CSIE30600/CSIEB0290
Database Systems

Lecture 11: Relational Database Design II

Outline

- Designing a Set of Relations
- Properties of Relational Decompositions
- Algorithms for Relational Database Schema
- Multivalued Dependencies and Fourth Normal Form
- Join Dependencies and Fifth Normal Form
- Inclusion Dependencies
- Other Dependencies and Normal Forms

Relational Synthesis

- **Designing database using relational synthesis (Bottom-up Design):**
  - Assumes that all possible functional dependencies are known.
  - First constructs a minimal set of FDs
  - Then applies algorithms that construct a target set of 3NF or BCNF relations.
  - Additional criteria may be needed to ensure the *set of relations* in a relational database are satisfactory.

Design Goals

- **Lossless join property** (a must)
  - Algorithm 15.3 tests for general losslessness.
- **Dependency preservation property**
  - Algorithm 15.5 decomposes a relation into BCNF components by sacrificing the dependency preservation.
- **Additional normal forms**
  - 4NF (based on multi-valued dependencies)
  - 5NF (based on join dependencies)

Relational Decompositions (1)

- Universal Relation Schema:
  - A relation schema R = {A1, A2, ..., An} that includes all the attributes of the database.
- Universal relation assumption:
  - Every attribute name is unique.

Relational Decompositions (2)

- Relational Decomposition:
  - The process of decomposing the universal relation schema R into a set of relation schemas D = {R1,R2, ..., Rm} that will become the relational database schema by using the functional dependencies.
- Attribute preservation condition:
  - Each attribute in R will appear in at least one relation schema Ri in the decomposition so that no attributes are "lost".

Relational Decompositions (3)

- Another goal of decomposition is to have each individual relation Ri in the decomposition D be in BCNF or 3NF.
- Additional properties of decomposition are needed to prevent from generating spurious tuples

Dependency Preservation (1)

- **Definition**: Given a set of dependencies F on R, the **projection** of F on R_i, denoted by $\pi_{Ri}(F)$ where R_i is a subset of R, is the set of dependencies X → Y in F^+ such that the attributes in X ∪ Y are all contained in R_i.
- Hence, the projection of F on each relation schema R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left- and right-hand-side attributes are in R_i.

Dependency Preservation (2)

- Dependency Preservation Property:
  - A decomposition D = {R1, R2, ..., Rm} of R is **dependency-preserving** with respect to F if the union of the projections of F on each Ri in D is equivalent to F; that is
    $((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+$
- Claim 1:
  - It is always possible to find a dependency-preserving decomposition D with respect to F such that each relation Ri in D is in 3NF.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to F)
  - *result* = α
    while (changes to *result*) do
    for each R_i in the decomposition
    $t = (result \cap R_i)^+ \cap R_i$
    $result = result \cup t$
  - If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.

Testing for Dependency Preservation

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

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Dependency preserving
  - Lossless-join decomposition (next slide)

Lossless (Non-additive) Join

- **Definition**: Lossless join property: a decomposition D = {R1, R2, ..., Rm} of R has the **lossless** (**nonadditive**) **join property** with respect to the set of dependencies F on R if, for *every* relation state r of R that satisfies F, the following holds, where * is the natural join of all the relations in D:

$$* (\pi_{R1}(r), \ldots, \pi_{Rm}(r)) = r$$

- Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for "loss of information" a better term is "**addition of spurious information**"

Testing Lossless Join (1)

- **Algorithm 15.3: Testing for Lossless Join Property**
  - **Input**: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set S(i,j):=b_{ij} for all matrix entries. (* each b_{ij} is a distinct symbol associated with indices (i,j) *).
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
   {if (relation Ri includes attribute A_j) then
   set S(i,j):= a_j;};}; (* each a_j is a distinct symbol associated with index (j) *)

Testing Lossless Join (2)

4. Repeat the following loop until no changes to S
   {for each functional dependency X →Y in F
   {for all rows in S *which have the same symbols* in the columns corresponding to attributes in X
   { make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:
   If any of the rows has an "a" symbol for the column, set the other rows to that *same* "a" symbol in the column.
   If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to that same "b" symbol in the column ;
   };
   };
   };
5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise it does not.

