DATABASE SYSTEM PRINCIPLE
- RELATIONAL DATABASE DESIGN THEORY
PART A: NORMAL FORM

李旭东 Li-Xudong
LEEXUDONG@NANKAI.EDU.CN
NANKAI UNIVERSITY

OBJECTIVES

- Features of Good Relational Design
- First Normal Form
- Decomposition Using Functional Dependencies
- BCNF, 3NF, 2NF
- Decomposition Using Multivalued Dependencies
- 4NF, 5NF
- Database-Design Process
- Modeling Temporal Data

FEATURES OF GOOD RELATIONAL DESIGN

FEATURES OF GOOD RELATIONAL DESIGN

- Design Alternative:
  - Larger Schemas、 Smaller Schemas

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

COMBINE SCHEMAS? LARGE SCHEMAS

- Suppose we combine *instructor* and *department* into *inst_dept*
  - *(No connection to relationship set 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

- Inconsistency
- Data redundancy数据冗余
- Insertion anomaly插入异常
- Deletion anomaly删除异常
- Update anomaly更新异常
- inability to represent information无法表示信息

A COMBINED SCHEMA WITHOUT REPETITION

- 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

WHAT ABOUT SMALLER SCHEMAS? 1/2

- 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*?
- Write a rule "if there were a subschema (*dept_name, building, budget*), then *dept_name* would be a candidate key"

WHAT ABOUT SMALLER SCHEMAS? 2/2

- Denote as a **functional dependency**: 函数依赖

  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*

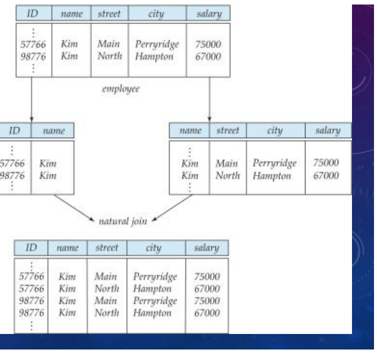
WHAT ABOUT SMALLER SCHEMAS? 1/3

- Not all decompositions are good.
- Suppose we decompose
  employee(ID, name, street, city, salary) into

  employee1 (*ID, name*)

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

WHAT ABOUT SMALLER SCHEMAS? 2/3 : A LOSSY DECOMPOSITION



WHAT ABOUT SMALLER SCHEMAS? 3/3

- Not all decompositions are good.
- Suppose we decompose
  employee(ID, name, street, city, salary) into

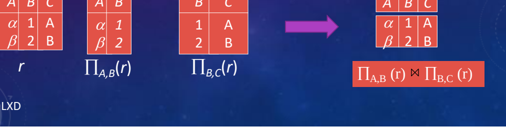
  employee1 (*ID, name*)

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

EXAMPLE OF LOSSLESS-JOIN DECOMPOSITION

- **Lossless join decomposition无损连接之分解**
- Decomposition of $R = (A, B, C)$

  $R_1 = (A, B)$ $R_2 = (B, C)$



NORMAL FORM OF RELATIONAL SCHEMAL

FIRST NORMAL FORM

- Domain is atomic原子的 if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Address: street_city_state_zip

FIRST NORMAL FORM第一范式

- 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

FIRST NORMAL FORM

- 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 *CS2018012* 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.

EXAMPLE OF NORMAL FORM

- Does the following schema belong to 1NF?
- SLC(<u>Sid</u>, Sdept, Dloc, <u>Cid</u>, Cscore)
  - Sid: student id, Sdept: student's department
  - Cid:Course id, Dloc: location of department
  - Cscore: student's score of course

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 $\{R_1, R_2, ..., R_n\}$ such that
  - (1) each relation is in good form
  - (2) the decomposition is a **lossless-join decomposition**

GOAL :DEVISE A THEORY FOR THE FOLLOWING

- Decide whether a particular relation R is in “good” form.
- Our theory is based on:
  - **functional dependencies**函数依赖
  - **multivalued dependencies**多值依赖
  - …

FUNCTIONAL DEPENDENCIES函数依赖

- 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概化 of the notion of a *key*

FUNCTIONAL DEPENDENCIES (CONT.)

- Let R be a relation schema

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

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on(成立) 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,

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

FUNCTIONAL DEPENDENCIES (CONT.)

- 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: SUPERKEY V.S. CANDIDATE KEY

- 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 $\alpha \subset K$, $\alpha \rightarrow R$
- *For any candidate key K, if* $\alpha \in K$, *then* α is prime attribute (主属性), otherwise α is nonprime attribute (非主属性),

FUNCTIONAL DEPENDENCIES V.S. SUPERKEY

- 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:

  $$dept\_name \rightarrow building$$

  and $$ID \rightarrow building$$

  but would not expect the following to hold:

  $$dept\_name \rightarrow salary$$

USE OF FUNCTIONAL DEPENDENCIES 1/2

- We use functional dependencies to:
  - (1) test relations to see if(判定) 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(满足) *F*.
  - (2) specify说明 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*.

