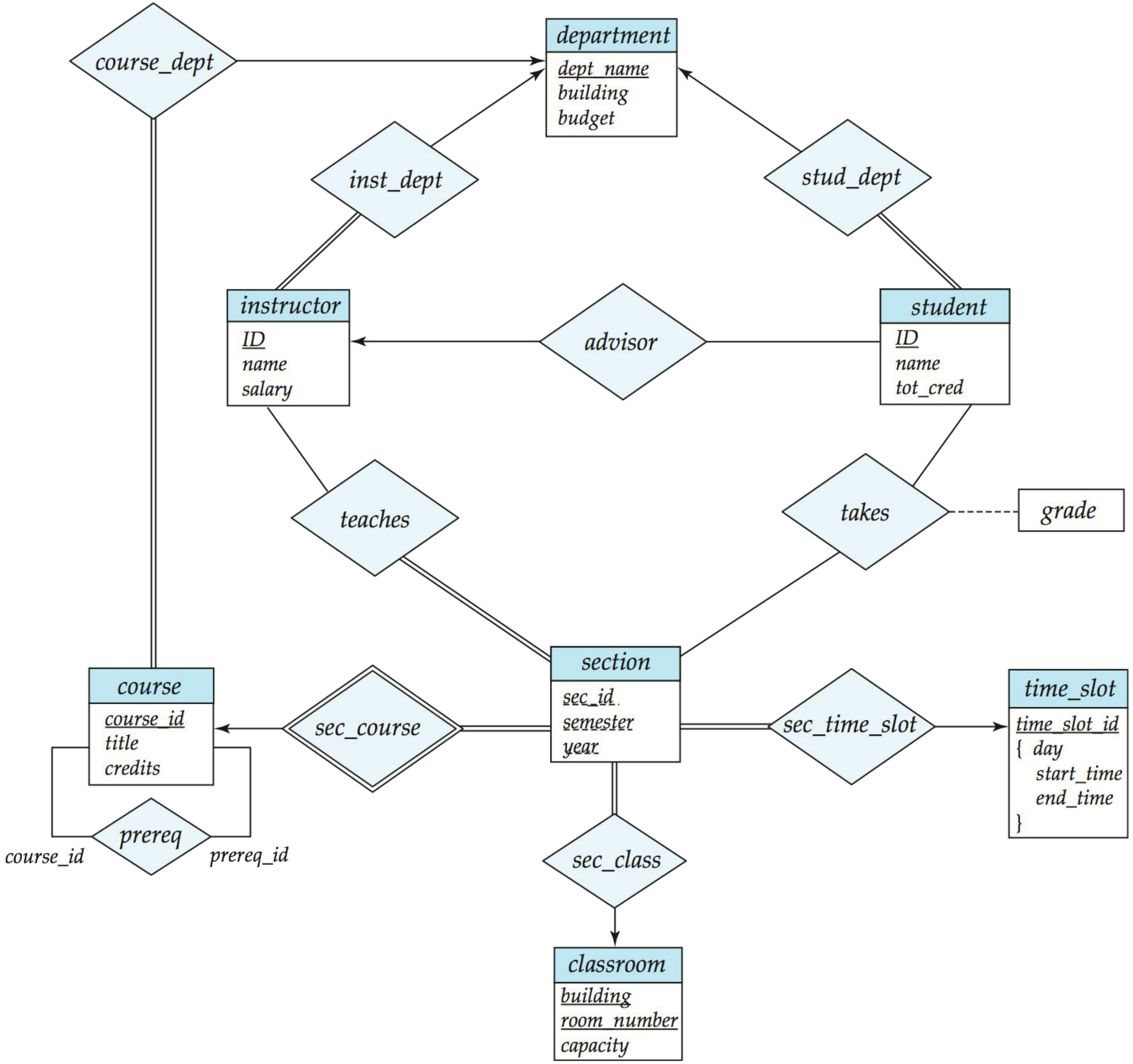
Chapter 7: Relational Database Design

Outline

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

E-R Diagram for a University Enterprise



Reduce to Relational Schemas

- Relational Schemas of a University Enterprise

classroom(<u>building</u>, <u>room_number</u>, capacity)
department(<u>dept_name</u>, building, budget)
course(<u>course_id</u>, title, dept_name, credits)
instructor(<u>ID</u>, name, dept_name, salary)
section(<u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, building, room_number, time_slot_id)
teaches(<u>ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>)
student(<u>ID</u>, name, dept_name, tot_cred)
takes(<u>ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, grade)
advisor(<u>s_ID</u>, i_ID)
time_slot(<u>time_slot_id</u>, <u>day</u>, <u>start_time</u>, end_time)
prereq(<u>course_id</u>, <u>prereq_id</u>)

