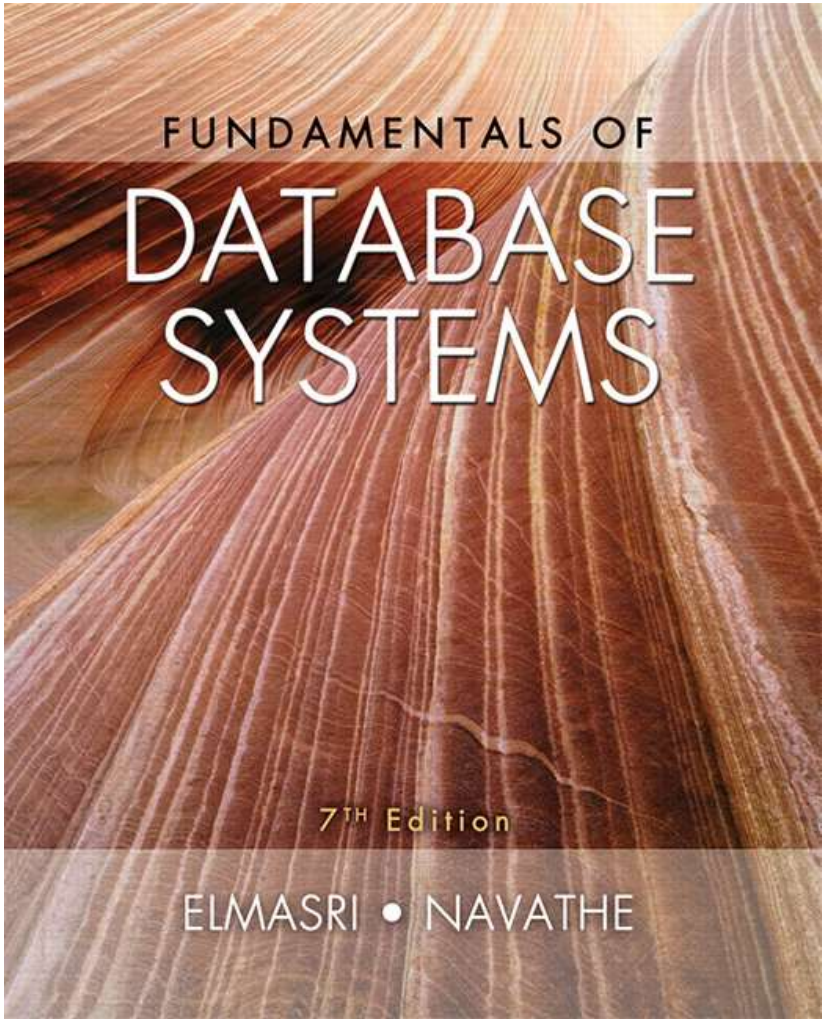
FUNDAMENTALS OF
DATABASE
SYSTEMS
7TH Edition
ELMASRI • NAVATHE

CHAPTER 15

Relational Database Design Algorithms and Further Dependencies

Chapter Outline

- 1. Further topics in Functional Dependencies
  - 1.1 Inference Rules for FDs
  - 1.2 Equivalence of Sets of FDs
  - 1.3 Minimal Sets of FDs
- 2. Properties of Relational Decompositions
- 3. Algorithms for Relational Database Schema Design
- 4. Nulls, Dangling Tuples, Alternative Relational Designs

Chapter Outline

- 5. Multivalued Dependencies and Fourth Normal Form – further discussion
- 6. Other Dependencies and Normal Forms
  - 6.1 Join Dependencies
  - 6.2 Inclusion Dependencies
  - 6.3 Dependencies based on Arithmetic Functions and Procedures
  - 6.2 Domain-Key Normal Form

1. Functional Dependencies : Inference Rules, Equivalence and Minimal Cover

- We discussed functional dependencies in the last chapter.
- To recollect:

A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y.

- Our goal here is to determine the properties of functional dependencies and to find out the ways of manipulating them.

Defining Functional Dependencies

- $X \rightarrow Y$ holds if whenever two tuples have the same value for X, they *must have* the same value for Y
  - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], *then* t1[Y]=t2[Y]
- $X \rightarrow Y$ in R specifies a *constraint* on all relation instances r(R)
- Written as $X \rightarrow Y$; can be displayed graphically on a relation schema as in Figures in Chapter 14. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

1.1 Inference Rules for FDs (1)

- **Definition:** An FD $X \rightarrow Y$ is **inferred from** or **implied by** a set of dependencies F specified on R if $X \rightarrow Y$ holds in *every* legal relation state r of R; that is, whenever r satisfies all the dependencies in F, $X \rightarrow Y$ also holds in r.

