CS143
Normalization

Professor Junghoo “John” Cho

Relational Design Theory

- How do we design “good” tables for a relational database?
  - Typically, we start with E/R or UML and convert it into tables
  - Still, there are many choices to make in E/R (or UML) that lead to different tables. Which one is better? Which design should we choose?
- Relational design theory (Normalization theory)
  - Theory on what are “good” table designs
  - Tries to minimize “redundancy” in table design
  - Algorithms that convert “bad” design into “good” design automatically

StudentClass Table

- Q: Is this a good table design?

StudentClass

sid	name	addr	dept	cnum	title	unit
301	James	11 West	CS	143	Database	04
105	Elaine	84 East	EE	284	Signal Processing	03
301	James	11 West	ME	143	Mechanics	05
105	Elaine	84 East	CS	143	Database	04
207	Susan	12 North	EE	128	Microelectronics	03

Redundancies in StudentClass Table

- The same information is included multiple times
- Redundancy leads to potential "anomalies" down the road
  1. Update anomaly: Information may be updated partially and inconsistently
     - Q: What if a student changes the address?
  2. Insertion anomaly: We may not include some information at all
     - Q: What if a student does not take any class?
  3. Deletion anomaly: While deleting some information, we may delete others
     - Q: What if the only class that a student takes gets cancelled?

sid	name	addr	dept	cnum	title	unit
301	James	11 West	CS	143	Database	04
105	Elaine	84 East	EE	284	Signal Processing	03
301	James	11 West	ME	143	Mechanics	05
105	Elaine	84 East	CS	143	Database	04
207	Susan	12 North	EE	128	Microelectronics	03

StudentClass Table

- Q: Is there a better table design? What table(s) will you use?

StudentClass

sid	name	addr	dept	cnum	title	unit
301	James	11 West	CS	143	Database	04
105	Elaine	84 East	EE	284	Signal Processing	03
301	James	11 West	ME	143	Mechanics	05
105	Elaine	84 East	CS	143	Database	04
207	Susan	12 North	EE	128	Microelectronics	03

Coming up with Better Tables

- Q: Any way to arrive at the better design more systematically?
- Q: Where is the redundancy from?

StudentClass

sid	name	addr	dept	cnum	title	unit
301	James	11 West	CS	143	Database	04
105	Elaine	84 East	EE	284	Signal Processing	03
301	James	?	ME	143	Mechanics	05
105	?	84 East	CS	143	?	?
207	Susan	12 North	EE	128	Microelectronics	03

Intuition behind Normalization Theory

- Functional Dependency (FD)
  - Some attributes are “determined” by other attributes:
    e.g., sid $\rightarrow$ (name, addr), (dept, cnum) $\rightarrow$ (title, unit)
  - When there is a funtional dependency we may have redundancy
    e.g., (105, Elaine, 84 East) is stored redundantly, so is (CS, 143, database, 04)
- Decomposition
  - When there is a FD, no need to store multiple instances of this relationship. Store it in a separate table

sid	name	addr	dept	cnum	title	unit
301	James	11 West	CS	143	Database	04
105	Elaine	84 East	EE	284	Signal Processing	03
301	James	11 West	ME	143	Mechanics	05
105	Elaine	84 East	CS	143	Database	04
207	Susan	12 North	EE	128	Microelectronics	03

“Decomposing” StudentClass Table

- StudentClass(sid, name, addr, dept, cnum, title, unit)
  FD: sid → (name, addr), (dept, cnum) → (title, unit)

- Basic idea of “normalization”
  - Whenever there is FD, the table may be ”bad” due to redundancy
  - We use FDs to split (or “decompose”) table and remove the redundancy
- We learn the functional dependency and decomposition theory as the next topic

Overview

- Functional dependency (FD)
  - Definition
  - Trivial functional dependency
  - Logical implication
  - Closure
  - FD and key
- Decomposition
  - Lossless decomposition
- Boyce-Codd Normal Form (BCNF)
  - Definition
  - BCNF decomposition algorithm
- Most theoretical part of the class. Pay attention!
  - If you can’t follow the lecture, you are unlikely to get it by reading textbook

Functional Dependency

Functional Dependency (FD)

- Notation: u[X] – values for the attributes X of tuple u
  - u = (sid: 100, name: James, addr: Wilshire)
    u[sid, name] = (100, James)
- Functional dependency $X \rightarrow Y$
  - For any $u_1, u_2 \in R,$ if $u_1[X] = u_2\,[X],$ then $u_1[Y] = u_2\,[Y]$
  - Informally, $X \rightarrow Y$ means “no two tuples in R can have the same X values but different Y values.”
- Example: StudentClass(sid, name, addr, dept, cnum, title, unit)
  - Q: sid → name?
  - Q: dept, cnum → title, unit?
  - Q: dept, cnum → sid?
  - Whether FD holds or not depends on real-world semantics

Functional Dependency (FD)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_3

Q: $AB \rightarrow C$?