- What about combining **instructor** and **department** ?
- What about combining **instructor** and **teaches**?
- What about splitting **student** → **S1(ID, department)** , **S2(name, tot_cred)?**

Combine Schemas?

- Suppose we combine **instructor** and **department** into **inst_dept**
- Result is possible repetition of information

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

- Pitfalls of the “bad” relations
  - **Information repetition (信息重复)**
  - **Insertion anomalies (插入异常)**
  - **Update difficulty (更新困难)**

A Combined Schema Without Repetition

- Consider combining relations
  - *student(<u>id</u>, name, tot_cred)*
  - *stud_dept(<u>id</u>, deprt_name)*

  ➔

  - *student(<u>id</u>, name, tot_cred, deprt_name)*
- No repetition in this case

What About Smaller Schemas?

- Suppose we had started with

  inst_dept(id,name,salary,dept_name, building, budget)

- How would we know to split up (decompose) it into instructor and department?

  if there were a schema **department (*dept_name, building, budget*),**

  then ***dept_name*** would be a candidate key"

- Denote as a functional dependency:

  id → name, salary, dept_name

  dept_name → building, budget

- In ***inst_dept***, 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***

What About Smaller Schemas?

- Not all decompositions are good. Suppose we decompose

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

  →

  employee1 **(*ID*, *name*)**

  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

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

employee

ID	*name*
⋮	
57766	Kim
98776	Kim
⋮	

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
⋮				

Example of Lossless-Join Decomposition

- Lossless join decomposition
- Decomposition of $R = (\underline{A}, B, C)$

  $R_1 = (\underline{A}, B)$, $R_2 = (\underline{B}, C)$

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

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

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

A	*B*	*C*
α	1	A
β	2	B

Lossless-join Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R . That is R = R_1 U 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 U R_2. Formally, $\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$
- And, conversely a decomposition is lossy if

  $$r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$$

  Note: more tuples implies more uncertainty (less information).

- A decomposition of R into R1 and R2 is lossless join if at least one of the following dependencies holds:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

First Normal Form

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic.
- 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
- 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.

Goal — Devise a Theory for the Following

- 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 a set of relations {R1, R2, ..., Rn} such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies
- Normal Forms(NF):
  - 1NF → 2NF → 3NF→ **BCNF**→ 4NF

Functional 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 are uniquely identified by their ID.
  - Each student has only one name.
  - Each student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.
  - Above constraints are captured in the ER diagram of the university database.
- 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

Functional Dependencies (Cont.)

- Functional Dependencies are 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.

  dept_name* → *building

- A functional dependency is a generalization of the notion of a key.

Functional Dependencies (Cont.)

- Let R be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

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 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 (Cont.)

- K is a superkey for relation schema R **if and only if K → R**
- K is a candidate key for R if and only if
  - ***K*** **→ *R***, and
  - **for no α ⊂ *K*, α → *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*).**

  We expect these functional dependencies to hold:

  dept_name* → *building

  ID* → *building

  but would not expect the following to hold:

  dept_name* → *salary

Functional Dependencies (Cont.)

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

Closure(闭包) 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 F+.
- F^+ is a superset of *F*.
- $F=\{A \rightarrow B, B \rightarrow C\}$, what is F^+?

  $F^+=\{A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow B, AB \rightarrow C, \ldots \}$

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,自反率)**
  - if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$ **(augmentation，增补率)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity，传递率)**
- These rules are
  - **Sound（正确有效的）** (generate only functional dependencies that actually hold), and
  - **Complete（完备的）** (generate all functional dependencies that hold).

