Chapter 8: Relational Database Design

Database System Concepts, 6th Ed.

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

A Combined Schema without Repetition

- Consider a combined relation

section_A (course_id, sec_id, semester, year, building, room_number, time_slot_id)

From two existing relations

sec_class(course_id, building, room_number, time_slot_id)

section(course_id, sec_id, semester, year)

into one relation

- No data repetition in the combined relation

course_id	*building*	*room_number*	*time_slot_id*
BIO-101	Painter	514	B
BIO-301	Painter	514	A
CS-101	Packard	101	H
CS-101	Packard	101	F
CS-190	Taylor	3128	E
CS-190	Taylor	3128	A
CS-315	Watson	120	D
CS-319	Watson	100	B
CS-319	Taylor	3128	C
CS-347	Taylor	3128	A
EE-181	Taylor	3128	C
FIN-201	Packard	101	B
HIS-351	Painter	514	C
MU-199	Packard	101	D
PHY-101	Watson	100	A

course_id	*sec_id*	*semester*	*year*
BIO-101	1	Summer	2009
BIO-301	1	Summer	2010
CS-101	1	Fall	2009
CS-101	1	Spring	2010
CS-190	1	Spring	2009
CS-190	2	Spring	2009
CS-315	1	Spring	2010
CS-319	1	Spring	2010
CS-319	2	Spring	2010
CS-347	1	Fall	2009
EE-181	1	Spring	2009
FIN-201	1	Spring	2010
HIS-351	1	Spring	2010
MU-199	1	Spring	2010
PHY-101	1	Fall	2009

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Combining schemas may cause some problems…..

inst_dept

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Fig 8.02

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

What About Smaller Schemas?

- Suppose we had started with *inst_dept.*
  - How would we know to split up (**decompose**) it into *instructor* and *department*?
  - In the *department* schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
    - Denote as a **functional dependency**: *dept_name* $\rightarrow$ *building, budget*
  - In *inst_dept schema*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
    - This indicates the need to decompose *inst_dept*
- Not all decompositions are good.
- Suppose we decompose *employee(ID, name, street, city, salary)* into

  employee1 (*ID*, *name*) and *employee2* (*name*, *street, city, salary*)
- The next slide shows how we lose information
  - we cannot reconstruct the original *employee* relation
  - and so, this is a **lossy decomposition**.

A Lossy Decomposition

Fig 8.03

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

employee1

ID	*name*
⋮	
57766	Kim
98776	Kim
⋮	

employee2

name	*street*	*city*	*salary*
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Lossless-Join Decomposition: Relational Algebra Viewpoint

- Decomposition of $R = (A, B, C)$ ➔ $R_1 = (A, B)$, $R_2 = (B, C)$

r

A	*B*	*C*
α	1	A
β	2	B

A	*B*
α	1
β	2

$\Pi_{A,B}(r)$

B	*C*
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	*B*	*C*
α	1	A
β	2	B

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

First Normal Form (1NF) [1/2]

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form (and revisit this in Chapter 22: Object Based Databases)
  - 허락되지 않거나 바람직하지 않은 attribute:
    - Set_valued attribute
    - Relational_valued attribute
    - Divisible_valued attribute

First Normal Form (1NF) [2/2]

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic
    - Human의 입장에서는 CS0012는 non-atomic 으로 간주
    - Database application이 CS0012에서 CS를 break않는다면 atomic으로 간주
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database
    - 응용프로그램으로 출석번호를 분석해서 CS학생수를 count하는것같은 작업이 되는것은 바람직하지 않다
    - 만약 CS학과가 CSE로 이름을 바꾸게 됬다면 CS0012도 쓸데없이 추가작업을 해야…

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form (Data Redundancy가 없는!)
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in "good" form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies
- Functional Dependencies (FD)
  - Constraints on the set of legal relations.
  - Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
  - A functional dependency is a generalization of the notion of a *key.*
  - Provide the theoretical basis for "good" decomposition

Functional Dependencies [1/2]

- Let R be a relation schema : $\alpha \subseteq R$ *and* $\beta \subseteq R$

- The **functional dependency** $\alpha \rightarrow \beta$ **holds on** R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β.

  That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider R(A,B) with the following instance r of R

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Functional Dependencies [2/2]

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys

Consider the schema:

inst_dept (ID, name, salary, dept_name, building, budget)
where superkey = (ID, dept_name)

Naturally, *ID, dept_name* $\rightarrow$ *name, salary, building, budget*

The constraint like "each department has a unique budget" cannot be expressed in superkey, but the following FD can capture the constraint

$$dept\_name \rightarrow budget$$

Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if a relation *r* is legal under a set *F* of functional dependencies. If so, we say that *r* **satisfies** *F.*
  - specify constraints on the set of legal relations
- We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance relation of *instructor* may, by chance, satisfy $name \rightarrow ID$.