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_1

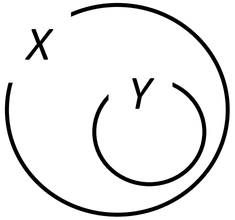
Q: $AB \rightarrow C$?

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_3

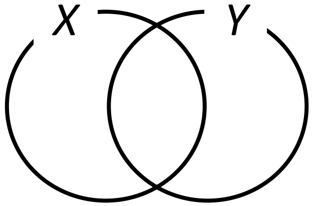
Q: $AB \rightarrow C$?

Trivial Functional Dependency

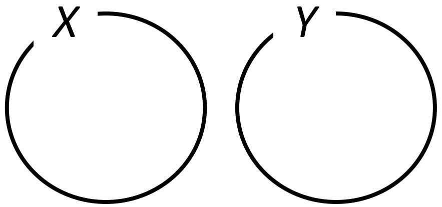
- Trivial FD: $X \to Y$ is a *trivial functional dependency* when $Y \subseteq X$
  - $X \to Y$ is *always true* regardless of real-world semantics



- Non-trivial FD: $X \to Y$ when $Y \nsubseteq X$



- Completely non-trivial FD: $X \to Y$ when $X \cap Y = \emptyset$



Logical Implication

- $R(A, B, C, G, H, I)$
  $\mathrm{F} = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
  Q: Is $A \rightarrow H$ true given F?
- F *logically implies* $A \rightarrow H$
- Canonical database: a method to check logical implication

	A	B	C	G	H	I
u_1	a_1	b_1	c_1	d_1	h_1	i_1
u_2						

Q: $A \rightarrow H$?

Q: $AG \rightarrow I$?

	A	B	C	G	H	I
u_1	a_1	b_1	c_1	g_1	h_1	i_1
u_2						

Closure

- Closure of functional dependency set F: F+
  - F+: the set of all FD's that are logically implied by F
- Closure of attribute set X: X+
  - X+: the set of all attributes that are functionally determined by X
  - Example: what is {sid, dept, cnum}+ given
    sid $\rightarrow$ name, (dept, cnum) $\rightarrow$ (title, unit)?

Closure X+ Computation Algorithm

Start with X+ = X

Repeat until no change in X+:

  If there is $Y \rightarrow Z$ with Y $\subset$ X+, then X+ $\leftarrow$ (X+ $\cup$ Z)

Attribute Closure Example

- $R(A, B, C, G, H, I)$
  $\mathrm{F} = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- Q: {A}+?
- Q: {A, G}+?

Functional Dependency and Key

- $R(A, B, C, G, H, I)$
  $\mathrm{F} = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- Q: Is {A, G} a key of *R*? Is {A, B} a key of *R*?

- X is a key of R if and only if
  1. $X \rightarrow$ all attributes of R (i.e., $X^+ = R$)
  2. No subset of X satisfies the condition 1 (i.e. X is minimal)

Projecting Functional Dependency

- $R(A, B, C, D)$
  $\mathrm{F} = \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$
- Q: What FDs hold for $R'(B, C, D)$ which is a projection of R?

- Note
  - In order to find FD's after projection, we need to compute F+ and pick the FDs from F+ that holds on the projected table