Example

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

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - If $\alpha \rightarrow \beta$ holds *and* $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union，合并**)
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition，分解**)
  - If $\alpha \rightarrow \beta$ holds *and* $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity，伪传递**)
- The above rules can be inferred from Armstrong’s axioms.

Exercise 1

- The functional dependency $\alpha\ \gamma \rightarrow \beta\ \gamma$ holds on **R(**α **,** β **,** γ **),** prove that $\alpha\ \gamma \rightarrow \beta\ \gamma$ is equivalent with $\alpha\ \gamma \rightarrow \beta$.

  Duplicated attributes can be removed from right side of a functional dependency.

  Prove:

  $\alpha\ \gamma \rightarrow \beta\ \gamma$, $\beta\ \gamma \rightarrow \beta$ ==> $\alpha\ \gamma \rightarrow \beta$

  (reflexivity)

  $\alpha\ \gamma \rightarrow \beta$ ==> $\alpha\gamma\ \gamma \rightarrow \beta\ \gamma$ ==> $\alpha\ \gamma \rightarrow \beta\ \gamma$

  (Augmentation)

Closure of Attribute Sets

- Given a set of attributes ***a***, define the closure of ***a*** under F (denoted by ***a+***) as the set of attributes that are functionally determined by ***a*** under F
- $R(A,B,C,D)$ $\quad F=\{A \to B, B \to C, B \to D\}$

  $A^+=ABCD$

  $B^+=BCD$

  $C^+=C$

- Algorithm to compute α^+, the closure of α under F

  result := α;
  while (changes to *result*) **do**
      for each $\beta \to \gamma$ **in** *F* **do**
        begin
          if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
        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 $(AG)^+ \supseteq R$
  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

- There are several uses of the attribute closure algorithm:
- **Testing for superkey:**
  - To test if α is a superkey, we compute α^{+}, 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 $\gamma \rightarrow S$.

Uses of Attribute Closure

- Computing closure of F:

R(<u>A</u>,B,C), F={A→B, B→C}

computing:

A^+=ABC, B^+=BC, C^+=C

$(AB)^+$=ABC, $(AC)^+$=ABC, $(BC)^+$=BC, $(ABC)^+$=ABC

F⇔{A→ABC, B→BC, C→C, AB→ABC, AC→ABC, BC→BC, ABC→ABC}

⇔{A→A, A→B,A→C, A→AB,A→AC,A→BC,A→ABC

B→B,B→C, B→BC,

C→C,

AB→A, AB→B,AB→C, AB→AB,AB→AC,AB→BC,AB→ABC,

AC→A, AC→B,AC→C, AC→AB,AC→AC,AC→BC,AC→ABC,

BC→B, BC→C, BC→BC,

ABC→A, ABC→B,ABC→C, ABC→AB, ABC→AC, ABC→BC, ABC→ABC}

=F^+

Canonical Cover（正则覆盖）

- Sets of functional dependencies may have redundant dependencies 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 functional dependency may be redundant
  - E.g.: on RHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}$

    can be simplified to

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

    .A→CD=> *A→D*

    .A→B,B→C => *A→C* , A→D =>*A→CD*
  - E.g.: on LHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ can be simplified to $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$

    .A→B,B→C => A→C=> A→AC, AC→D =>A→D

    .A→D => *AC→CD* => AC→D
- Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

Extraneous Attributes(无关属性)

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ 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 functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- 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

Canonical Cover

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

- To compute a canonical cover for F:

  repeat

  Use the union rule to replace any dependencies in *F*

  $\alpha_1 \to \beta_1$ and $\alpha_1 \to \beta_2$ with $\alpha_1 \to \beta_1 \beta_2$

  Find a functional dependency $\alpha \to \beta$ with an extraneous attribute either in α or in β

  If an extraneous attribute is found, delete it from $\alpha \to \beta$

  until *F* does not change

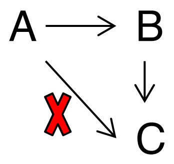
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Computing a 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* $A \rightarrow BC$
  - 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 dependencies
    - Yes: in fact, $B \rightarrow C$ is already present!
  - 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 dependencies
    - Yes*:* using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C.
      - Can use attribute closure of A in more complex cases
- The canonical cover is: $\{ A \rightarrow B$
  $B \rightarrow C \}$

Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$



- The canonical cover is: $\{A \rightarrow B, \quad B \rightarrow C\}$

Exercise 2

- For relation schema R(A,B,C,D,E) with functional dependencies set F={A→BC, AD→E , B→C, D→E}.
  a) Compute canonical cover Fc.
  b) Compute $(AE)^+$
  c) Find all candidate keys of R

Answers

a) Fc={A→B, B→C, D→E}.

b) $(AE)^+$ =ABCE

c) Candaidate key: AD

$$A \longrightarrow B \longrightarrow C$$

$$D \longrightarrow E$$

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*

- Example schema not in BCNF:

  instr_dept (<u>*ID,*</u> *name, salary,* <u>*dept_name,*</u> *building, budget*)

  because ***dept_name*** $\rightarrow$ ***building, budget***
  holds on *instr_dept,* but *dept_name* is not a superkey

Decomposing a Schema into BCNF

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

  We decompose *R* into:
  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$
- In our example,
  - α = ***dept_name***
  - β = ***building, budget***

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

BCNF Decomposition Algorithm

```
result := {R };
done := false;
while (not done) do
        if (there is a schema Ri in result  that is not in BCNF)
        then begin
                let α→ β  be a nontrivial functional dependency that
                holds on Ri  such that α+ does not contain Ri  and α ∩ β  = ∅;
                result := (result – Ri ) ∪ (Ri – β) ∪ (α, β );
            end
        else done := true;
```

Note: each R_i is in BCNF, and decomposition is lossless-join.

Exercise 3

- For the relation schema R(A,B,C,D) with the functional dependencies set F={A→B, B→CD},
  a) List all candidate keys of the relation.
  b) Decompose the relation into a collection of BCNF relations. The decomposition must be lossless-join.

$$\begin{tikzcd} A \arrow[r] & B \arrow[r] \arrow[dr] & C \\ & & D \end{tikzcd}$$

- Answer:
  a) candidate keys: A
  b) R1=($\underline{B}$,C,D), R2=($\underline{A}$,B).

     F1={B→CD}, F2={A->B}

     The decomposition is lossless-join ,

     because $R_1 \cap R_2 = (B)$*, and B is a candidate key of R1.*

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is ***dependency preserving* (*保持依赖*).**

  （如果通过检验单一关系上的函数依赖，就能确保所有的函数依赖成立，那么这样的分解是依赖保持的）

  （或者，原来关系R上的每一个函数依赖，都可以在分解后的单一关系上得到检验或者推导得到。）
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as third normal form.

Dependency Preservation

- Let F_i be the set of all functional dependencies in F^+ that include only attributes in R_i. *(F_i: the restriction of F on Ri)*
  - A decomposition is **dependency preserving**, if

    $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

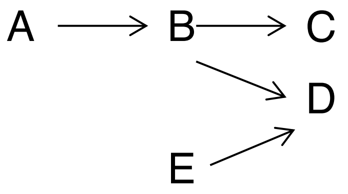
Example

- $R = (A, B, C)$
  $F = \{A \to B$
  $B \to C\}$
  Key $= \{A\}$

  $A \longrightarrow B \longrightarrow C$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - F1={$A \to B$ }， F2={$B \to C$}
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving, since $(F1 \cup F2)^+ = F^+$
- Decomposition $R_1 = (A, B)$, $R_2 = (A, C)$
  - F1={$A \to B$ }， F2={$A \to C$}
  - R1 and R2 in BCNF
  - Lossless-join decomposition
  - Not dependency preserving， since $(F1 \cup F2)^+ <> F^+$

Exercise 4

- For the relation schema R(A,B,C,D,E) with the functional dependencies set F={A→B, B→CD, E→D },
  a) List all candidate keys of the relation.
  b) Decompose the relation into a collection of BCNF relations. The decomposition must be lossless-join.
  c) Whether the decomposition of (b) is dependency preserving or not?

