Institute of Information and Communication and Technology
Bangladesh University of Engineering and Technology

ICT 5103: Database Design and Management

Lecture 7

Instructor: Samin Rahman Khan

The slides were taken from the following chapter of the book

Chapter 7: Normalization

Database System Concepts, 7th Ed.

Outline

- Features of Good Relational Design
- Functional Dependencies
- Decomposition Using Functional Dependencies
- Normal Forms
- Functional Dependency Theory
- Algorithms for Decomposition using Functional Dependencies
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Atomic Domains and First Normal Form
- Database-Design Process
- Modeling Temporal Data

Overview of Normalization

Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *department*

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

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)

A Combined Schema Without Repetition

Not all combined schemas result in repetition of information

- Consider combining relations
  - *sec_class(sec_id, building, room_number)* and
  - *section(course_id, sec_id, semester, year)*

  into one relation
  - *section(course_id, sec_id, semester, year, building, room_number)*
- No repetition in this case

Decomposition

- The only way to avoid the repetition-of-information problem in the i*n_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose

  employee(ID, name, street, city, salary)

  into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)

  The problem arises when we have two employees with the same name
- 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

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

employee

ID	*name*
⋮	
57766	*Kim*
98776	*Kim*
⋮	

name	*street*	
⋮		
Kim	*Main*	*Pe*
Kim	*North*	*Ha*
⋮		

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 Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R . That is R = $R_1 \cup R_2$
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R_1 \cup R_2$
- Formally,

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

- And, conversely a decomposition is lossy if

$$r \subset \prod_{R1}(r) \bowtie \prod_{R2}(r) = r$$

Example of Lossless Decomposition

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

A	*B*	*C*
α	1	A
β	2	B

r

A	*B*
α	1
β	2

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

B	*C*
1	A
2	B

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

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

A	*B*	*C*
α	1	A
β	2	B

Normalization Theory

- Decide whether a particular relation R is in “good” form.
- In the case that a relation R is not in “good” form, decompose it into set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world.
- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID.
  - Each student and instructor has only one name.
  - Each instructor and student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.

Functional Dependencies (Cont.)

- An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation;
- A legal instance of a database is one where all the relation instances are legal instances
- 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*.

Functional Dependencies Definition

- Let R be a relation schema

  $\alpha \subseteq R$ *and* $\beta \subseteq R$

- The **functional dependency**

  $\boldsymbol{\alpha \rightarrow \beta}$

  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 of r.

1	4
1	5
3	7

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

Closure of a Set of Functional Dependencies

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

Keys and Functional Dependencies

- 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:

  in_dep (*ID, name, salary, dept_name, building, budget*).

  We expect these functional dependencies to hold:

  $$dept\_name \rightarrow building$$

  $$ID \rightarrow building$$

  but would not expect the following to hold:

  $$dept\_name \rightarrow salary$$

Use of Functional Dependencies

- We use functional dependencies to:
  - To test relations to see if they are legal under a given set of functional dependencies.
    - If a relation r is legal under a set F of functional dependencies, we say that r **satisfies** F.
  - To 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 of *instructor* may, by chance, satisfy
    $name \rightarrow ID$.

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 $\beta \subseteq \alpha$

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless.
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

  $$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$
- A decomposition of R into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless decomposition:

    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless decomposition:

    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
- *Note:*
  - $B \rightarrow BC$

    is a shorthand notation for
  - $B \rightarrow \{B, C\}$

Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low
- When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Produced.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT **dependency preserving**.

Dependency Preservation Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)

- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$

- In the above design we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.
- To fix this, we need to decompose *dept_advisor*
  - *(s_ID, i_ID), (i_ID, dept_name)*
- Any decomposition will not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$

- Thus, the composition NOT be dependency preserving

Normal Forms

First Normal Form

- 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)

First Normal Form (Cont.)

- 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.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

Second Normal Form

A functional dependency $\alpha \rightarrow \beta$ is called a partial dependency if there is a proper subset γ of α such that $\gamma \rightarrow \beta$; we say that β is partially dependent on α. A relation schema R is in second normal form (2NF) if each attribute A in R meets one of the following criteria:

- It appears in a candidate key.
- It is not partially dependent on a candidate key.

It can be shown that every 3NF schema is in 2NF. (Since, every partial dependency is a transitive dependency.)

Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies 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

Boyce-Codd Normal Form (Cont.)