©LXD

USE OF FUNCTIONAL DEPENDENCIES 2/2

- 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 $name \rightarrow ID$

©LXD

FUNCTIONAL DEPENDENCIES: TRIVIAL平凡的

- *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 $\beta \subseteq \alpha$
- *Nontrivial* *非平凡的*

©LXD

FUNCTIONAL DEPENDENCIES: FULL/PARTIAL FD

- Full FD 完全函数依赖
  - *IF* $X \rightarrow Y$, *and* $X' \not\rightarrow Y \quad (\forall X' \subset X)$
  - *THEN* $X \xrightarrow{F} Y$
- Partial FD 部分函数依赖
  - $X \xrightarrow{P} Y$

Takes:
(<u>S_ID</u>, <u>C_ID</u>, score)

$(S\_ID, C\_ID) \xrightarrow{F} score$

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

$(ID, dept\_name) \xrightarrow{P} building$

©LXD

FUNCTIONAL DEPENDENCIES: TRANSITIVE FD

- Given a set *F* of functional dependencies
  - If $A \rightarrow B$ and $B \rightarrow C$, and $B \not\subset A$, and $C \not\subset B$
  - then we can infer that $A \rightarrow C$
- we can call $A \rightarrow C$ *is a transitive FD (传递式函数依赖)*
- We also call $A \rightarrow C$ *is logically implied*(逻辑蕴涵) by *F*

©LXD

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逻辑蕴涵 by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by F^+.
- F^+ is a superset of *F*.

©LXD

BOYCE-CODD NORMAL FORM: BCNF巴斯范式

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

$\alpha \to \beta$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

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

BOYCE-CODD NORMAL FORM: BCNF

Example schema *not* in BCNF:

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

note: ID i.e. instructor id

because *dept_name* → *building*, *budget*

holds on *instr_dept*, but *dept_name* is not a superkey

In fact: BCNF eliminates all redundancy that can be discovered based on functional dependencies

DECOMPOSING A SCHEMA INTO BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \to \beta$ causes a violation(违规) of BCNF.

  We decompose *R* into:
  - R1: ($\alpha \cup \beta$)
  - R2: ($R - (\beta - \alpha)$)

DECOMPOSING A SCHEMA INTO BCNF

- In our example

  instr_dept (<u>*ID*</u>, *name*, *salary*, *dept_name*, *building*, *budget*)
  - α = *dept_name*
  - β = *building*, *budget*

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

EXAMPLE OF NORMAL FORM

- Does the following schema belong to BCNF?
- SLC(<u>Sid</u>, Sdept, Dloc, <u>Cid</u>, Cscore)
  - Sid: student id, Cid:Course id, Dloc: Dept location

$(\mathrm{Sid}, \mathrm{Cid}) \xrightarrow{F} Cscore$

$\mathrm{Sid} \to \mathrm{Sdept}, \quad (Sid, Cid) \xrightarrow{P} Sdept$

$Sid \to \mathrm{D}loc, \quad (Sid, Cid) \xrightarrow{P} \mathrm{D}loc,$

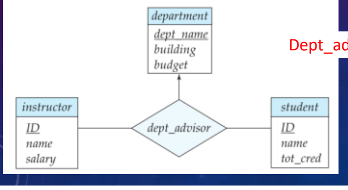
$Sdept \to \mathrm{D}loc$

DEPENDENCY PRESERVATION(保持依赖)

- *dependency preserving*
  - 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*.

EXAMPLE: DEPENDENCY PRESERVATION (1/3)

- Suppose: an instructor can be associated with only a single department and a student may have more than one advisor, but at most one from a given department



Dept_advisor (s_ID , i_ID , dept_name)

EXAMPLE: DEPENDENCY PRESERVATION (2/3)

- The following functional dependencies hold on dept_advisor:
  - (1) i_ID → dept_name
  - (2) s_ID , dept_name → i_ID

In fact: dept_advisor is not in BCNF because i_ID is not a superkey

EXAMPLE: DEPENDENCY PRESERVATION (3/3)

- If BCNF decomposition:
  - (i_ID , dept_name), (s_ID , i_ID)
  - Both the above schemas are BCNF

This design makes it computationally hard to enforce this functional dependency (2)：
(2) s_ID , dept_name → i_ID
So it is **not dependency preserving**

BCNF & DEPENDENCY PRESERVATION

- Constraints, including functional dependencies, are ***costly*** to check in practice unless they pertain to only one relation
- It is not always possible to achieve both BCNF and dependency preservation
- So, we consider a weaker normal form, known as *third normal form*.