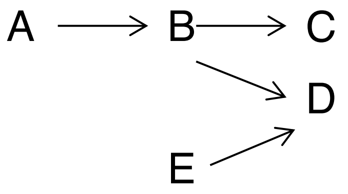


- Answer:
  a) candidate keys: AE
  b) R1(E,D), R2(B,C), R3(A,B), R4(A,E)
  c) Above decomposition is not dependency preserving, because B→D cannot be inferred by all functional dependencies holds on R1, R2, R3, and R4.

F1=(E→D} F2={B→C} F3={A→B} F4={AE→AE}

Exercise 4

- For the relation schema R(A,B,C,D,E) with the functional dependencies set F={A→B, B→CD, E→D },
  - a) List all candidate keys of the relation.
  - b) Decompose the relation into a collection of BCNF relations. The decomposition must be lossless-join.
  - c) Whether the decomposition of (b) is dependency preserving or not?



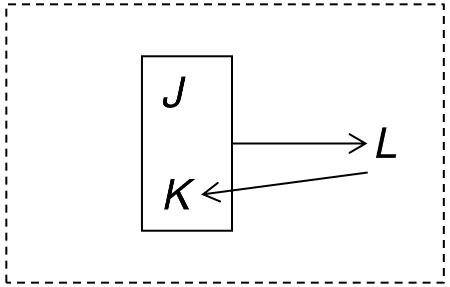
- Another Answer:
  - a) candidate keys: AE
  - b) R1(B,C,D), R2(A,B), R3(A,E)
  - c) Above decomposition is not dependency preserving, because E→D cannot be inferred by all functional dependencies holds on R1, R2, R3.

    F1=(B→CD}   F2={A→B}   F3={AE→AE}

BCNF and Dependency Preservation

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

  - $R = (J, K, L)$
    $F = \{JK \rightarrow L$
    $L \rightarrow K\}$



  - *Two candidate keys = JK and JL*
  - R is not in BCNF
  - Decomposition R into R1(L,K) , R2(J,L)
  - All decomposition of R will fail to preserve

$$JK \rightarrow L$$

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

Third Normal Form: Motivation

- 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
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

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.

3NF Example

- Relation *dept_advisor*:
  - dept_advisor (s_ID, i_ID, dept_name)

    F = {s_ID, dept_name $\rightarrow$ i_ID, i_ID $\rightarrow$ dept_name}

  - Two candidate keys: s_ID, dept_name, and i_ID, s_ID
  - R is in 3NF
    - s_ID, dept_name $\rightarrow$ i_ID
      - s_ID, dept_name is a superkey
    - i_ID $\rightarrow$ dept_name
      - dept_name is contained in a candidate key

Redundancy in 3NF

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

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)
  - *(i_ID, dept_name)*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).
  - *(i_ID, dept_namel)* if there is no separate relation mapping instructors to departments

3NF Decomposition Algorithm

```
Let Fc be a canonical cover for F;
i := 0;
for each  functional dependency α → β in Fc do
      begin
                 i := i  + 1;     Ri  := α β
      end

if none of the schemas Rj, 1 ≤ j  ≤ i contains a candidate key for R
then begin
                i := i  + 1;
                Ri := any candidate key for R;
         end

/* Optionally, remove redundant relations */
repeat
     if any schema Rj is contained in another schema Rk
     then /* delete Rj  */
          begin
             Rj = Ri ;   i=i-1;
until no more Rj s can be deleted
return (R1, R2, ..., Ri)
```

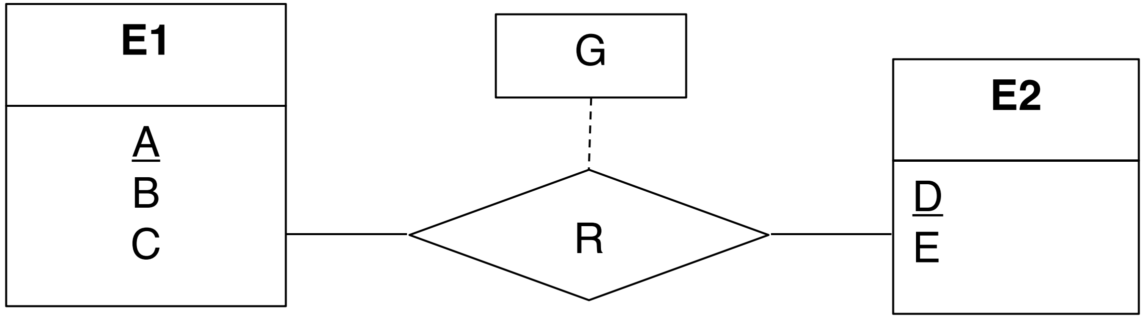
Comparison of BCNF and 3NF

- 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.

