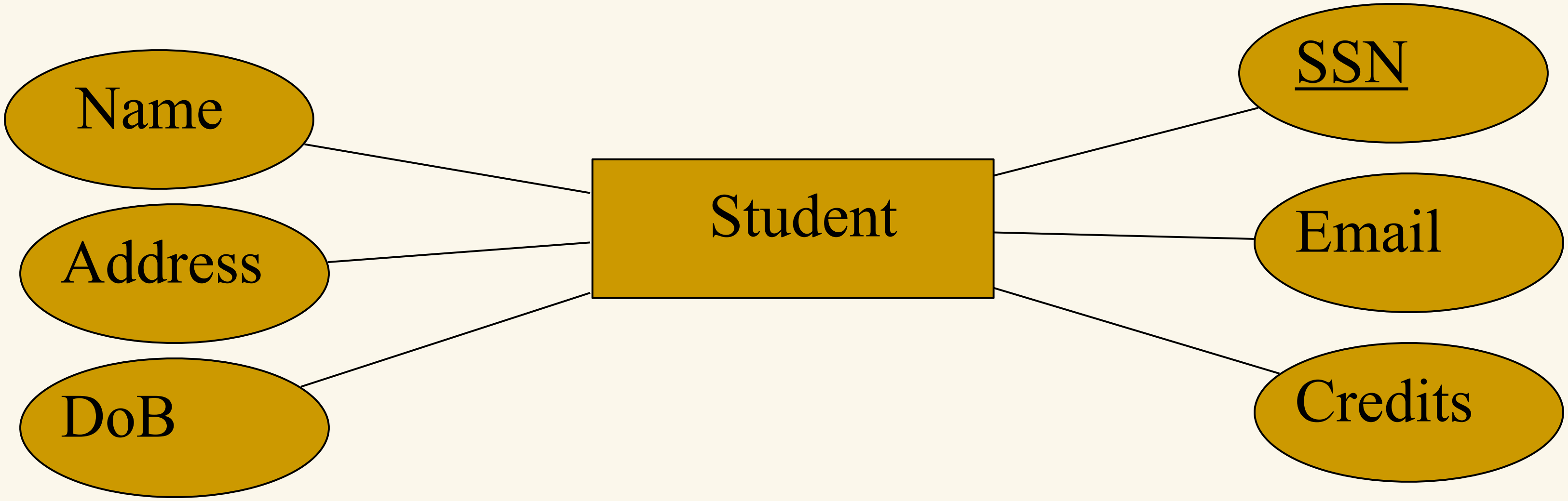
Discovering keys for relations

- Relation ← entity set
  - Key of relation = (minimized) key of entity set

- Relation ← binary relationship
  - Many-many: union of keys of both entity sets
  - Many(M)-one(O): only key of M (why?)
  - One-one: key of either entity set (but not both!)

Review: entity sets

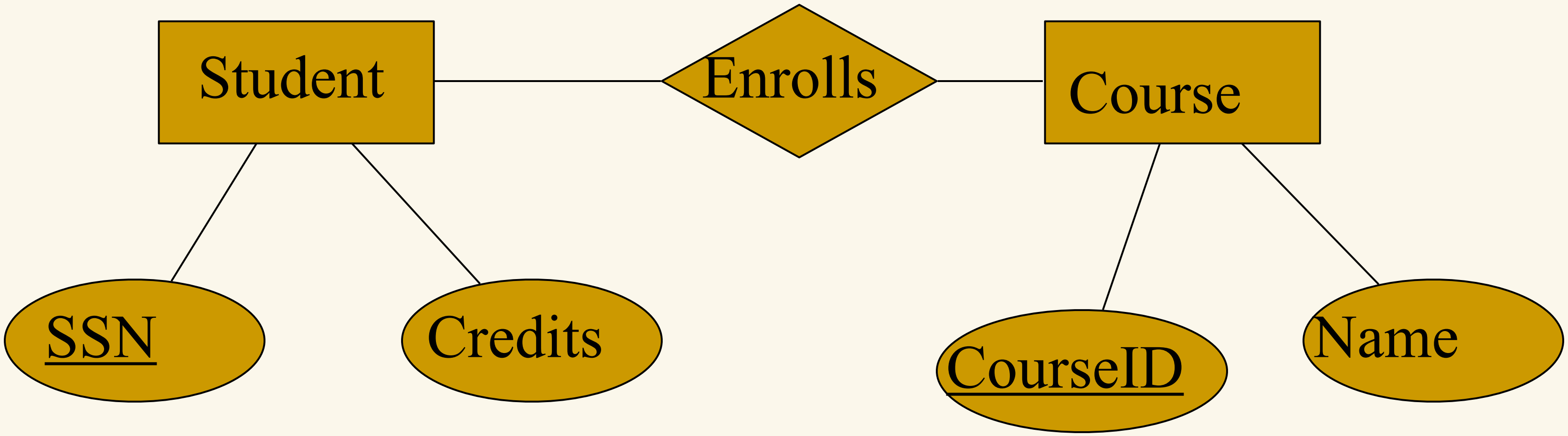
- Key of entity set = key of relation



- Student(Name, Address, DoB, <u>SSN</u>, Email, Credits)