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold

Inference Rules for FDs (2)

- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X → Y
  - IR2. (**Augmentation**) If X → Y, then XZ → YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X → Y and Y → Z, then X → Z

- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules
  - These are rules hold and all other rules that hold can be deduced from these

Inference Rules for FDs (3)

- Some additional inference rules that are useful:
  - **Decomposition:** If X → YZ, then X → Y and X → Z
  - **Union:** If X → Y and X → Z, then X → YZ
  - **Psuedotransitivity:** If X → Y and WY → Z, then WX → Z

- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Closure

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F
- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X
- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

Algorithm to determine Closure

- **Algorithm 15.1.** Determining X^+, the Closure of X under F
- **Input:** A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

$X^+ := X$;

repeat

  old$X^+ := X^+$;

for each functional dependency $Y \rightarrow Z$ in F do

    if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;

until ($X^+ =$ oldX^+);

Example of Closure (1)

- For example, consider the following relation schema about classes held at a university in a given academic year.

CLASS (Classid, Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity).

- Let *F*, the set of functional dependencies for the above relation include the following f.d.s:

FD1: Sectionid $\rightarrow$ Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity;

FD2: Course# $\rightarrow$ Credit_hrs;

FD3: {Course#, Instr_name} $\rightarrow$ Text, Classroom;

FD4: Text $\rightarrow$ Publisher

FD5: Classroom $\rightarrow$ Capacity

These f.d.s above represent the meaning of the individual attributes and the relationship among them and defines certain rules about the classes.

Example of Closure (2)

- The closures of attributes or sets of attributes for some example sets:

  { Classid }$^+$ = { Classid , Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity } = CLASS

  { Course#}$^+$ = { Course#, Credit_hrs}

  { Course#, Instr_name }$^+$ = { Course#, Credit_hrs, Text, Publisher, Classroom, Capacity }

  Note that each closure above has an interpretation that is revealing about the attribute(s) on the left-hand-side. The closure of { Classid }$^+$ is the entire relation CLASS indicating that all attributes of the relation can be determined from Classid and hence it is a key.

1.2 Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if $F^+ = G^+$
- Definition (**Covers**):
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if G^+ *subset-of* F^+)
- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

1.3 Finding Minimal Cover of F.D.s (1)

- Just as we applied inference rules to expand on a set *F* of FDs to arrive at *F*+, its closure, it is possible to think in the opposite direction to see if we could shrink or reduce the set *F* to its *minimal form* so that the minimal set is still equivalent to the original set *F.*
- **Definition:** An attribute in a functional dependency is considered **extraneous attribute** if we can remove it without changing the closure of the set of dependencies. Formally, given F, the set of functional dependencies and a functional dependency $X \rightarrow A$ in *F* , attribute *Y* is extraneous in *X* if *Y is a subset of X,* and *F* logically implies $(F - (X \rightarrow A) \cup \{ (X - Y) \rightarrow A \})$

Minimal Sets of FDs (2)

- A set of FDs is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper-subset-of X and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs (3)

- **Algorithm 15.2. Finding a Minimal Cover F for a Set of Functional Dependencies E**
  - **Input: A set of functional dependencies E.**

1. Se tF:=E.
2. Replace each functional dependency $X \rightarrow \{A1, A2, ..., An\}$ in F by the n functional dependencies $X \rightarrow A1$, $X \rightarrow A2$, ..., $X \rightarrow An$.
3. For each functional dependency $X \rightarrow A$ in F
   for each attribute B that is an element of X
   if $\{ \{F - \{X \rightarrow A\} \} \cup \{ (X - \{B\}) \rightarrow A\} \}$ is equivalent to F
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.

   (* The above constitutes a removal of the extraneous attribute B from X *)
4. For each remaining functional dependency $X \rightarrow A$ in F if $\{F - \{X \rightarrow A\} \}$ is equivalent to F, then remove $X \rightarrow A$ from F.

   (* The above constitutes a removal of the redundant dependency $X \rightarrow A$ **from F *)**

Computing the Minimal Sets of FDs (4)

We illustrate algorithm 15.2 with the following:
Let the given set of FDs be E : {$B \rightarrow A$, $D \rightarrow A$, $AB \rightarrow D$}.We have to find the minimum cover of E.