Decomposition

Decomposition

- Our previous "decomposition" example

  StudentClass(sid, name, addr, dept, cnum, title, unit) →

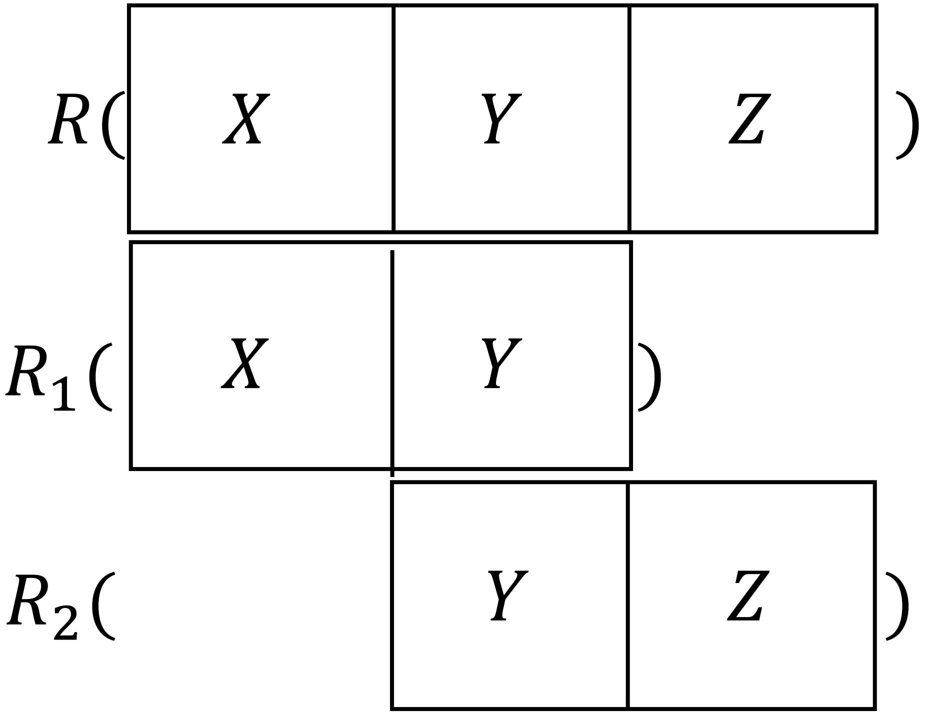
    A(sid, name, addr), B(sid, dept, cnum, title, unit)

- Hopefully, we can "remove redundancy" through a sequence of decompositions using FD's

General Decomposition

- Split $R(A_1, \ldots, A_n) \rightarrow R_1(A_1, \ldots, A_i), R_2\left(A_j, \ldots, A_n\right)$

$$\{A_1, \ldots, A_n\} = \{A_1, \ldots, A_i\} \cup \left\{A_j, \ldots, A_n\right\}$$



Lossless Decomposition

- Q: When we decompose R to R_1 and R_2, what should we watch out for?
- A: Do not lose data!!!
- Lossless-Join Decomposition
  - Decomposition of R into R_1 and R_2 is *lossless-join decomposition* if and only if $R = R_1 \bowtie R_2$
  - After a lossless-join decomposition, we can always get back the original table R if needed!
- Q: When is a decomposition lossless-join?

Lossless-Join Decomposition

cnum	sid	name
143	1	James
143	2	Elaine
325	3	Susan

- Q: Decompositoin into S1(cnum, sid), S2(cnum, name). Lossless?

S1

cnum	sid
143	1
143	2
325	3

S2

cnum	name
143	James
143	Elaine
325	Susan

S1 ⋈ S2

cnum	sid	name
143	1	James
143	1	Elaine
143	2	James
143	2	Elaine
325	3	Susan

Lossless-Join Decomposition

cnum	sid	name
143	1	James
143	2	Elaine
325	3	Susan

- Q: Decompositoin into R1(cnum, sid), R2(sid, name). Lossless?

R1

cnum	sid
143	1
143	2
325	3

R2

sid	name
1	James
2	Elaine
3	Susan

$\text{R1} \bowtie \text{R2}$

cnum	sid	name
143	1	James
143	2	Elaine
325	3	Susan

Lossless-Join Decomposition

- Q: Why is S1(cnum, sid), S2(cnum, name) lossy, but R1(cnum, sid), R2(sid, name) is not?

S1

cnum	sid
143	1
143	2
325	3

S2

cnum	name
143	James
143	Elaine
325	Susan

R1

cnum	sid
143	1
143	2
325	3

R2

sid	name
1	James
2	Elaine
3	Susan

$S1 \bowtie S2$

cnum	sid	name
143	1	James
143	1	Elaine
143	2	James
143	2	Elaine
325	3	Susan

$R1 \bowtie R2$

cnum	sid	name
143	1	James
143	2	Elaine
325	3	Susan

Lossless-Join Decomposition

- Decomposition $R(X, Y, Z) \rightarrow R_1(X, Y),\ R_2(Y, Z)$ is lossless-join if $Y \rightarrow X$ or $Y \rightarrow Z$
  - Shared attribute(s) are the key of one of the decomposed tables
  - This condition can be checked using FDs
- Example
  - StudentClass(sid, name, addr, dept, cnum, title, unit) →
    R1(sid, name, addr), R2(sid, dept, cnum, title, unit)
    using sid → (name, addr). Lossless-join?

Boyce-Codd Normal Form

FD, Key, and Redundancy

- Q: StudentClass(sid, name, addr, dept, cnum, title, unit). Does FD sid → (name, addr) cause redundancy under StudentClass?

- Q: Student (sid, name, addr). Does FD sid → (name, addr) cause redundancy under Student?

- Q: Why does the same FD cause redundancy in one case but not in the other?