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: Third Normal Form
  - Allows some redundancy (with resultant problems)
  - 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第三范式

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

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

THIRD NORMAL FORM第三范式

- 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
  - Preserves function dependency of key attributes

THIRD NORMAL FORM第三范式: EXAMPLE

Dept_advisor (s_ID , i_ID , dept_name)

- The following functional dependencies hold on dept_advisor:
  - (1) i_ID → dept_name
  - (2) s_ID , dept_name → i_ID
- dept_advisor is not in BCNF because i_ID is not a superkey
- But dept_advisor is in third normal form
  - Because α=i_ID, *β=dept_name, so β- α= 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)

- for Dept_advisor (s_ID , i_ID , dept_name): (*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).

- for Dept_advisor :(*i_ID, dept_name)* if there is no separate relation mapping instructors to departments

EXAMPLE OF NORMAL FORM

- Does the following schema belong to 3NF?
- SLC(Sid, Sdept, Dloc, Cid, Cscore)
  - Sid: student id, Cid:Course id, Dloc: Dept location

$$(\mathrm{Sid}, \mathrm{Cid}) \xrightarrow{F} Cscore$$

$$\mathrm{Sid} \rightarrow \mathrm{Sdept}, \quad (Sid, Cid) \xrightarrow{P} Sdept$$

$$Sid \rightarrow \mathrm{Dloc}, \quad (Sid, Cid) \xrightarrow{P} \mathrm{Dloc},$$

$$Sdept \rightarrow \mathrm{Dloc}$$

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.

THE SECOND NORMAL FORM

- A relation schema R is in second normal form (2NF) if each attribute A in R meets one of the following criteria:
- (1) It appears in a candidate key
- (2) It is not partially dependent on a candidate key
  - 即非主属性完全函数依赖于任何一个候选码

EXAMPLE OF NORMAL FORM

- Does the following schema belong to 2NF?
- SLC(<u>Sid</u>, Sdept, Dloc, <u>Cid</u>, Cscore)
  - Sid: student id, Cno:Course id, Dloc: Dept location

$$(\text{Sid, Cid}) \xrightarrow{F} Cscore$$

$$\text{Sid} \rightarrow \text{Sdept}, \quad (Sid, Cid) \xrightarrow{P} Sdept$$

$$Sid \rightarrow \text{D}loc, \quad (Sid, Cid) \xrightarrow{P} \text{D}loc,$$

$$Sdept \rightarrow \text{D}loc$$

GOALS OF NORMALIZATION 1/3

- 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
  - (1) each relation scheme is in good form
  - (2) the decomposition is a **lossless-join** decomposition
  - (3) Preferably, the decomposition should be **dependency preserving**

DESIGN GOALS FOR DB 2/3

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - (prefer) Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF

DESIGN GOALS FOR DB 3/3

- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.
  - 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 functional dependency whose left hand side is not a key.

HOW GOOD IS BCNF?

- There are db schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation
  - An instructor can have many children and phone number, the phone numbers may be shared by multiple people.
  - ID->child_name, ID->phone
- If we combine the above two schemas into one:

inst_info (ID, child_name, phone)

HOW GOOD IS BCNF?

- Consider a relation

inst_info (ID, child_name, phone)

This belongs to BCNF

- 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

HOW GOOD IS BCNF? (CONT.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies插入异常
  - 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)

©LXD

HOW GOOD IS BCNF? (CONT.)

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

inst_child

ID	*child_name*
99999	David
99999	Willian

inst_phone

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

This is in 4NF

©LXD

MULTIVALUED DEPENDENCIES多值依赖

- Suppose we record names of children, and phone numbers for instructors, this decomposition is in 4NF:
  - *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 also in BCNF!

©LXD

MULTIVALUED DEPENDENCIES
&
THE FOURTH AND THE FIFTH NORMAL FORM

©LXD

MULTIVALUED DEPENDENCIES (MVDS) 1/2

- 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:

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

©LXD

MULTIVALUED DEPENDENCIES (MVDS) 2/2

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

©LXD

MVDS 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*)

  $< y_1, z_1, w_1 > \in r$ 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 $Y \rightarrow\rightarrow W$

MVDS 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.

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 1/2

- 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
- If $\alpha \rightarrow\rightarrow \beta$, then $\alpha \rightarrow\rightarrow R - \alpha - \beta$
- In fact
  - Functional dependency: equality-generating dependency
  - Multivalued dependency: tuple-generating dependency

函数依赖只是多值依赖的特例而已

THEORY OF MVDS 2/2

- 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.