Testing Lossless Join (3)

Lossless (nonadditive) join test for *n*-ary decompositions.
(a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test.
(b) A decomposition of EMP_PROJ that has the lossless join property.

(a) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2\}$
R_1 = EMP_LOCS = {Ename, Plocation}
R_2 = EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(No changes to matrix after applying functional dependencies)

(b)

EMP

Ssn	Ename

PROJECT

Pnumber	Pname	Plocation

WORKS_ON

Ssn	Pnumber	Hours

Testing Lossless Join (4)

Lossless (nonadditive) join test for n-ary decompositions.
(c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

(c) $R = \{Ssn, Ename, Pnumber, Pname, Plocation, Hours\}$ $D = \{R_1, R_2, R_3\}$

$R_1 = EMP = \{Ssn, Ename\}$

$R_2 = PROJ = \{Pnumber, Pname, Plocation\}$

$R_3 = WORKS\_ON = \{Ssn, Pnumber, Hours\}$

$F = \{Ssn \rightarrow Ename; Pnumber \rightarrow \{Pname, Plocation\}; \{Ssn, Pnumber\} \rightarrow Hours\}$

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Original matrix S at start of algorithm)

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(Matrix S after applying the first two functional dependencies; last row is all "a" symbols so we stop)

Testing Lossless Join (4)

- **Testing Binary Decompositions for Lossless Join Property**
  - **Binary Decomposition:** Decomposition of a relation R into two relations.
  - **PROPERTY LJ1 (lossless join test for binary decompositions):** A decomposition $D = \{R_1, R_2\}$ of R has the lossless join property with respect to a set of functional dependencies F on R *if and only if* either
    - The FD $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or
    - The FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+.

Example

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

Successive Lossless Join

- **Successive Lossless Join Decomposition:**
  - **Claim 2 (Preservation of non-additivity in successive decompositions):**
    - If a decomposition D = {R1, R2, ..., Rm} of R has the lossless (non-additive) join property with respect to a set of functional dependencies F on R,
    - and if a decomposition Di = {Q1, Q2, ..., Qk} of Ri has the lossless (non-additive) join property with respect to the projection of F on Ri,
      - then the decomposition D2 = {R1, R2, ..., Ri-1, Q1, Q2, ..., Qk, Ri+1, ..., Rm} of R has the non-additive join property with respect to F.

Algorithms for RDB Design - Finding Minimal Cover

- Algorithm 15.2: Find a minimal cover F for a set of FDs E

1. F := E
2. Replace each FD X→{A1, A2, ..., An} in F by X→A1, X→A2, ..., X→An
3. For each X→A in F

    For each attribute B∈X

        if {F – {X→A}}∪{(X-{B})→A} ≡ F

            replace X→A with (X-{B})→A in F

1. For each X→A in F

    if F – {X→A} is equivalent to F

        remove X→A from F

Computing a Minimal Cover

- $R = (A, B, C)$
  $F = \{ A \rightarrow BC,\ B \rightarrow C,\ A \rightarrow B,\ AB \rightarrow C \}$
- Replace $A \rightarrow BC$ by $A \rightarrow B$ and $A \rightarrow C$
  - Set is now $\{A \rightarrow B, A \rightarrow C, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - Set is now $\{A \rightarrow B, A \rightarrow C, B \rightarrow C\}$
- $A \rightarrow C$ is extraneous since it can be inferred from $A \rightarrow B$ and $B \rightarrow C$
- The canonical cover is: $\{ A \rightarrow B, B \rightarrow C \}$

Algorithms for RDB Design - Key Determination

- **Algorithm 15.2(a) Finding a Key K for R Given a set F of Functional Dependencies**
  - **Input: A universal relation R and a set of FDs F**

1. Set K := R;
2. For each attribute A in K {

    Compute (K - A)+ with respect to F;

    If (K - A)+ contains all the attributes in R,

        then set K := K - {A};

   }

Relational Synthesis Algorithm into 3NF

- **Algorithm 15.4: Relational Synthesis into 3NF with Dependency Preservation**
  - **Input: A universal relation R and a set of FDs F**

1. Find a minimal cover G for F
2. For each LHS X of a FD in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_k$ are the only dependencies in G with X as LHS (X is the key of this relation) ;
3. Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

- **Claim 3: Every relation schema created by Algorithm 15.4 is in 3NF.**

Relational Decomposition into BCNF

- **Algorithm 15.5: Relational Decomposition into BCNF with Lossless (non-additive) join property**
  - **Input: A universal relation R and a set of FDs F**

1. Set D := {R};
2. While (there is a schema Q in D that is not in BCNF) do {
   choose a schema Q in D that is not in BCNF;
   find a FD X → Y in Q that violates BCNF;
   replace Q in D by two schemas (Q - Y) and (X ∪ Y);
   };

Assumption: No null values are allowed for the join attributes.