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, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in good form (i.e., BCNF or 3NF)
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving.

E-R Modeling and Normal Forms

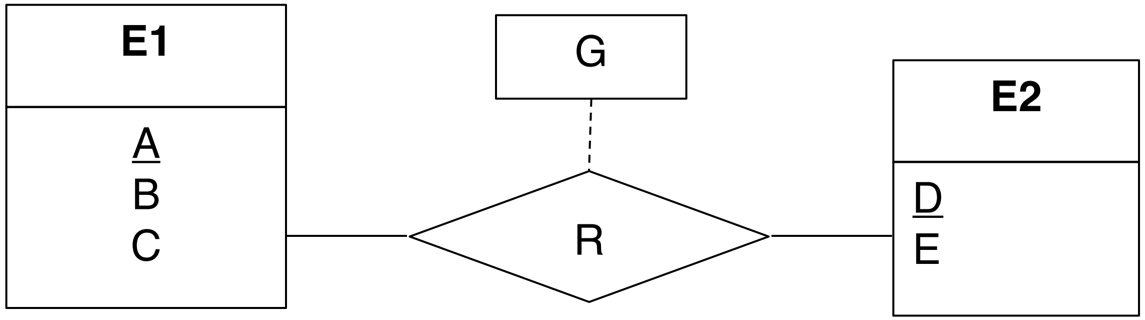


- Above is an E-R diagram that depicts a many-to-many relationship R between two entity sets E1 and E2. The candidate key of E1 and E2 are underlined in the diagram. If the E-R diagram is transform to a relation schema S(A, B, C, D, E, G).

  Please answer following questions:

  1. Give the no-trivial functional dependency set F that holds on S.
  2. List the candidate key(s) of S.
  3. Prove that S is not in BCNF.
  4. Explain the issues exist with S.
  5. Prove that decomposing of S into three relation schemas R1(A,B,C), R2(D,E) and R3(A,D,G) is lossless join.

E-R Modeling and Normal Forms



- Please answer following questions:
  1. Give the no-trivial functional dependency set F that holds on S.

     F={A→BC, D→E, AD→G}
  2. List the candidate key(s) of S.

     AD
  3. Prove that S is not in BCNF.

     A→BC, A is not a key.
  4. Explain the issues exist with S.

     Information repetition, Insertion anomalies, Update difficulty
  5. Prove that decomposing of S into three relation schemas

     R=R1(A,B,C) $\bowtie$ (R2(D,E) $\bowtie$ R3(A,D,G))

How good is BCNF?

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

  inst_info (<u>*ID, child_name, phone*</u>)

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

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

inst_info

How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- 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)

How good is BCNF? (Cont.)

- Therefore, it is better to decompose inst_info into:

inst_child

ID	*child_name*
99999	David
99999	David
99999	William
99999	Willian

inst_phone

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

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF).

Multivalued Dependencies

Multivalued Dependencies

- 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 (MVDs)

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

$$\alpha \to\to \beta$$

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:

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

$\rightarrow\!\!\!\!\twoheadrightarrow$

MVD (Cont.)

- 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	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Example

- 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)

$$< y1, z1, w1 > \in r \text{ and } < y1, z2, w2 > \in r$$

  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 $Y \rightarrow\rightarrow W$

Example (Cont.)

- 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) $Z_1 = Z_2$

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.