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

Figure 8.04: instance of relation r

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

A ➔ C

Figure 8.05: instance of the classroom relation

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

FD: room_number ➔ capacity
옆의 instance에서는 hold하지만
옆의 instance의 schema에서는
hold한다고 생각하기 어렵다.

What if (GateHall, 101, 400) in it?

Trivial Functional Dependencies

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example*:*
    - *ID, name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by $\mathbf{F^+}$.
- F^+ is a superset of *F*.

Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of FDs if for all FDs in F^+ of the form $\alpha \rightarrow \beta$ where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for *R*

Example:

instr_dept (*ID, name, salary, dept_name, building, budget*)

F = { *dept_name* $\rightarrow$ *building, budget* }

where superkey = (ID, dept_name)

The relation *instr_dept* is not in BCNF because a FD *dept_name* $\rightarrow$ *building, budget* holds on *instr_dept,* but *dept_name* is not a superkey

BCNF Intuition: 모든 FD의 left-part가 superkey

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial FD $\alpha \rightarrow \beta$ causes a violation of BCNF.

  Then, we decompose *R* into two smaller relation schemas:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

- In previous example, *instr_dept* (<u>*ID,*</u> *name, salary*<u>*, dept_name,*</u> *building, budget*)

  and we have a FD *dept_name* $\rightarrow$ *building, budget*
  - α = *dept_name*
  - β = *building, budget*

  and *inst_dept* is replaced by 2 relations R1, R2
  - R1 = $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - R2 = $(R - (\beta - \alpha))$ = (*ID, name, salary, dept_name*)

- **Of course, we are only interested in lossless join decomposition!**

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- In order to ensure that *all* functional dependencies hold in a decomposition
  - It is sufficient to test only those dependencies on each individual relation of a decomposition
  - If so, the decomposition is *dependency preserving.*

- But, It is not always possible to achieve a decomposition having properties of both BCNF and dependency preservation
- Therefore, we consider a weaker normal form, known as *third normal form.*

  3NF < BCNF

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:

  $$\alpha \rightarrow \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

  (**NOTE***:* each attribute may be in a different candidate key)

- If a relation is in BCNF, it is in 3NF (since in BCNF one of the first two conditions above must hold).

- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

3NF Intuition: FD의 left-part가 superkey or FD의 right-part – left-part가 다른key에 속해있으면

3NF, but not BCNF Example

- Relation *dept_advisor* :
  - *dept_advisor* (*stu_ID, inst_ID, dept_name)*
    FDs = {*stu_ID, dept_name* $\rightarrow$ *inst_ID*, *inst_ID* $\rightarrow$ *dept_name*}
  - Two candidate keys: (*stu_ID, dept_name)* and *(inst_ID, stu_ID)*
  - R is not BCNF
    - *stu_ID, dept_name* $\rightarrow$ *inst_ID*

      stu_ID dept_name is a superkey
    - *inst_ID* $\rightarrow$ *dept_name*
      - *Inst_ID is not a superkey*
  - *R* is in 3NF
    - *stu_ID, dept_name* $\rightarrow$ *inst_ID*

      stu_ID dept_name is a superkey
    - *inst_ID* $\rightarrow$ *dept_name*
      - *dept_name* is contained in a candidate key

Goals of Normalization

- Let R be a relation scheme with a set F of FDs
- Decide whether a relation scheme R is in “good” form
- In the case that a relation scheme R is not in “good” form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving

Is BCNF Good Enough? [1/2]

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

inst_info (ID, child_name, phone)

  - where an instructor may have more than one phone and can have multiple children

inst_info

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

- There are no non-trivial FDs and therefore the relation is in BCNF
- Even in BCNF, we can have insertion anomalies
  - i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)
(99999, William, 981-992-3443)

Is BCNF Good Enough? [2/2]

- Therefore, it is better to decompose *inst_info* into:

inst_info

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

inst_child

ID	*child_name*
99999	David
99999	William

inst_phone

ID	*phone*
99999	512-555-4321
99999	512-555-1234

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later.

1NF < 2NF < 3NF < BCNF < 4NF

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- <u>8.4 Functional Dependency Theory</u>
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

Wait!!!   Why We Care Functional Dependency?

- Good Property 이 보장되는 smaller relation schema들을 FD를 이용하여 구할수 있다!

- FD는 기본적으로 data의 semantics의 표현
  - Relation에 발생하는 update들에 대해서 functional dependency checking을 해서 violation이 있는 update는 불허함으로 data integrity가 보장된다!