Example of BCNF Decomposition

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

Example of BCNF Decomposition

- Original relation R and FD F

  R = (*branch_name, branch_city, assets, customer_name, loan_number, amount*)

  F = {*branch_name* → *assets branch_city*
  loan_number → *amount branch_name* }

  Key = {*loan_number, customer_name*}
- Decomposition
  - R_1 = (*branch_name, branch_city, assets*)
  - R_2 = (*branch_name, customer_name, loan_number, amount*)
  - R_3 = (*branch_name, loan_number, amount*)
  - R_4 = (*customer_name, loan_number*)
- Final decomposition: R_1, R_3, R_4

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
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

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

Relational Synthesis Algorithm into 3NF

- **Algorithm 15.4 Relational Synthesis into 3NF with Dependency Preservation and Lossless (Non-Additive) Join Property**
  - **Input: A universal relation R and a set of FDs F**

1. Find a minimal cover G for F (Algorithm 15.2)
2. For each LHS X of a FD in G, create a schema in D with attributes {X ∪ {A1} ∪ {A2} ... ∪ {Ak}}, where X → A1, X → A2, ..., X –>Ak are the only dependencies in G with X as LHS (X is the key of this relation).
3. If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R. *(Use Algorithm 15.2(a) to find the key of R)*
4. Eliminate redundant relations (subsumed by others)

Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id*, *branch_name, type*)

- The functional dependencies for this relation schema are:

  $customer\_id, employee\_id \rightarrow branch\_name, type$

  $employee\_id \rightarrow branch\_name$

- The **for** loop generates:

  (*customer_id, employee_id, branch_name, type*)

  It then generates

  (*employee_id*, *branch_name*)

  but does not include it in the decomposition because it is a subset of the first schema.

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF st:
  - 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 st:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Discussion of Normalization Algorithms

- Problems:
  - The database designer must first specify *all* the relevant functional dependencies among the database attributes.
  - These algorithms are *not deterministic* in general.
  - It is not always possible to find a decomposition into relation schemas that preserves dependencies and allows each relation schema in the decomposition to be in BCNF (instead of 3NF as in Algorithm 15.4).

Summary of Algorithms

Table 15.1 Summary of the Algorithms Discussed in This Chapter

Algorithm	Input	Output	Properties/Purpose	Remarks
15.1	An attribute or a set of attributes *X*, and a set of FDs *F*	A set of attributes in the closure of *X* with respect to *F*	Determine all the attributes that can be functionally determined from *X*	The closure of a key is the entire relation
15.2	A set of functional dependencies *F*	The minimal cover of functional dependencies	To determine the minimal cover of a set of dependencies *F*	Multiple minimal covers may exist—depends on the order of selecting functional dependencies
15.2a	Relation schema *R* with a set of functional dependencies *F*	Key *K* of *R*	To find a key *K* (that is a subset of *R*)	The entire relation *R* is always a default superkey
15.3	A decomposition *D* of *R* and a set *F* of functional dependencies	Boolean result: yes or no for nonadditive join property	Testing for nonadditive join decomposition	See a simpler test NJB in Section 14.5 for binary decompositions
15.4	A relation *R* and a set of functional dependencies *F*	A set of relations in 3NF	Nonadditive join and dependency-preserving decomposition	May not achieve BCNF, but achieves *all* desirable properties and 3NF
15.5	A relation *R* and a set of functional dependencies *F*	A set of relations in BCNF	Nonadditive join decomposition	No guarantee of dependency preservation
15.6	A relation *R* and a set of functional and multivalued dependencies	A set of relations in 4NF	Nonadditive join decomposition	No guarantee of dependency preservation

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys. (Can specify FDs using assertions, but they are expensive to test)
- 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.