- All above dependencies are in canonical form; so we have completed step 1 of Algorithm 10.2 and can proceed to step 2. In step 2 we need to determine if $AB \rightarrow D$ has any redundant attribute on the left-hand side; that is, can it be replaced by $B \rightarrow D$ or $A \rightarrow D$?
- Since B → A, by augmenting with B on both sides (IR2), we have $BB \rightarrow AB$, or $B \rightarrow AB$ (i). However, $AB \rightarrow D$ as given (ii).
- Hence by the transitive rule (IR3), we get from (i) and (ii), $B \rightarrow D$. Hence $AB \rightarrow D$ may be replaced by $B \rightarrow D$.
- We now have a set equivalent to original E , say E' : {$B \rightarrow A$, $D \rightarrow A$, $B \rightarrow D$}. No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
- In step 3 we look for a redundant FD in E′. By using the transitive rule on $B \rightarrow D$ and $D \rightarrow A$, we derive $B \rightarrow A$. Hence $B \rightarrow A$ is redundant in E' and can be eliminated.
- Hence the minimum cover of E is {$B \rightarrow D$, $D \rightarrow A$}.

Minimal Sets of FDs (5)

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs. The process of Algorithm 15.2 is used until no further reduction is possible.
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set
  - E.g., see algorithm 15.4

DESIGNING A SET OF RELATIONS (1)

- **The Approach of 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 the *set of relations* in a relational database are satisfactory (see Algorithm 15.3).

DESIGNING A SET OF RELATIONS (2)

- **Goals:**
  - Lossless join property (a must)
    - Algorithm 15.3 tests for general losslessness.
  - Dependency preservation property
    - Observe as much as possible
    - 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)

Algorithm to determine the key of a relation

- **Algorithm 15.2a Finding a Key K for R, given a set F of Functional Dependencies**
  - **Input: A universal relation R and a set of functional dependencies F on the attributes of R.**

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};
    }

2. Properties of Relational Decompositions (1)

- **Relation Decomposition and Insufficiency of Normal Forms:**
  - 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.

Properties of Relational Decompositions (2)

2.1 Relation Decomposition and Insufficiency of Normal Forms (cont.):

- 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 R_i in the decomposition so that no attributes are "lost".

Properties of Relational Decompositions (3)

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

Properties of Relational Decompositions (4)

2.2 Dependency Preservation Property of a Decomposition:

- Definition: Given a set of dependencies F on R, the **projection** of F on R_i, denoted by $p_{Ri}(F)$ where R_i is a subset of R, is the set of dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \cup Y$ 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.

Properties of Relational Decompositions (5)

- **Dependency Preservation Property of a Decomposition (cont.):**
  - 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

      $$((\pi_{R1}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^+$$
    - (See examples in Fig 14.13a and Fig 14.12)
- Claim 1:
  - It is always possible to find a dependency-preserving decomposition D with respect to F such that each relation R_i in D is in 3nf.

Properties of Relational Decompositions (6)

2.3 Non-additive (Lossless) Join Property of a Decomposition:

- 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**"

Properties of Relational Decompositions (7)

Lossless (Non-additive) Join Property of a Decomposition :

- **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 Ri in D, and one column j for each attribute Aj in R.

2. Set S(i,j):=bij for all matrix entries. (* each bij is a distinct symbol associated with indices (i,j) *).

3. For each row i representing relation schema Ri

{for each column j representing attribute Aj

{if (relation Ri includes attribute Aj) then set S(i,j):= aj;};};

  - (* each aj is a distinct symbol associated with index (j) *)

- CONTINUED on NEXT SLIDE

Properties of Relational Decompositions (8)

- **Lossless (Non-additive) Join Property of a Decomposition (cont.):**

Algorithm 15.3: Testing for Lossless Join Property (continued)

4. Repeat the following loop until a complete loop execution results in 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.

Properties of Relational Decompositions (9)

Figure 15.1 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 $\rightarrow$ Ename; Pnumber $\rightarrow$ $\{$Pname, Plocation$\}$; $\{$Ssn, Pnumber$\}$ $\rightarrow$ 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

Properties of Relational Decompositions (10)

Nonadditive join test for n-ary decompositions.
(Figure 15.1)
(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 → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → 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	$\cancel{b_{32}}$ a_2	a_3	$\cancel{b_{34}}$ a_4	$\cancel{b_{35}}$ a_5	a_6

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

Test for checking non-additivity of Binary Relational Decompositions (11)

2.4 Testing Binary Decompositions for Non-additive Join (Lossless Join) Property

- **Binary Decomposition:** Decomposition of a relation R into two relations.
- **PROPERTY NJB (non-additive join test for binary decompositions):** A decomposition D = {R1, R2} of R has the lossless join property with respect to a set of functional dependencies F on R *if and only if* either
  - The f.d. ((R1 ∩ R2) → (R1- R2)) is in F^+, or
  - The f.d. ((R1 ∩ R2) → (R2 - R1)) is in F^+.