Functional-Dependency Theory

- The formal theory for how FDs are implied logically by a given set of FDs
- Algorithms to generate lossless decompositions into BCNF and 3NF
- Algorithms to test if a decomposition is dependency-preserving

Closure of a Set of Functional Dependencies

- Given a set *F* set of FDs, there are certain other FDs that are logically implied by *F*.
  - For e.g.: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** FDs logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by F^+.

Closure of a Set of Functional Dependencies

- We can find $F^{+,}$ the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\, \alpha \rightarrow \gamma\, \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - **sound** (generate only FDs that actually hold)
  - **complete** (generate all FDs that hold)

- Additional rules for **Armstrong's Axioms** :
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds and $\gamma\, \beta \rightarrow \delta$ holds, then $\alpha\, \gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

  Additional rules are convenient……

Example: Computing the Closure of FDs F^+

- $R = (A, B, C, G, H, I)$
  $F = \{ A \to B$
  $A \to C$
  $CG \to H$
  $CG \to I$
  $B \to H\}$
- some members of F^+
  - $A \to H$
    - by transitivity from $A \to B$ *and* $B \to H$
  - $AG \to I$
    - by augmenting $A \to C$ with G, to get $AG \to CG$
      and then transitivity with $CG \to I$
  - $CG \to HI$
    - by augmenting $CG \to I$ to infer $CG \to \mathrm{CG}I$,
      and augmenting of $CG \to H$ to infer $CGI \to HI$,
      and then transitivity

Algorithm for Computing F^+

- Implementing Armstrong's axioms
- To compute the closure of a set of FDs F:

$F^+ = F$
repeat
    for each FD f in F^+
        apply reflexivity and augmentation rules on f
        add the resulting FDs to F^+
    for each pair of FDs f_1 and f_2 in F^+
        if f_1 and f_2 can be combined using transitivity rule
            then add the resulting FD to F^+
until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

Brute-Force Algorithm : Exponential Computation

Computing Attribute Sets Closure α^+

- Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F*
- Algorithm to compute α^+, the closure of α under *F*

result := α;
while (changes to *result*) **do**
  for each $\beta \rightarrow \gamma$ **in** *F* **do**
   begin
    if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
   end

O(F^2) : Quadratic time in the size of F

Intuitively, FD의 왼쪽이 α 일때 오른쪽에 있는 attribute들의 집합

Ex. F = { A → B, A → C} then A^+ → {A, B, C}

Example of Attribute Set Closure α^+

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is AG a candidate key?
  1. Is AG a superkey?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$?
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$?
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$?

Uses of Attribute Closure α^+

There are several uses of the attribute closure (α^+) algorithm:

- Testing for superkey
  - To test if α is a superkey, we check if α^+ contains all attributes of R
  - This is polynomial time, but for every $\alpha \subseteq R$, *checking if* α is superkey is exponential time
- Testing FDs
  - To check if a FD $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$
    - It is a simple and cheap test, and very useful
- Computing closure of F (= F^+)
  - Definitely F^+ requires exponential time!
  - But, alternative simple way (of course, not perfect) for F^+
    - For each $\gamma \subseteq R$, compute γ^+, and for each $S \subseteq \gamma^+$, generate a FD $\gamma \rightarrow S$

Redundant FDs

- Intuitively, a canonical cover of F
  - a "minimal" set of FDs equivalent to F
  - No redundant FDs or No redundant parts of FDs (extraneous attributes in FDs)

- Sets of FDs may have redundant FDs that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ \ B \rightarrow C,\ A \rightarrow C\}$
  - Parts of a FD may be redundant
    - E.g.: on RHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}$ can be simplified to
      $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$
    - E.g.: on LHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ can be simplified to
      $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$

Extraneous Attributes in FDs

- Consider a set F of FDs and the FD $\alpha \rightarrow \beta$ in F.
  - Attribute A is **extraneous** in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is **extraneous** in β if $A \in \beta$ and the set of FDs $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" FD always implies a weaker FD
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C

Redundant FD의 왼쪽이나 오른쪽에 있는 attribute중에 없어도 되는 attribute

Testing for Extraneous Attribute

- Consider a set *F* of FDs and the FD $\alpha \rightarrow \beta$ in *F*.
- To test if attribute $A \in \alpha$ is extraneous in α
  1. compute $(\alpha - A)^+$ using the FDs in *F*
  2. check that $(\alpha - A)^+$ contains β; if it does, *A* is extraneous in α

- To test if attribute $A \in \beta$ is extraneous in β
  1. compute α^+ using only the FDs in $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  2. check that α^+ contains *A*; if it does, *A* is extraneous in β

Canonical Cover F_c

Minimal Cover

- A **canonical cover** for F is a set of FDs F_c such that
  - F logically implies all FDs in F_c, and
  - F_c logically implies all FDs in F, and
  - No FD in F_c contains an extraneous attribute, and
  - Each left side of FD in F_c is unique

- To compute a canonical cover for F:

repeat
1. Replace any FDs in F $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$ using the Union rule
2. Find a FD $\alpha \rightarrow \beta$ with an extraneous attribute either in α or in β
   /* Note: test for extraneous attributes done using F_c, not F*/
   If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until F does not change

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

- Canonical cover에 있는 FD에 대해서만 update가 valid한지를 보면 충분!