Multivalued Dependencies (MVDs)

- 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 s.t.:

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$ $t_3[R-\beta] = t_2[R-\beta]$
$t_4[\beta] = t_2[\beta]$ $t_4[R-\beta] = t_1[R-\beta]$

MVD (Cont.)

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

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

Another View of MVD

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

  $$Y, Z, W$$

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

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

  then

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

- Note that since the behavior of Z and W are identical it follows that $Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

Example

- In our (course, teacher, book) example:

  course $\rightarrow\rightarrow$ teacher   *course* $\rightarrow\rightarrow$ *book*

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

Example of MVD

	course	*teacher*	*book*
t3	database	Avi	DB Concepts
t1	database	Avi	Ullman
t2	database	Hank	DB Concepts
t4	database	Hank	Ullman
	database	Sudarshan	DB Concepts
	database	Sudarshan	Ullman
	operating system	Avi	OS Concepts
	operating system	Avi	Stallings
	operating system	Pete	OS Concepts
	operating system	Pete	Stallings

Use of Multivalued Dependencies

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

Theory of MVDs

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

  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 MVDs, 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.

Inference Rules for FD's and MVD's

- The following set of rules is **sound** and **complete**.
- For FD's:

  IR1 (**Reflexive rule**):

  $$\{X \supseteq Y\} \models X \to Y$$

  IR2 (**Augmentation rule**):

  $$\{X \to Y\} \models XZ \to YZ$$

  IR3 (**Transitive rule**):

  $$\{X \to Y, Y \to Z\} \models X \to Z$$

Inference Rules for MVD's

IR4 (**Complementation rule**):
$\{X \rightarrow\rightarrow Y\} \models \{X \rightarrow\rightarrow (R - (X \cup Y))\}$

IR5 (**Multivalued augmentation rule**):
if $X \rightarrow\rightarrow Y$ and $W \supseteq Z$ then $WX \rightarrow\rightarrow YZ$

IR6 (**Multivalued transitive rule**):
$\{X \rightarrow\rightarrow Y, Y \rightarrow\rightarrow Z\} \models X \rightarrow\rightarrow (Z - Y)$

IR7 (**Replication rule**):
$\{X \rightarrow Y\} \models X \rightarrow\rightarrow Y$

IR8 (**Coalescence rule**):
if $X \rightarrow\rightarrow Y$ and $\exists W$ such that $W \cap Y = \emptyset$ and $W \rightarrow Z$ and $Y \supseteq Z$, then $X \rightarrow Z$

- Note that an FD is a special case of MVD.

Fourth Normal Form (4NF)

- 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
  (Proof: Exercise)

Example of 4NF

(a) The EMP relation with two MVDs: ENAME —>> PNAME and ENAME —>> DNAME.
(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.

(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John
Brown	W	Jim
Brown	X	Jim
Brown	Y	Jim
Brown	Z	Jim
Brown	W	Joan
Brown	X	Joan
Brown	Y	Joan
Brown	Z	Joan
Brown	W	Bob
Brown	X	Bob
Brown	Y	Bob
Brown	Z	Bob

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y
Brown	W
Brown	X
Brown	Y
Brown	Z

EMP_DEPENDENTS

Ename	Dname
Smith	Anna
Smith	John
Brown	Jim
Brown	Joan
Brown	Bob

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 MVD that holds on R_i s.t.
     $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \phi$;
    result := (*result* - R_i) $\cup$ (R_i - β) $\cup$ (α, β);
   end
  else *done*:= true;
(Note: each R_i is in 4NF, and decomposition is lossless-join)

Lossless (Non-additive) Join Decomposition into 4NF Relations

- The relation schemas R_1 and R_2 form a lossless (non-additive) join decomposition of R with respect to a set F of functional *and* multivalued dependencies if and only if
  - $(R_1 \cap R_2) \rightarrow\rightarrow (R_1 - R_2)$
- or by symmetry, if and only if
  - $(R_1 \cap R_2) \rightarrow\rightarrow (R_2 - R_1)$
- **Proof**: Exercise.