Properties of Relational Decompositions (12)

2.5 Successive Non-additive 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.

3. Algorithms for Relational Database Schema Design (1)

- **Design of 3NF Schemas:**

Algorithm 15.4 Relational Synthesis into 3NF with Dependency Preservation and Non-Additive (Lossless) Join Property

- **Input: A universal relation R and a set of functional dependencies F on the attributes of R.**

1. Find a minimal cover G for F (use Algorithm 15.0).

2. For each left-hand-side X of a functional dependency that appears in G,

create a relation 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 left-hand-side (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 15.4a to find the key of R)*

Algorithms for Relational Database Schema Design (2)

- **Design of BCNF Schemas**

Algorithm 15.5: Relational Decomposition into BCNF with Lossless (non-additive) join property

- **Input: A universal relation R and a set of functional dependencies F on the attributes of R.**

1. Set D := {R};

2. While there is a relation schema Q in D that is not in BCNF

do {

choose a relation schema Q in D that is not in BCNF;

find a functional dependency X → Y in Q that violates BCNF;

replace Q in D by two relation schemas (Q - Y) and (X ∪ Y);

};

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

4. Problems with Null Values and Dangling Tuples (1)

4.1 Problems with NULL values

- when some tuples have NULL values for attributes that will be used to join individual relations in the decomposition that may lead to incomplete results.
- E.g., see Figure 15.2(a), where two relations EMPLOYEE and DEPARTMENT are shown. The last two employee tuples—'Berger' and 'Benitez'—represent newly hired employees who have not yet been assigned to a department (assume that this does not violate any integrity constraints).
- If we want to retrieve a list of (Ename, Dname) values for all the employees. If we apply the NATURAL JOIN operation on EMPLOYEE and DEPARTMENT (Figure 15.2(b)), the two aforementioned tuples will *not* appear in the result.
- In such cases, LEFT OUTER JOIN may be used. The result is shown in Figure 15.2 (c).

.

Problems with Null Values and Dangling Tuples (2)

(a)

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnum
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	NULL

DEPARTMENT

Dname	Dnum	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

Figure 15.2
Issues with NULL-value joins. (a) Some EMPLOYEE tuples have NULL for the join attribute Dnum.

Problems with Null Values and Dangling Tuples (3)

(b)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

(c)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL	NULL	NULL
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX	NULL	NULL	NULL

Figure 15.2
Issues with NULL-value joins.
(b) Result of applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT relations.
(c) Result of applying LEFT OUTER JOIN to EMPLOYEE and DEPARTMENT

Problems with Null Values and Dangling Tuples (4)

Problems with Dangling Tuples

- Consider the decomposition of EMPLOYEE into EMPLOYEE_1 and EMPLOYEE_2 as shown in Figure 15.3 (a) and !5.3 (b).
- Their NATURAL JOIN yields the original relation EMPLOYEE in Figure 15.2(a).
- We may use the alternative representation, shown in Figure 15.3(c), where we *do not include a tuple* in EMPLOYEE_3 if the employee has not been assigned a department (instead of including a tuple with NULL for Dnum as in EMPLOYEE_2).
- If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result; these are called **dangling tuples** in EMPLOYEE.

Problems with Null Values and Dangling Tuples (5)

(a) EMPLOYEE_1

Ename	Ssn	Bdate	Address
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX

(b) EMPLOYEE_2

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) EMPLOYEE_3

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figure 15.3
The dangling tuple problem. (a) The relation EMPLOYEE_1 (includes all attributes of EMPLOYEE from Figure 15.2(a) except Dnum). (b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values). (c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).

About Normalization Algorithms

4.2 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 15.5).

Summary of Algorithms for Relational Database Schema Design (1)

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

Summary of Algorithms for Relational Database Schema Design (2)

Table 15.1 Summary of the Algorithms Discussed in This Chapter

Algorithm	Input	Output	Properties/Purpose	Remarks
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

5. Multivalued Dependencies and Fourth Normal Form – Further Discussion (1)

Definition:

- A **multivalued dependency** (**MVD**) $X \twoheadrightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote (R 2 ($X \cup Y$)):
  - $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
  - $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
  - $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.
- An MVD $X \twoheadrightarrow Y$ in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) $X \cup Y = R$.