Example: Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Drop $A \rightarrow B$ and the FD set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other FDs
    - Yes: in fact, $B \rightarrow C$ is already present!
  - Drop $AB \rightarrow C$ and the FD set is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other FDs
    - Yes: $A \rightarrow C$ is logically implied using transitivity on $A \rightarrow B$ and $B \rightarrow$ C
    - Replace $A \rightarrow BC$ with $A \rightarrow B$
- The final canonical cover is: $A \rightarrow B$
  $B \rightarrow C$

FD and Lossless-Join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if at least one of the following FDs is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

- The above FDs are a sufficient condition for lossless join decomposition; the FDs are a necessary condition only if all constraints are FDs

Example: Decomposition with FDs

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{B\} \quad \text{and } B \rightarrow BC$$

  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{A\} \quad \text{and } A \rightarrow AB$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Dependency Preservation Checking after Decomposition

- Let F_i be the set of FDs in F^+ that include only attributes in R_i
- A Brute-Force Algorithm for **dependency preserving**, checking if

  $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$

  - If it is not, then checking updates for violation of FDs may require computing joins, which is expensive

Compute F^+

For each schema R_i in D do

F_i = the restriction of F^+ *to* R_i // F^+의 FD중에 lhs와 rhs의 attribute 가 R_i 에 속함

$F' = \{\}$

For each restriction F_i in D do

$F' = F' \cup F_i$

Compute F'^+

If $F'^+ == F^+$ *Then return(True) Else return(False)*

- Computing F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$ require exponential time
- There is alternative for dependency preservation checking that avoids computing F^+

Polynomial Time Testing for Dependency Preservation

- To check if a FD $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to F)

  - *result* = α
    while (changes to *result*) do
      for each R_i in the decomposition
        $t = (result \cap R_i)^+ \cap R_i$
        $result = result \cup t$
  - If *result* contains all attributes in β, then the FD $\alpha \rightarrow \beta$ is preserved
  - Intuition: $R_1, R_2, \ldots, R_n$ 중에 (α, β) 를 포함하는 R_i 이 있으면 FD $\alpha \rightarrow \beta$ 는 preserved된것

- We apply the test on all FDs in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Example: Dependency Preserving Decomposition

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $\quad B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF

  ($A \rightarrow B$ 는 A가 key이므로 OK, $B \rightarrow C$ 는 B가 key가 아니므로 not OK)

- Therefore, need to decompose R *into* $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving: $A \rightarrow B$ *is preserved in* R_1, $B \rightarrow C$ *is preserved in* R_2

- However, it is not always possible to have a BCNF decomposition which has both the lossless-join property and the dependency preservation property

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- <u>8.5 Algorithms for Decomposition</u>
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

BCNF Testing with FDs only in F

- To check if a non-trivial FD $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. Compute α^+ (the attribute closure of α), and
  2. Check if α^+ includes all attributes of *R*, that is, it is a superkey of *R*

- Biscup, Dayal, and Bernstein at 1979 SIGMOD proved
  - Suppose FDs in F contains only attributes in R
  - If none of the FDs in *F* causes a violation of BCNF, then none of the FDs in F^+ will cause a violation of BCNF either
- Therefore, we can have the simple test to check if a relation schema *R* is in BCNF when FDs in F contains only attributes in R ,
  - it suffices to check only the FDs in the given set *F* for violation of BCNF, rather than checking all FDs in F^+
- Example: Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B,\ BC \rightarrow D\}$
  - FD in F 들은 전부 R의 attribute를 가지고 있으므로, R이 BCNF인지 아닌지는 FD in F 만 check하는것으로 충분!
  - 이 예제에서는 A도 R의superkey가 아니고, BC도 R의 superkey가 아니므로 R은 BCNF가 아니다

The Simple BCNF Test is not always working

- Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B,\ BC \rightarrow D\}$
  - We concluded R is not in BCNF using the Simple BCNF test in the previous slide
  - Suppose we decompose R into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
  - Suppose we check whether R_1 and R_2 are in BCNF using the simple BCNF test
    - R_1 is in BCNF because only FD in F contain only attributes from R_1 is $A \rightarrow B$ *and A is the superkey of* R_1
    - R_2 may be thought in BCNF because neither of the FDs in F contain only attributes from R_2 (A,C,D,E)
  - In fact, a FD $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF
    - The simplified BCNF test is not working after decomposition
    - 그리고 F^+ requires exponential time

** R_1 and R_2 로 decompose 된이후에 $BC \rightarrow D$ 가 R_1 and R_2 에 있는 attribute로만 구성된것이 아니므로 the simple BCNF Test를 쓸수가 없다.