Checking for Lossless Join Decomposition

- **Theorem**: R_1 and R_2 is a lossless join decomposition of R if and only if
  - $R_1 \cap R_2 \rightarrow\rightarrow R_1$, or
  - $R_1 \cap R_2 \rightarrow\rightarrow R_2$
- **Proof**: Exercise.

Decomposition into 4NF relations with non-additive join property

- **Algorithm 15.7: Input:** A universal relation R and a set of functional and multivalued dependencies F.

1. Set D := { R };
2. While there is a relation schema *Q* in *D* that is not in 4NF do {

   choose a relation schema *Q* in *D* that is not in 4NF;

   find a nontrivial MVD $X \rightarrow\rightarrow Y$ in Q that violates 4NF;

   replace Q in D by two relation schemas (Q - Y) and $(X \cup Y)$;

   };

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)
  c) $R_3 = (C, G, H)$ (R_3 is in 4NF)
  d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF)
- Since $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow HI$, $A \rightarrow\rightarrow HI$, $A \rightarrow\rightarrow I$
  e) $R_5 = (A, I)$ (R_5 is in 4NF)
  f) $R_6 = (A, C, G)$ (R_6 is in 4NF)

Another Example

- R = (course, teacher, book)
  course $\rightarrow\rightarrow$ teacher *course* $\rightarrow\rightarrow$ *book*
  R is not in 4NF
- R can be decomposed into
  R1 = (course, teacher)
  R2 = (course, book)
  Both R1 and R2 are now in 4NF.

Join Dependencies and Fifth Normal Form (1)

- A **join dependency** (**JD**), denoted by JD(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R.
  - The constraint states that every legal state r of R should have a non-additive join decomposition into R_1, R_2, ..., R_n; that is, for every such r we have

$$* (\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$$

***Note**: an MVD is a special case of a JD where n = 2.*

- A join dependency JD(R_1, R_2, ..., R_n), specified on relation schema R, is a **trivial JD** if one of the relation schemas R_i in JD(R_1, R_2, ..., R_n) is equal to R.

Join Dependencies

- A **join dependency** (**JD**), denoted by JD(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R.
  - The constraint states that every legal state r of R should have a non-additive join decomposition into R_1, R_2, ..., R_n; that is, for every such r we have
  - $* (\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$

***Note**: an MVD is a special case of a JD where n = 2.*

- A join dependency JD(R_1, R_2, ..., R_n), specified on relation schema R, is a **trivial JD** if one of the relation schemas R_i in JD(R_1, R_2, ..., R_n) is equal to R.

JD and Fifth Normal Form

- A relation schema R is in **fifth normal form** (**5NF**) (or **Project-Join Normal Form** (**PJNF**)) with respect to a set F of functional, multivalued, and join dependencies if,
  - for every nontrivial join dependency JD(R_1, R_2, ..., R_n) in F^+ (that is, implied by F), every R_i is a superkey of R.

Example of 5NF Normalization

(c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R_1, R_2, R_3).
(d) Decomposing the relation SUPPLY into the 5NF relations R_1, R_2, R_3.

(c) SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Inclusion Dependencies (1)

- An **inclusion dependency** $R.X < S.Y$ between two sets of attributes(X of relation schema R, and Y of relation schema S)specifies the constraint that, at any specific time when r is a relation state of R and s a relation state of S, we must have

$$\pi_X(r(R)) \supseteq \pi_Y(s(S))$$

- **Note**:
  - The $\supseteq$ (subset) relationship does not necessarily have to be a proper subset.
  - The sets of attributes on which the inclusion dependency is specified—X of R and Y of S—must have the same number of attributes.
  - In addition, the domains for each pair of corresponding attributes should be compatible.

Inclusion Dependencies (2)

- **Objective of Inclusion Dependencies:**
  - To formalize two types of interrelational constraints which cannot be expressed using F.D.s or MVDs:
    - Referential integrity constraints
    - Class/subclass relationships
- **Inclusion dependency inference rules**
  - IDIR1 (reflexivity): $R.X < R.X$.
  - IDIR2 (attribute correspondence): If $R.X < S.Y$
    - where X = {A1, A2 ,…., An} and Y = {B1, B2, …, Bn} and Ai Corresponds-to Bi, then $R.A_i < S.B_i$ for $1 \le i \le n$.
  - IDIR3 (transitivity): If $R.X < S.Y$ and $S.Y < T.Z$, then $R.X < T.Z$

Other Dependencies and Normal Forms (1)

Template Dependencies:

- Template dependencies provide a technique for representing constraints in relations that typically have no easy and formal definitions.
- The idea is to specify a template—or example—that defines each constraint or dependency.
- There are two types of templates:
  - tuple-generating templates
  - constraint-generating templates.
- A template consists of a number of hypothesis tuples that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the template conclusion.