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:
  - (1) $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - (2) 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;

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

4NF DECOMPOSITION ALGORITHM: 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)

4NF DECOMPOSITION ALGORITHM: EXAMPLE

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

  …

  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)

FIFTH NORMAL FORM: 5NF

- Example: agents, companies, products
- If agents represent companies, companies make products, and agents sell products, then we might want to keep a record of which agent sells which product for which company

AGENT	COMPANY	PRODUCT
Smith	Ford	Car
Smith	GM	Truck

FIFTH NORMAL FORM: 5NF

- But suppose that a certain rule is in effect:
  - If an agent sells a certain product and he represents the company making that product, then he sells that product for that company

AGENT	COMPANY	PRODUCT
Smith	Ford	Car
Smith	Ford	Truck
Smith	GM	Car
Smith	GM	Truck
Jones	Ford	Car

have some redundancies

FIFTH NORMAL FORM: 5NF

- The following decomposition is in 5NF:

OK

AGENT	COMPANY
Smith	Ford
Smith	GM
Jones	Ford

COMPANY	PRODUCT
Ford	Car
Ford	Truck
GM	Car
GM	Truck

AGENT	PRODUCT
Smith	Car
Smith	Truck
Jones	Car

FIFTH NORMAL FORM: 5NF

- One advantage of fifth normal form is that certain redundancies can be eliminated.
  - If we add a new agent who sells x products for y companies, where each of these companies makes each of these products, we have to add x+y new records to the normalized form, but x*y new records to the unnormalized form.
- Fifth normal form does not differ from fourth normal form unless there exists a symmetric constraint such as the rule about agents, companies, and products.
- Fifth normal form is also called project-join normal form投影-连接范式

SUMMARY OF NORMAL FORM

SUMMARY OF NORMAL FORM

1NF

↓ 消除**非主属性**对码的部分函数依赖

2NF

↓ 消除**非主属性**对码的传递函数依赖

3NF

↓ 消除**主属性**对码的部分和传递函数依赖

BCNF

↓ 消除非平凡且非函数依赖的**多值**依赖

4NF

↓

5NF

FURTHER NORMAL FORMS

- A class of even more general constraints, leads to a normal form called domain-key normal form(DKNF).
- 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

OVERALL DATABASE DESIGN PROCESS

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* → *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

NAMING OF ATTRIBUTES AND RELATIONSHIPS

- A desirable feature of a database design is the unique-role assumption, which means that each attribute name has a unique meaning in the database.
  - This prevents us from using the same attribute to mean different things in different schemas.

DENORMALIZATION FOR PERFORMANCE 1/2

- 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 a materialized view defined as: *course* *prereq*
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

DENORMALIZATION FOR PERFORMANCE 2/2

- (continue…)
- For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*
- Alternative 2: 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

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

MODELING TEMPORAL DATA

MODELING TEMPORAL DATA

- **Temporal data时态数据** have an association time interval during which the data are *valid.*
- A **snapshot快照** 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

MODELING TEMPORAL DATA

- Adding a temporal component results in functional dependencies like

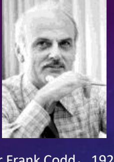
  ID → street, city

  not to hold, 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*).

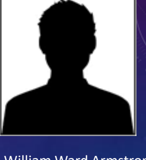
MODELING TEMPORAL DATA: EXAMPLE

- 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



Edgar Frank Codd，1923—2003



William Ward Armstron

BIBS

- Codd E F. A Relational Model for Large Shared Data Banks[J]. Communications of the Acm, 1970, 13(1):377--387. {1NF, 2NF, 3NF}
- Codd E F. Further Normalization of the Data Base Relational Model[J]. Data Base Systems, 1971. {BCNF}
- Biskup J, Dayal U, Bernstein P A. Synthesizing independent database schemas[C]// ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June. DBLP, 1979:143-151.
  - {a lossless dependency-preserving decomposition into 3NF}

BIBS

- Fagin R. Multivalued Dependencies And A New Normal Form For Relational Databases[J]. Acm Transactions on Database Systems, 1977, 2(3):262--278. {4NF}
- Armstrong W W. Dependency Structures of Data Base Relationships[C]// IFIP Congress. 1974:580-583. {Armstrong's axioms }
- Beeri C. A complete axiomatization for functional and multivalued dependencies in database relations[C]// ACM SIGMOD International Conference on Management of Data. ACM, 1977:47-61.
- ...

数据库规范式理论有感 - LXD

盘古开天万点尘，物分类聚隐前因。
盲人摸象模型现，蚂蚁缘槐范式循。
函数关联出混沌，多值依赖解迷津。
登高渐览群山面，回首方知此道真。

SUMMARY

- Features of Good Relational Design
- First Normal Form
- Decomposition Using Functional Dependencies
- BCNF, 3NF, 2NF
- Decomposition Using Multivalued Dependencies
- 4NF, 5NF
- Database-Design Process
- Modeling Temporal Data

THANKS!

leexudong@nankai.edu.cn

Q&A?