Polynomial Time BCNF Test

- To check if a relation R_i in a decomposition of R is in BCNF,
  - For every subset α of attributes in R_i
    check that α^+ either includes no attribute of $R_i - \alpha$,
    or includes all attributes of R
  - If the above condition is violated by some set of attributes α in R_i
    the following FD $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ can be shown in F^+
    Therefore, we can conclude the FD shows that R_i violates BCNF
- Later we use the above FD $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ to decompose R_i into two smaller relations having the BCNF property

BCNF Intuition: 모든 FD의 left-part가 superkey

BCNF Decomposition Algorithm

result := {*R* };
done := false;
compute F^+;   // exponential time
while (not *done*) **do**
  if (there is a schema R_i in *result* that is not in BCNF)   // polynomial time
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial FD that holds on R_i
        such that $\alpha \rightarrow R_i$ is not in F^+, and $\alpha \cap \beta = \emptyset$;

      result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    end
  else *done* := **true;**

Note: Each R_i is in BCNF and the decomposition is lossless-join

The above algorithm is exponential time!
In literature, there is a polynomial time algorithm for
BCNF Decomposition generating over-normalized schemas!

F에 있는 FD들이 R에 있는 attribute들로만 구성되어 있으면 F^+ 를 구하지 않고 F에 있는 FD만을 check해서 decomposition 하면 된다.

BCNF Decomposition with The Simplified BCNF Test

We can use the simple BCNF test with only FDs in F (i.e. without computing F^+) if all FDs contains only attributes in R

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ 에서 B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

Example: BCNF Decomposition [1/2]

In this example, we use the simplified test without computing F^+

- *R* = *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- FDs:
  - (FD1) *course_id* → *title*, *dept_name*, *credits*
  - (FD2) *building*, *room_number* → *capacity*
  - (FD3) *course_id*, *sec_id*, *semester*, *year* → *building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.

- BCNF Decomposition:
  - (FD1) *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey ➔ the *class* relation schema is not BCNF
  - We replace the *class relation schema* by the following 2 relation schema:
    - *course* (*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

  The course relation schema is in BCNF

Example of BCNF Decomposition [2/2]

- *Now we check class-1 schema with FD2 and FD3 (note that FD2 and FD3 contains only attributes from class-1)*

  class-1 (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

- (FD2) *building*, *room_number* → *capacity* holds on the *class-1 relation schema*
  - but {*building*, *room_number*} is not a superkey for *class-1* ➔ class-1 is not BCNF
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

- *Now we check section schema with FD3 (note that FD3 contains only attributes from section)*
- (FD3) *course_id*, *sec_id*, *semester*, *year* → *building*, *room_number*, *time_slot_id* *holds* in the section relation schema and the lhs of FD3 is a superkey of section
  - So the section relation schema is in BCNF

- So, finally we get "course", "classroom", "section" BCNF decomposition from "class"

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\quad L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF *($L \rightarrow K$ 에서 L이 key가 아니므로)*
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

This implies that testing for $JK \rightarrow L$ requires a join of decomposed schemas

- BCNF가 아니라서 decomposition을 해서 작은 relation schema들로 만들어
  BCNF를 만들고 싶지만 이번에는 dependency preservation이 지켜지질 않는다!

Motivation of Third Normal Form

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But FDs can be checked on individual relations without computing a join
  - There is always a lossless-join, dependency-preserving decomposition into 3NF

- A relation schema *R* is in **third normal form (3NF)** if for all:

$$\alpha \rightarrow \beta \text{ in } F^+$$

at least one of the following holds:

  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R.*

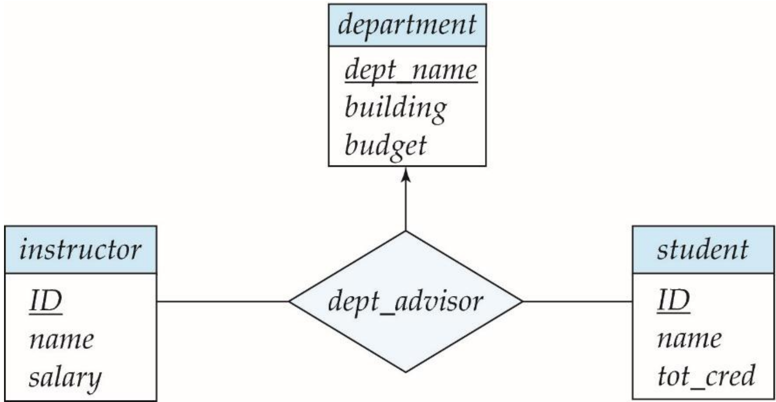
  (**NOTE**: each attribute may be in a different candidate key)