Multivalued Dependencies and Fourth Normal Form (2)

- **Inference Rules for Functional and Multivalued Dependencies**:
  - IR1 (**reflexive rule for FDs**): If $X \supseteq Y$, then $X \rightarrow Y$.
  - IR2 (**augmentation rule for FDs**): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.
  - IR3 (**transitive rule for FDs**): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.
  - IR4 (**complementation rule for MVDs**): $\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow (R - (X \cup Y))\}$.
  - IR5 (**augmentation rule for MVDs**): If $X \twoheadrightarrow Y$ and $W \supseteq Z$ then $WX \twoheadrightarrow YZ$.
  - IR6 (**transitive rule for MVDs**): $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$.
  - IR7 (**replication rule for FD to MVD**): $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$.
  - IR8 (**coalescence rule for FDs and MVDs**): If $X \twoheadrightarrow Y$ and there exists W with the properties that
    - (a) $W \cap Y$ is empty, (b) $W \rightarrow Z$, and (c) $Y \supseteq Z$, then $X \rightarrow Z$.

Multivalued Dependencies and Fourth Normal Form (3)

Definition:

- A relation schema *R* is in **4NF** with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in F^+, *X* is a superkey for R.
  - Note: F^+ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state *r* of *R* that satisfies *F*. It is also called the **closure** of *F*.

Multivalued Dependencies and Fourth Normal Form (4)

Fig. 15.4 Decomposing a relation state of EMP that is not in 4NF.
(a) EMP relation with additional tuples.
(b) Two corresponding 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

Multivalued Dependencies and Fourth Normal Form (5)

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

- **PROPERTY NJB’**
  - 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
    - $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$
  - or by symmetry, if and only if
    - $(R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$).

Multivalued Dependencies and Fourth Normal Form (6)

Algorithm 15.7: Relational decomposition into 4NF relations with non-additive join property

- **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 $X \longrightarrow\!\!> Y$ in *Q* that violates 4NF;
   replace *Q* in *D* by two relation schemas $(Q - Y)$ and $(X \cup Y)$;
   };

6. Other Dependencies and Normal Forms

Join Dependency was defined in Chapter 14:

Definition:

- 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

$$* (\pi_{R1}(r), \pi_{R2}(r), ..., \pi_{Rn}(r)) = r$$

***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 and Fifth Normal Form

Definition of 5NF:

- 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.
- Discovering join dependencies in practical databases with hundreds of relations is next to impossible. Therefore, 5NF is rarely used in practice.

Inclusion Dependencies (1)

Definition:

- 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

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

- **Note**:
  - The $\subseteq$ (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 $X = \{A_1, A_2, \ldots, A_n\}$ and $Y = \{B_1, B_2, \ldots, B_n\}$ and A_i *Corresponds-to* B_i, then $R.A_i < S.B_i$
    - for $1 \leq i \leq n$.
  - **IDIR3 (transitivity)**: If $R.X < S.Y$ and $S.Y < T.Z$, then $R.X < T.Z$.

Functional Dependencies based on Arithmetic functions and procedures (1)

Arithmetic Functions:

- As long as a unique value of *Y* is associated with every *X*, we can still consider that the FD $X \rightarrow Y$ exists.

For example,consider the relation:

ORDER_LINE (Order#, Item#, Quantity, Unit_price, Extended_price, Discounted_price)

- each tuple represents an item from an order with a particular quantity, and the price per unit for that item. In this relation,

(Quantity, Unit_price) $\rightarrow$ Extended_price by the formula

Extended_price = Quantity * Unit_price .

- Hence, there is a unique value for Extended_price for every pair (Quantity, Unit_price), and thus it conforms to the definition of functional dependency.

Functional Dependencies based on Arithmetic functions and procedures (2)

Procedures:

- There may be a procedure that takes into account the quantity discounts, the type of item, and so on and computes a discounted price for the total quantity ordered for that item. Therefore, we can say
- (Item#, Quantity, Unit_price) $\rightarrow$ Discounted_price, or
- (Item#, Quantity, Extended_price) $\rightarrow$ Discounted_price.

Other Dependencies and Normal Forms (3)

6.4 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

Recap

- Functional Dependencies Revisited
- Designing a Set of Relations by Synthesis
- Properties of Relational Decompositions
- Algorithms for Relational Database Schema Design in 3Nf and BCNF
- Multivalued Dependencies and Fourth Normal Form
- Other Dependencies and Normal Forms