Review: many-many

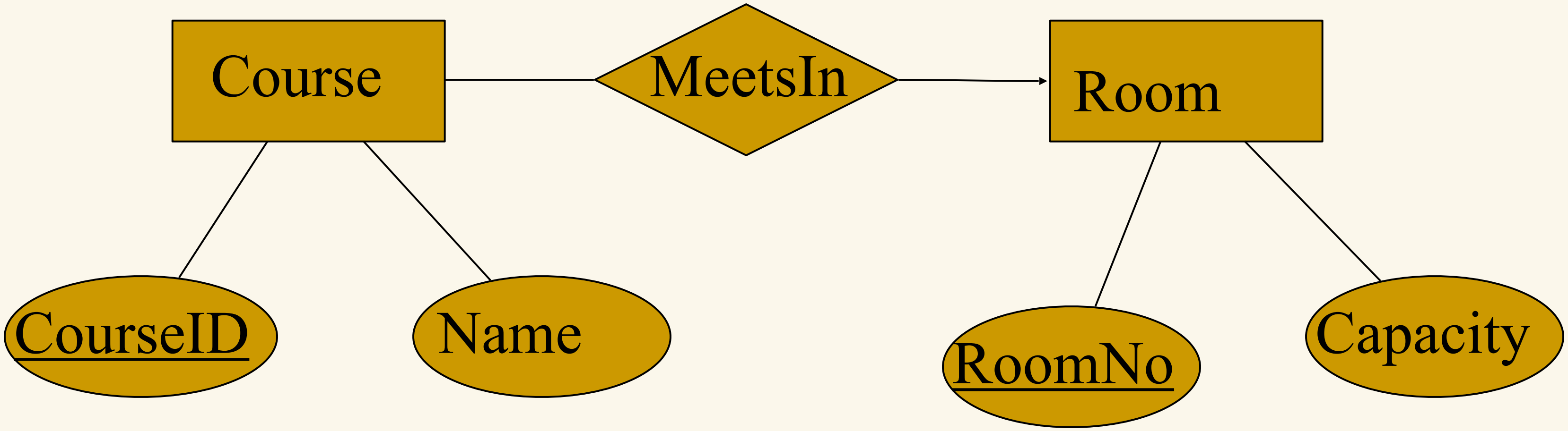
- Many-many key: union of both ES keys



Enrolls(<u>SSN</u>,<u>CourseID</u>)

Review: many-one

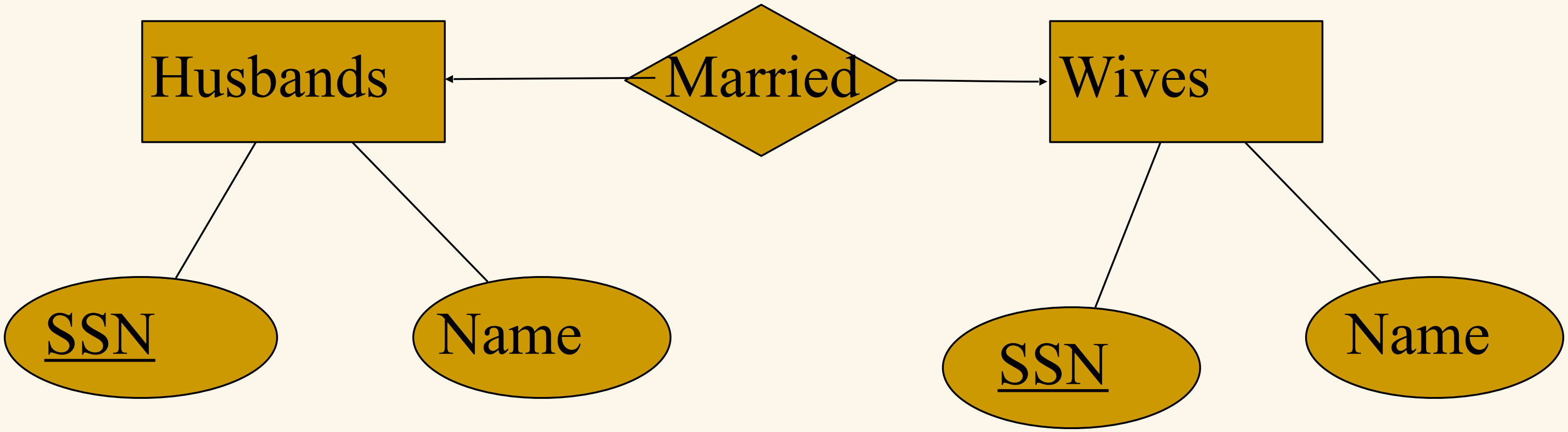
- Key of the *many* ES but not of the *one* ES
  - keys from both would be non-minimal