Example: 3NF < BCNF

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\quad L \rightarrow K\}$
  Two candidate keys = *JK* and *JL*
- *R* is not in BCNF
  - *$L \rightarrow K$ 에서 L이 key가 아니므로*
  - *R을 decompose해야 BCNF 인 schema를 만들지만 dependency preservation이 안되는 문제가 있다*
- *R* is in 3NF
  - *$L \rightarrow K$ 에서 L이 key가 아니지만 K가 candidate key의 일부이므로*

Example: Testing 3NF

- Relation *dept_advisor* :
  - *dept_advisor (stu_ID, inst_ID, dept_name)*
    FDs = {*stu_ID, dept_name* $\rightarrow$ *inst_ID,*   *inst_ID* $\rightarrow$ *dept_name*}
  - Two candidate keys: (*stu_ID, dept_name)* and *(inst_ID, stu_ID)*
  - *dept_advisor* is in 3NF
    - *stu_ID, dept_name* $\rightarrow$ *inst_ID*
      - *stu_ID dept_name* is a superkey
    - *inst_ID* $\rightarrow$ *dept_name*
      - *inst_ID is not a superkey*   *(so, dept_advisor* is not BCNF *)*
      - *but, dept_name* is contained in a candidate key



Redundancy in 3NF

- There is some redundancy in this 3NF schema (not BCNF)
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$
  - Superkey = JK

FD $L \rightarrow K$ 에서 L이 key가 아니지만, K가 key의 일부분, so 3NF

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- repetition of information (e.g., the relationship l_1, k_1)
- Sometimes, need to use null values
  - e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J

- *3NF dept_advisor (stu_ID, inst_ID, dept_name)*
  FDs = {*stu_ID, dept_name* $\rightarrow$ *inst_ID*, *inst_ID* $\rightarrow$ *dept_name*}
  Two candidate keys: (*stu_ID, dept_name*) and (*inst_ID, stu_ID*)

CS과의 L교수가 지도학생이 없으면 ➔ *(null, L, CS)*

Testing for 3NF

- 3NF Testing Algorithm
  - Use attribute closure to check for each FD $\alpha \rightarrow \beta$, if α is a superkey
  - If α is not a superkey, further check if each attribute in β is contained in a candidate key of *R*
- Testing for 3NF is NP-hard since it involve finding all candidate keys which is actually computing F^+
- Interestingly, decomposition into 3NF (described shortly) can be done in polynomial time
- The algorithm in the next page ensures:
  - each relation schema R_i is in 3NF
  - decomposition is dependency preserving and lossless-join

3NF Testing이 every candidate key를 다 생성해서 testing을 하면 exponential time이지만
Candidate key set도 주어지고, 주어진 FD의 lhs와 rhs가 전부 R의 attribute 들이면
3NF Testing도 the simplied version으로 할수 있다

3NF Decomposition Algorithm

The loss-less join property and dependency-preservation property is guaranteed!

Let F_c be a canonical cover for F;
i := 0;
for each FD $\alpha \rightarrow \beta$ in F_c **do**
    $i := i + 1$;
    $R_i := \alpha\ \beta$

if none of the schemas R_j, $j = 1,2,..i$ contains a candidate key for R
    then
        $i := i + 1$;
        R_i := any candidate key for R;

/* Optionally, remove redundant relations */

repeat
    if any schema R_j is contained in another schema R_k
    then /* delete R_j */
        $R_j = R_i$;
        $i = i\text{-}1$;
until no more R_js can be deleted
return $(R_1, R_2, ..., R_i)$

Polynomial Time Algorithm
- Not use F^+
- Not finding all candidate keys

** Correctness of This Algorithm is shown at the last part of PPT

Example: 3NF Decomposition [1/2]

- Relation schema:

  cust_banker_branch = (*<u>customer_id, employee_id</u>, branch_name, type*)

  - The FDs for this relation schema are:
    1. *customer_id, employee_id* $\rightarrow$ *branch_name, type*
    2. *employee_id* $\rightarrow$ *branch_name*
    3. *customer_id, branch_name* $\rightarrow$ *employee_id*
  - Candidate key ➔ {*<u>customer_id, employee_id</u>* }

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st FD
  - No other attribute is extraneous,

    so we get F_C =

    customer_id, employee_id $\rightarrow$ *type*
    employee_id $\rightarrow$ *branch_name*
    customer_id, branch_name $\rightarrow$ *employee_id*

Example: 3NF Decomposition [2/2]

- The **for** loop generates following 3NF schema (union of lhs and rhs of FD):

  (*customer_id, employee_id, type*)

  (*employee_id, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added

- At end of for loop, detect and delete redundant schemas, such as (*employee_id, branch_name*), which are subsets of other schemas

  - result will not depend on the order in which FDs are considered

- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

And loss-less join property and dependency-preservation property is guaranteed!