$\rightarrow\rightarrow$

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

$\rightarrow\!\!\!\rightarrow$

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

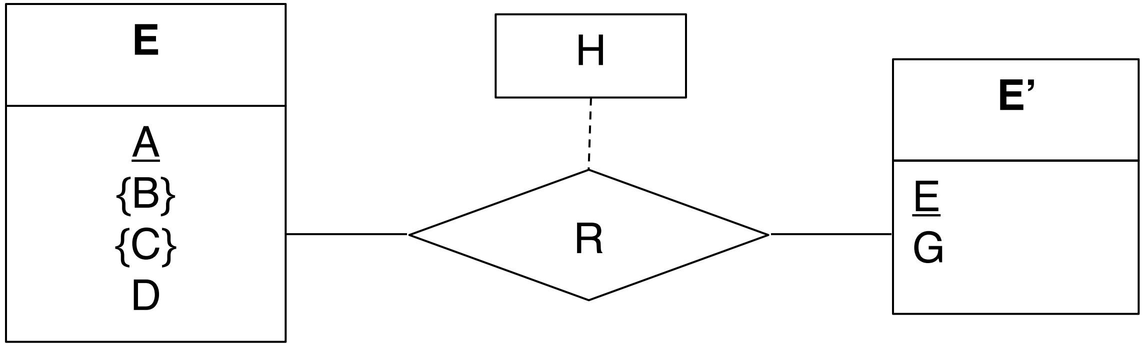
Note: each R_i is in 4NF, and decomposition is lossless-join

$\twoheadrightarrow$

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)

E-R Modeling and Normal Forms

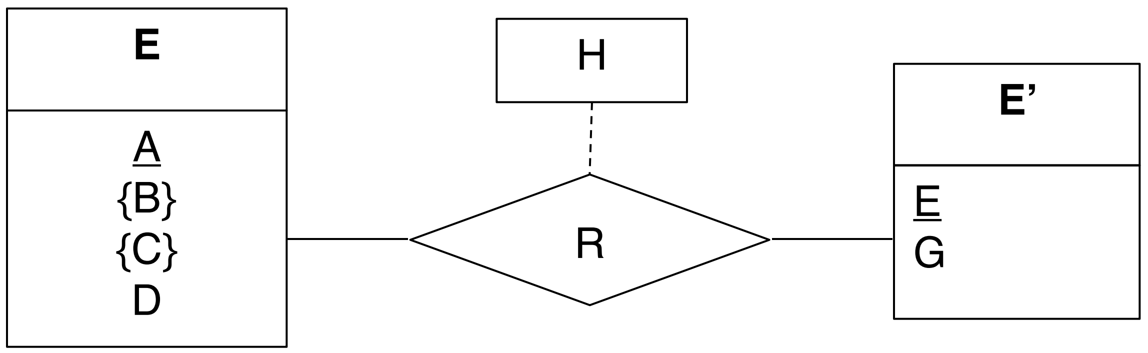


- Above is an E-R diagram that depicts a many-to-many relationship R between two entity sets E and E’. The candidate key of E and E’ are underlined in the diagram. B and C are multivalued attributes of Entity E. If the Entity E is transform to a relation schema E(A,B,C,D).

  Please answer following questions:

  1. Give the no-trivial functional dependency and multivalued dependency set D that holds on E.
  2. List the candidate key(s) of E.
  3. Prove that E is not in 4NF.
  4. Explain the issues exist with E.
  5. Please decompose E into 4NF relations.

E-R Modeling and Normal Forms



- Please answer following questions:
  1. Give the no-trivial functional dependency and multivalued dependency set D that holds on E.

     D = {A→D, A→→B, A→→C}
  2. List the candidate key(s) of E ： ABCD
  3. Prove that E is not in 4NF.

     A→→B is no-trivial , and A is not a key.
  4. Explain the issues exist with E.

     Information repetition, Insertion anomalies, Update difficulty
  5. Please decompose E into 4NF relations.

     R1（A，B），R2（A， C), R3(A, D)

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

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 : 3 approaches to represent the same information:
  - *1. earnings (company_id, year, amount)*
  - *2. ernings_2012(company_id, earnings)*

    earnings_2013(company_id, earnings)

    earnings_2014(company_id, earnings)
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *3. company_year (company_id, earnings_2012, earnings_2013, earnings_2014)*
    - 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 tools1

End of Chapter 7