- FD $X \rightarrow Y$ leads to redundancy only if X ***does not contain a key***
  - Key insight behind the definition of "Boyce-Codd Normal Form"

Boyce-Codd Normal Form (BCNF)

- Relation *R* is in ***Boyce-Codd Normal Form*** (BCNF) with regard to the set of functional dependencies *F* if and only if for every nontrivial functional dependency $(X \rightarrow Y) \in F$, X contains a key
  - Informally, “normal form” means “good table design”
  - BCNF ensures that there is no redundancy in the table due to FD
- When a table R is not in BCNF, we know that there is redundancy in the table and the design is “bad.”
  - When table R “violates” the BCNF condition, we will have to “redesign” the table so that the new design is in BCNF: “BCNF decomposition algorithm”
  - Decompose R until all decomposed tables are in BCNF

BCNF Example (1)

- Class(dept, cnum, title, unit). FD (dept,cnum) → (title, unit)
- Q: Intuitively, is it a good table design? Any redundancy? Any better design?

- Q: Is it in BCNF?

BCNF Example (2)

- Employee(name, dept, manager).
  F = { name→dept, dept→manager }
- Q: What is English interpretation of the two FDs?
- Q: Intuitively, is it a good table design? Any redundancy? Any better design?
- Q: Is it in BCNF?

BCNF Violation and Table Decomposition

- Decompose tables until all tables are in BCNF
  - For each FD $X \rightarrow Y$ that violates BCNF condition, separate those attributes out into another table to remove redundancy
  - We also have to ensure that this decomposition is lossless

BCNF Decomposition Algorithm

For any R in the schema

If (non-trivial $X \rightarrow Y$ holds on R AND X does not contain a key), then

1) Compute X^+ (X^+: closure of X)

2) Decompose R into R1(X^+) and R2(X, Z)

// X becomes common attributes

// Z: all attributes in R except X^+

Repeat until no more decomposition

BCNF Decomposition Example (1)

- ClassInstructor(dept, cnum, title, unit, instructor, office, fax)
  F = { instructor→office, office→fax, (dept,cnum) →(title,unit),
  (dept,cnum)→instructor }
- Q: What is English interpretation of instructor→office and office→fax?
- Q: Is it in BCNF?
- Q: Is it a good table design intuitively? Any redundancy? Better design?

BCNF Decomposition Example (1)

- ClassInstructor(dept, cnum, title, unit, instructor, office, fax)
  F = { instructor→office, office→fax, (dept,cnum) →(title,unit),
  (dept,cnum)→instructor }
- Normalize ClassInstructor into BCNF using the BCNF decomposition algorithm:

BCNF Decomposition Example (2)

- $R(A, B, C, G, H, I)$
  $\mathrm{F} = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- Q: Is it in BCNF?
- Normalize R into BCNF

Revisiting BCNF Decomposition Algorithm

For any R in the schema

If (non-trivial $X \rightarrow Y$ holds on R AND X does not contain a key), then

1) Compute X^+ (X^+: closure of X)

2) Decompose R into R1(X^+) and R2(X, Z)

// X becomes common attributes

// Z: all attributes in R except X^+

Repeat until no more decomposition

Q: Does the algorithm ensures that it is lossless-join decomposition?

Uniqueness of BCNF Decomposition

- Q: Does the BCNF decomposition algorithm always lead to a unique set of relations?

- Example: $R(A, B, C), \quad F = \{A \rightarrow C,\ B \rightarrow C\}$
  - Q: What if we start decomposition with $A \rightarrow C$?

  - Q: What if we start decomposition with $B \rightarrow C$?

Checking BCNF Condition

- Q: $R_1(A, B)$, $R_2(B, C, D)$ $F = \{A \rightarrow C,\ B \rightarrow A\}$. Are R_1 and R_2 in BCNF?

- We have to check BCNF compliance for all implied functional dependencies in F+, not just for the ones explicitly listed in F.

Good Table Design in Practice

- Normalization splits tables to reduce redundancy.
  - However, splitting tables has negative performance implication
- Example: Instructor: name, office, phone, fax
  name → office, office → (phone,fax)
  - (design 1) Instructor(name, office, phone, fax)
  - (design 2) Instructor(name, office), Office(offce, phone, fax)
  - Q: Retrieve (name, office, phone) from Instructor. Which design is better?
- As a rule of thumb, start with normalized tables and merge them if performance is not good enough

What We Learned

- Relational design theory
- Functional dependency
  - Trivial functional dependency
  - Logical implication
  - Closure
- Decomposition
  - Lossless-join decomposition
- Boyce-Codd Normal Form (BCNF)
  - BCNF decomposition algorithm
- There exist other definitions of “Normal forms”
  - Third normal form, Fourth normal form, …
  - BCNF is most useful and widely used