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved

- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Design Goals

- Goal for a relational database design is:
  - BCNF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying FDs other than superkeys.
- SQL can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a FD whose left hand side is not a key.

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

Multivalued Dependencies [1/2]

- Suppose we record names of children, and phone numbers for instructors:
  - *inst_child*(*ID*, *child_name*)
  - *inst_phone*(*ID*, *phone_number*)
- If we were to combine these schemas to get
  - *inst_info*(*ID*, *child_name*, *phone_number*)
  - Example data:
    (99999, David, 512-555-1234)
    (99999, David, 512-555-4321)
    (99999, William, 512-555-1234)
    (99999, William, 512-555-4321)
- This relation is in BCNF
  - Why?

Multivalued Dependencies [2/2]

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \rightarrow\rightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$
\begin{aligned}
&t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\
&t_3[\beta] = t_1[\beta] \\
&t_3[R - \beta] = t_2[R - \beta] \\
&t_4[\beta] = t_2[\beta] \\
&t_4[R - \beta] = t_1[R - \beta]
\end{aligned}
$$

- Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

$\rightarrow\rightarrow$

MVD Example [1/2]

- Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

$$Y, Z, W$$

- We say that $Y \rightarrow\rightarrow Z$ (Y **multidetermines** Z)
if and only if for all possible relations $r(R)$

$$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$

then

$$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

- Note that since the behavior of Z and W are identical it follows that

$Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

MVD Example [2/2]

- In our example:

$$ID \rightarrow\rightarrow child\_name$$
$$ID \rightarrow\rightarrow phone\_number$$

- The above formal definition is supposed to formalize the notion that given a particular value of Y (ID) it has associated with it a set of values of Z *(child_name)* and a set of values of W *(phone_number)*, and these two sets are in some sense independent of each other.
- Note:
  - If $Y \rightarrow Z$ then $Y \rightarrow\rightarrow Z$
  - Indeed we have (in above notation) $Z_1 = Z_2$
    The claim follows.

Example data:
(99999, David, 512-555-1234)
(99999, David, 512-555-4321)
(99999, William, 512-555-1234)
(99999, William, 512-555-4321)

Figure 8.14: An example redundancy in a relation on a BCNF schema

dept_name	*ID*	*street*	*city*
Physics	22222	North	Rye
Physics	22222	Main	Manchester
Finance	12121	Lake	Horseneck

Figure 8.15: An illegal r2 relation

dept_name	*ID*	*street*	*city*
Physics	22222	North	Rye
Math	22222	Main	Manchester

Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
  2. To specify **constraints** on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r′* that does satisfy the multivalued dependency by adding tuples to *r.*

Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  - If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$

  That is, every functional dependency is also a multivalued dependency
- The **closure** D^+ of *D* is the set of all functional and multivalued dependencies logically implied by *D*.
  - We can compute D^+ from *D*, using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules (see Appendix C).

$\rightarrow\rightarrow$

Fourth Normal Form

- A relation schema R is in **4NF** with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema R
- If a relation is in 4NF it is in BCNF

Restriction of Multivalued Dependencies

- The restriction of D to R_i is the set D_i consisting of
  - All functional dependencies in D^+ that include only attributes of R_i
  - All multivalued dependencies of the form

    $$\alpha \rightarrow\rightarrow (\beta \cap R_i)$$

    where $\alpha \subseteq R_i$ and $\alpha \rightarrow\rightarrow \beta$ is in D^+

4NF Decomposition Algorithm

result: = {R};
done := false;
compute D^+;
Let D_i denote the restriction of D^+ to R_i

while (**not** *done*)
    if (there is a schema $\mathbf{R}_i$ in *result* that is not in 4NF) **then**
        begin
            let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \phi$;
            result := (*result* - R_i) $\cup$ (R_i - β) $\cup$ (α, β);
        end
    else *done*:= true;

Note: each R_i is in 4NF, and decomposition is lossless-join

Example of 4NF Decomposition

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow\rightarrow B$

  $B \rightarrow\rightarrow HI$

  $CG \rightarrow\rightarrow H \}$
- R is not in 4NF since $A \rightarrow\rightarrow B$ and A is not a superkey for R
- Decomposition

  a) $R_1 = (A, B)$ (R_1 is in 4NF)

  b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF, decompose into R_3 and R_4)

  c) $R_3 = (C, G, H)$ (R_3 is in 4NF)

  d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF, decompose into R_5 and R_6)