Other Dependencies and Normal Forms (2)

Figure 16.5
Templates for some common type of dependencies.
(a) Template for functional dependency $X \to Y$.
(b) Template for the multivalued dependency $X \twoheadrightarrow Y$.
(c) Template for the inclusion dependency $R.X < S.Y$.

(a) $R = \{A, B, C, D\}$; $X = \{A, B\}$, $Y = \{C, D\}$

Hypothesis:

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2

Conclusion: $c_1 = c_2$ and $d_1 = d_2$

(b) $R = \{A, B, C, D\}$; $X = \{A, B\}$, $Y = \{C\}$

Hypothesis:

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2

Conclusion:

A	B	C	D
a_1	b_1	c_2	d_1
a_1	b_1	c_1	d_2

(c) $R = \{A, B, C, D\}$, $S = \{E, F, G\}$; $X = \{C, D\}$, $Y = \{E, F\}$

Hypothesis:

A	B	C	D
a_1	b_1	c_1	d_1

Conclusion:

E	F	G
c_1	d_1	g

Other Dependencies and Normal Forms (3)

Figure 16.6
Templates for the constraint that an employee's salary must be less than the supervisor's salary.

EMPLOYEE = {Name, Ssn, . . . , Salary, Supervisor_ssn}

	a	b	c	d
Hypothesis	e	d	f	g
Conclusion			c < f	

Domain-Key Normal Form(DKNF)

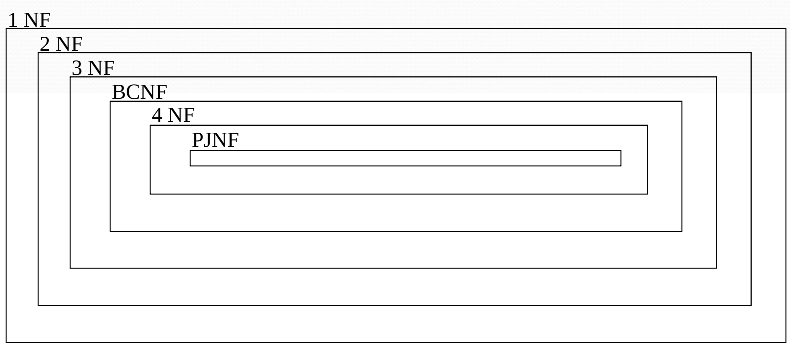
- **Definition:**
  - A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation.
- The idea is to specify (theoretically, at least) the "ultimate normal form" that takes into account all possible types of dependencies and constraints. .
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.
- The practical utility of DKNF is limited

Note on Higher Normal Forms

- 5NF and DKNF are rarely used
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.

Levels of Normalization

- The relationship between various normal forms:



Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting ER-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 FDs from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes *department_number* and *department_address*, and a functional dependency *department_number* $\rightarrow$ *department_address*
  - 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
- Eg, displaying *customer_name* along with *account_number* and *balance* requires join of *account* with *depositor*
- Alternative 1: Use denormalized relation containing all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work and possibility of error in extra code
- Alternative 2: use a materialized view defined as
  account ⋈ depositor
  - 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_2003, earnings_2004, earnings_2005, etc., all on the schema (*company_id, earnings*).
- Above are in BCNF, but make querying across years difficult and needs new table each year

Other Design Issues (cont.)

- *company_year*(*company_id*, *earnings_2000*, *earnings_2001*, *earnings_2002*)
  - 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

Recap

- Designing a Set of Relations
- Properties of Relational Decompositions
- Algorithms for Relational Database Schema
- Multivalued Dependencies and Fourth Normal Form
- Join Dependencies and Fifth Normal Form
- Other Dependencies and Normal Forms

Assignment 5

- Textbook exercises: 14.19, 14.24, 14.25, 15.26, 15.27
- No need to turn in!