MeetsIn(<u>CourseID</u>,RoomNo)

Review: one-one

- Keys of both ESs included in relation
- Key is key of either ES (but not both!)

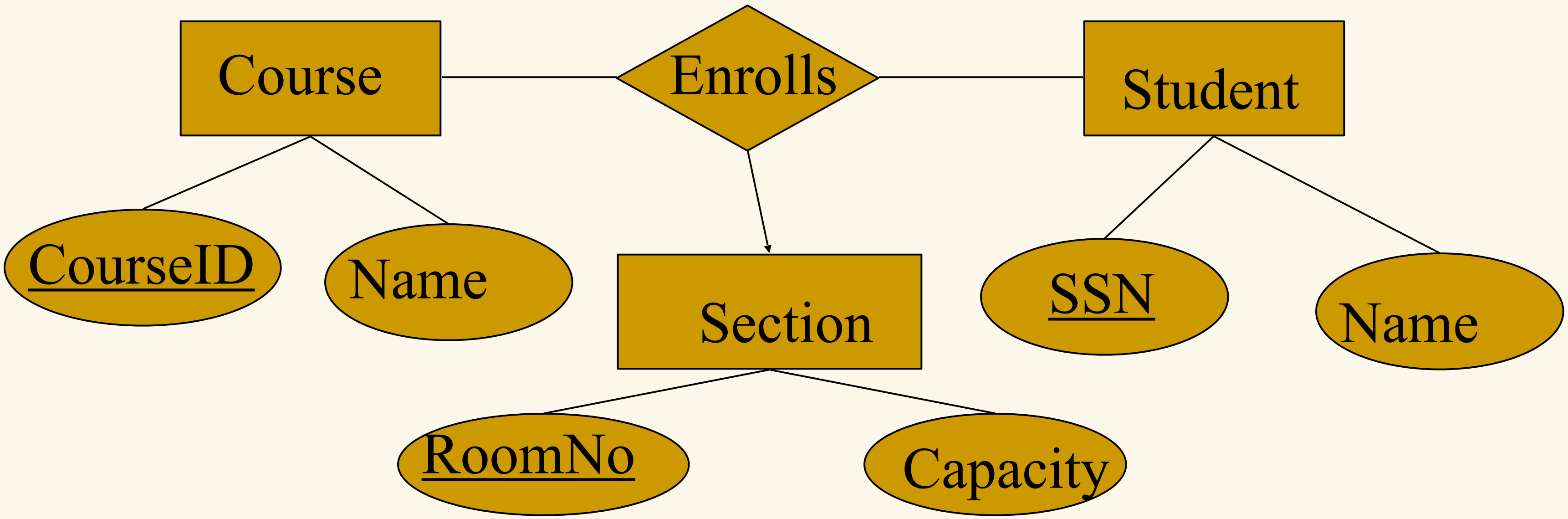


Married(HSSN, WSSN) *or*

Married(HSSN, WSSN)

Review: multiway relships

- Multiple ways – may not be obvious
- R:F,G,H→E is many-one → E's key is *included*
  - but not part of key
  - Recall that relship atts are *implicitly* many-one



Enrolls(<u>CourseID</u>,<u>SSN</u>,RoomNo)

Next topic: Combining FDs

If some FDs are satisfied, then others are satisfied too

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

Why?

Rules for FDs (quickly)

- Reasoning about FDs: given a set of FDs, infer other FDs – useful
  - E.g. $A \rightarrow B$, $B \rightarrow C$ $\Rightarrow$ $A \rightarrow C$

- Definitions: for FD-sets S and T
  - T **follows from** S if all relation-instances satisfying S also satisfy T.
  - S and T are **equivalent** if the sets of relation-instances satisfying S and T are the same.
  - I.e., S and T are equivalent if S follows from T, and T follows from S.