  - $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow HI$ ➔ $A \rightarrow\rightarrow HI$, (MVD transitivity), and
  - and hence $A \rightarrow\rightarrow I$ *(MVD restriction to R_4)*

  e) $R_5 = (A, I)$ (R_5 is in 4NF)

  f) $R_6 = (A, C, G)$ (R_6 is in 4NF)

Chapter 8: Relational Database Design

Further Normal Forms

- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables
  - *R* could have been a single relation containing *all* attributes that are of interest (called **universal relation**)
  - Normalization breaks *R* into smaller relations
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes *department_name* and *building*, and a functional dependency *department_name* $\rightarrow$ *building*
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*
- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as

  $course \bowtie prereq$

  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

  Instead of *earnings* (*company_id, year, amount*), use

  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

Chapter 8: Relational Database Design

- 8.1 Features of Good Relational Design
- 8.2 Atomic Domains and First Normal Form
- 8.3 Decomposition Using Functional Dependencies
- 8.4 Functional Dependency Theory
- 8.5 Algorithms for Decomposition
- 8.6 Decomposition Using Multivalued Dependencies
- 8.7 More Normal Forms
- 8.8 Database-Design Process
- 8.9 Modeling Temporal Data

Modeling Temporal Data [1/2]

- **Temporal data** have an association time interval during which the data are *valid*
- A **snapshot** is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard
- Adding a temporal component results in functional dependencies like

  $$ID \rightarrow street, city$$

  not to hold, because the address varies over time

- A **temporal functional dependency** $X \xrightarrow{\tau} Y$ holds on schema *R* if the functional dependency $X \rightarrow Y$ holds on all snapshots for all legal instances r (*R*).

Modeling Temporal Data [2/2]

- In practice, database designers may add start and end time attributes to relations
  - E.g., *course*(*course_id, course_title*) is replaced by

    course(*course_id, course_title, start, end*)
    - Constraint: no two tuples can have overlapping valid times
      - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time
  - E.g., student transcript should refer to course information at the time the course was taken

End of Chapter

Database System Concepts, 6th Ed.

Proof of Correctness of 3NF Decomposition Algorithm

Database System Concepts, 6^{th} Ed.

Correctness of 3NF Decomposition Algorithm [1/3]

- 3NF decomposition algorithm is dependency preserving (since there is a relation for every FD in F_c)
- Decomposition is lossless
  - A candidate key (C) is in one of the relations R_i in decomposition
  - Closure of candidate key under F_c must contain all attributes in R.
  - Follow the steps of attribute closure algorithm to show there is only one tuple in the join result for each tuple in R_i

Claim: if a relation R_i is in the decomposition generated by the above algorithm, then R_i satisfies 3NF.

- Let R_i be generated from the dependency $\alpha \rightarrow \beta$
- Let $\gamma \rightarrow$ B be any non-trivial functional dependency on R_i. (We need only consider FDs whose right-hand side is a single attribute.)
- Now, B can be in either β or α but not in both.
- Consider each case separately.

Correctness of 3NF Decomposition Algorithm [2/3]

- Case 1: If *B* in β:
  - If γ is a superkey, the 2nd condition of 3NF is satisfied
  - Otherwise α must contain some attribute not in γ
  - Since $\gamma \rightarrow B$ is in F^+ it must be derivable from F_c, by using attribute closure on γ.
  - Attribute closure not have used $\alpha \rightarrow \beta$. If it had been used, α must be contained in the attribute closure of γ, which is not possible, since we assumed γ is not a superkey.
  - Now, using $\alpha \rightarrow (\beta - \{B\})$ and $\gamma \rightarrow B$, we can derive $\alpha \rightarrow B$

    (since $\gamma \subseteq \alpha\, \beta$, and $B \notin \gamma$ since $\gamma \rightarrow B$ is non-trivial)
  - Then, *B* is extraneous in the right-hand side of $\alpha \rightarrow \beta$; which is not possible since $\alpha \rightarrow \beta$ is in F_c.
  - Thus, if *B* is in β then γ must be a superkey, and the second condition of 3NF must be satisfied.

Correctness of 3NF Decomposition Algorithm [3/3]

- Case 2: *B* is in α.
  - Since α is a candidate key, the third alternative in the definition of 3NF is trivially satisfied.
  - In fact, we cannot show that γ is a superkey.
  - This shows exactly why the third alternative is present in the definition of 3NF.

Q.E.D.