- Example schema that is ***not*** in BCNF:

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

  because :

  - *ID→name, dept_name, salary ; dept_name→building, budget*
    - holds on *in_dep* but
    - *dept_name* is not a superkey
- When decompose *in_dept* into *instructor* and *department*
  - *instructor* is in BCNF
  - *department* is in BCNF

Decomposing a Schema into BCNF

- Let R be a schema R that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that causes a violation of BCNF.
- We decompose R into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example of *in_dep*,
  - α = *dept_name*
  - β = *building, budget*

  and *in_dep* is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - $(R - (\beta - \alpha))$ = (*ID, name, dept_name, salary*)

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)
- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- *dept_advisor* is not in BCNF
  - *i_ID* is not a superkey.
- Any decomposition of *dept_advisor* will not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$
- Thus, the composition is NOT be dependency preserving

Third Normal Form

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

  $\alpha \rightarrow \beta$ 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 Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, dept_name)
- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID* }
- We have seen before that *dept_advisor* is not in BCNF
- *R,* however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - $i\_ID \rightarrow dept\_name$ *and* *i_ID* is NOT a superkey, but:
    - { *dept_name*} – {*i_ID* } = {*dept_name* } and
    - *dept_name* is contained in a candidate key

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

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

- What is wrong with the table?
  - Repetition of information
  - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF. It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Disadvantages to 3NF.
  - We may have to use null values to represent some of the possible meaningful relationships among data items.
  - There is the problem of repetition of information.

https://www.quora.com/What-is-an-example-of-a-table-which-is-in-2NF-but-not-3NF

https://stackoverflow.com/questions/23681453/finding-a-relation-in-3nf-but-not-in-bcnf

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies.
- Decide whether a relation scheme R is in "good" form.
- In the case that a relation scheme R is not in "good" form, need to decompose it into a set of relation scheme $\{R_1, R_2, \ldots, R_n\}$ such that:
  - Each relation scheme is in good form
  - The decomposition is a lossless decomposition
  - Preferably, the decomposition should be dependency preserving.

How good is BCNF?

- 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
  - Instance of *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

Higher Normal Forms

- It is better to decompose *inst_info* into:
  - *inst_child*:

ID	*child_name*
99999	David
99999	William

  - *inst_phone:*

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

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

Functional-Dependency Theory

Functional-Dependency Theory Roadmap

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- We then develop algorithms to test if a decomposition is dependency-preserving

Closure of a Set of Functional Dependencies

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

Closure of a Set of Functional Dependencies

- We can compute F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - **Reflexive rule:** if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation rule:** if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$
  - **Transitivity rule:** if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These rules are
  - **Sound** -- generate only functional dependencies that actually hold, and
  - **Complete** -- generate all functional dependencies that hold.

Example of F^+

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

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
  - **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
  - **Pseudotransitivity rule**:If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.
- The above rules can be inferred from Armstrong's axioms.

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

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

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

Closure of Attribute Sets

- 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* $\cup$ γ
      end

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  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 super key?
     1. Does $AG \rightarrow R$? == Is R $\supseteq$ $(AG)^+$
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is R $\supseteq$ $(A)^+$
     2. Does $G \rightarrow R$? == Is R $\supseteq$ $(G)^+$
     3. In general: check for each subset of size *n-1*

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute $\alpha^{+,}$ and check if α^{+} contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^{+}), just check if $\beta \subseteq \alpha^{+}$.
  - That is, we compute α^{+} by using attribute closure, and then check if it contains β.
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and for each $S \subseteq \gamma^{+}$, we output a functional dependency $\gamma \rightarrow S$.

Multivalued Dependencies

Multivalued Dependencies (MVDs)

- 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

- 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:

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

MVD -- Tabular representation

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

	α	β	R
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	a_j
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	b_j
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	b_j
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	a_j

MVD (Cont.)

- 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 $r(R)$

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

  then

  $< y_1, z_1, w_2 > \in r$ 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 $Y \rightarrow\rightarrow W$

Example

- 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.

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 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

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

- Let R be a relation schema, and let R1, R2, ..., Rn be a decomposition of R.
- 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 $\alpha \cap \beta = \varphi$;
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

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow\rightarrow B$
  $B \rightarrow\rightarrow HI$
  $CG \rightarrow\rightarrow H \}$
- 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)

Additional issues

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*,
    - 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 a *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

Modeling Temporal Data

- **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 holding, because the address varies over time
- A **temporal functional dependency** X → Y holds on schema *R* if the functional dependency X □ Y holds on all snapshots for all legal instances r (*R*).

Modeling Temporal Data (Cont.)

- 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