Splitting & combining FDs (quickly)

Splitting rule:

$A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_m$

$\rightleftarrows$

$A_1, A_2, \ldots, A_n \rightarrow B_1$
$A_1, A_2, \ldots, A_n \rightarrow B_2$
. . . . .
$A_1, A_2, \ldots, A_n \rightarrow B_m$

Note: doesn't apply to the left side

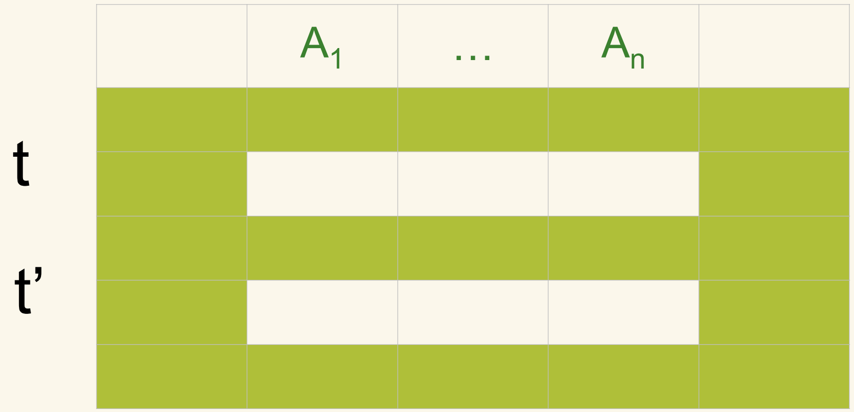
Combining rule:

Q: Can you split and combine the A's, too?

		A1	...	Am		B1	...	Bm	
t1									
t2									

Reflexive rule: trivial FDs (quickly)

$\boxed{A_1, A_2, \ldots, A_n \rightarrow A_i}$ with i in 1..n is a *trivial FD*



- FD $A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_k$ may be
  - **Trivial**: Bs are a subset of As
  - **Nontrivial**: >=1 of the Bs is not among the As
  - **Completely nontrivial**: none of the Bs is among the As
- Trivial elimination rule:
  - Eliminate common attributes from Bs, to get an equivalent completely nontrivial FD

Transitive rule (quickly)

If $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

and $B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p$

then $A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p$

		A_1	…	A_m		B_1	…	B_m		C_1	...	C_p	
t													
t'													

Augmentation rule (quickly)

If $A_1, A_2, \ldots, A_n \rightarrow B$

then $A_1, A_2, \ldots, A_n, C \rightarrow B, C$, for any C

		A_1	…	A_m		B_1	…	B_m		C_1	...	C_p	
t													
t'													

Rules summary (quickly)

1. $A \rightarrow B \Rightarrow AC \rightarrow B$ (by definition)
2. Separation/Combination
3. Reflexive
4. Augmentation
5. Transitivity

- Last 3 called Armstrong's Axioms
  - **Complete**: entire *closure* follows from these
  - **Sound**: no other FDs follow from these

1. Don't need to memorize details…

Inferring FDs example (quickly)

Start from the following FDs:

1. name → color
2. category → department
3. color, category → price

Infer the following FDs:

Inferred FD	Which Rule did we apply?
4. name, category → name	Reflexive rule
5. name, category → color	Transitivity(4,1)
6. name, category → category	Reflexive rule
7. name, category → color, category	combine(5,6) *or Aug(1)*
8. name, category → price	Transitivity(3,7)

Problem: infer *all* FDs

Given a set of FDs, infer all possible FDs

How to proceed?

1. Try all possible FDs, apply all rules
   1. E.g. R(A, B, C, D): how many FDs are possible?
2. Drop trivial FDs, drop augmented FDs
   1. Still way too many

1. Better: use the *Closure Algorithm*…

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+ = \{B \text{ in Atts: } A_1, \ldots, A_n \rightarrow B\}$

Example:

name → color
category → department
color, category → price

Closures:

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

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

$\{\text{color}\}^+ = \{\text{color}\}$

Closure Algorithm

Start with $X=\{A_1, \ldots, A_n\}$.

Repeat:

if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

until X doesn't change

Example:

name $\rightarrow$ color
category $\rightarrow$ department
color, category $\rightarrow$ price

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

Example

In class:

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

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

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

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

What are the keys?

Example: How to find keys

What are the keys?

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

Compute X^+, for every *set* X (*AB* is shorthand for *{A,B}*):

$A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$, BC+ = BC, BD+ = BD, CD+ = CD

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (*no need to compute–why?)*

$BCD^+ = BCD$, $ABCD^+ = ABCD$

Closure alg e.g.

1. Product(name, price, category, color)

   name, category → price
   category → color

   FDs are:
   Keys are: {name, category}

2. Enrollment(student, address, course, room, time)

   student → address
   room, time → course
   student, course → room, time

   FDs are:
   Keys are:

Next time

1. Check course homepage for homework
2. Read ch.19, sections 4-5