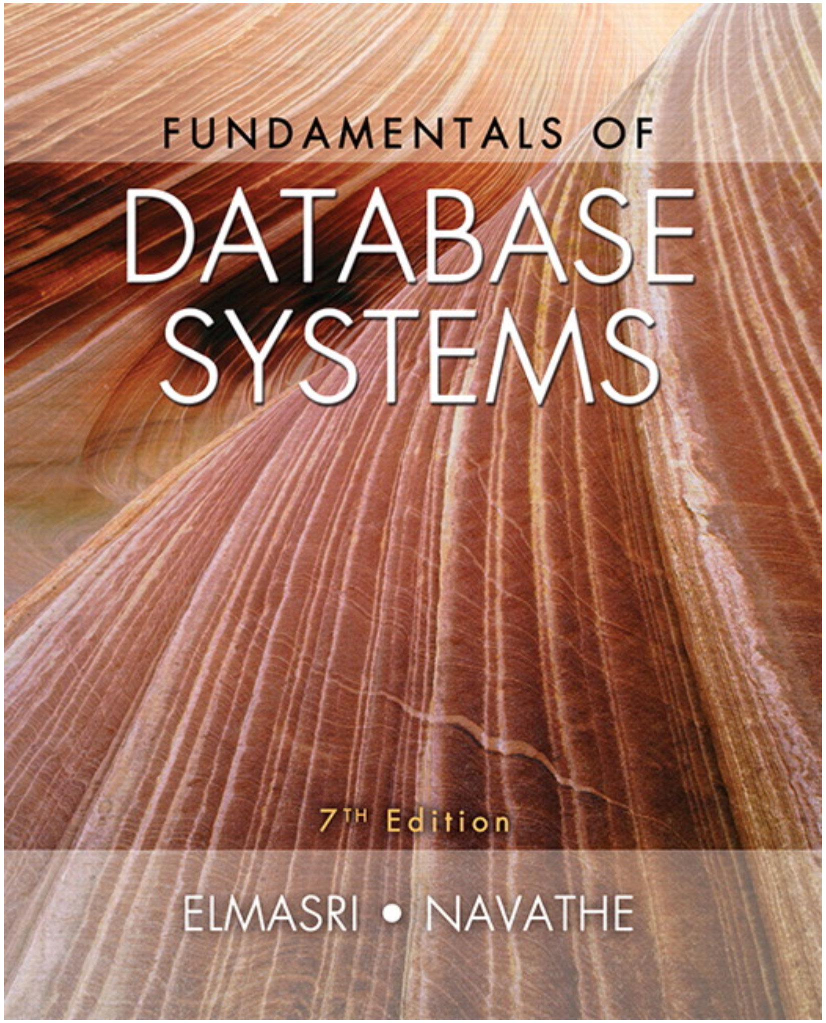
FUNDAMENTALS OF
DATABASE
SYSTEMS
7TH Edition
ELMASRI • NAVATHE

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

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)

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

FD1: Classid → Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity;

FD2: Course# → Credit_hrs;

FD3: {Course#, Instr_name} → Text, Classroom;

FD4: Text → Publisher

FD5: Classroom → Capacity

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*.

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

- **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.

Exercise

- Find the minimum cover for E:

$$E : \{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$$

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.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (**comment**).

1. Set $F := E$.
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$. (*This places the FDs in a canonical form for subsequent testing*)
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.

    (*This constitutes removal of an extraneous attribute B contained in the left-hand side X of a functional dependency $X \rightarrow A$ when possible*)
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. (*This constitutes removal of a redundant functional dependency $X \rightarrow A$ from F when possible*)

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 $\rightarrow$ 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

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.

Algorithm to determine the key of a relation

Algorithm 15.2(a). Finding a Key K for R Given a Set F of Functional Dependencies

Input: A 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)

- 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 $\pi_{R_i}(F)$ 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:

- 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
  $((\pi_{R1}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^{+} = F^{+}$

- 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. A better term is "**addition of spurious information**"

Algorithm 15.3. Testing for Nonadditive Join Property

Input: A universal relation R, a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R, and a set F of functional dependencies.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (**comment**).

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 R_i includes attribute A_j) then set $S(i, j) := a_j$;};}; (*Each a_j is a distinct symbol associated with index (j)*)
4. Repeat the following loop until a *complete loop execution* results in no changes to S
   {for each functional dependency $X \rightarrow Y$ in F
      {for all rows in S *that 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 appears 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 nonadditive 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.

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

$R_1 = \text{EMP\_LOCS} = \{\text{Ename, Plocation}\}$

$R_2 = \text{EMP\_PROJ1} = \{\text{Ssn, Pnumber, Hours, Pname, Plocation}\}$

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

	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) A decomposition of EMP_PROJ that has the lossless join property.
(c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

(b)

EMP

Ssn	Ename

PROJECT

Pnumber	Pname	Plocation

WORKS_ON

Ssn	Pnumber	Hours

(c)

$R = \{\text{Ssn, Ename, Pnumber, Pname, Plocation, Hours}\}$

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

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

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

$D = \{R_1, R_2, R_3\}$

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

	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)

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 15.2).
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \dots \cup \{A_k\}\}$, where $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_k$ 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. (Algorithm 15.2(a) may be used to find a key.)
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S in the schema; alternately, R is subsumed by S.[7]

Example 1 page 520

U(Emp_ssn, Pno, Esal, Ephone, Pname, Plocation)

FD1: Emp_ssn -> {Esal, Ephone, Dno}

FD2: Pno -> {Pname, Plocation}

FD3: Emp_ssn, Pno -> {Emp_ssn, Pno, Esal, Ephone, Pname, Plocation}

Key: Emp_ssn, Pno

Minimal cover:

FD1: Emp_ssn -> {Esal, Ephone, Dno}

FD2: Pno -> {Pname, Plocation}

Example 1 page 520

The second step of Algorithm 15.4 produces relations R_1 and R_2 as:

R_1 (<u>Emp_ssn</u>, Esal, Ephone, Dno)

R_2 (<u>Pno</u>, Pname, Plocation)

In step 3, we generate a relation corresponding to the key {Emp_ssn, Pno} of U. Hence, the resulting design contains:

R_1 (<u>Emp_ssn</u>, Esal, Ephone, Dno)

R_2 (<u>Pno</u>, Pname, Plocation)

R_3 (<u>Emp_ssn</u>, <u>Pno</u>)

Please workout Example 2 and 3 in Pages 520-521

Algorithms for Relational Database Schema Design (2)

Algorithm 15.5. Relational Decomposition into BCNF with Nonadditive 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 $X \rightarrow Y$ in Q that violates BCNF;

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

   } ;

Exercise

R(A, B, C, D)

FD1: C -> D

FD2: C -> A

FD3: B -> C

Find the key using Algorithm 15.2 (a)

Key: B

FD1 and FD2 violates BCNF

Exercise

FD1: C -> D
FD2: C -> A
FD3: B -> C

R1(A,B,C), R2(C,D)

R1 violates BCNF because of C-A

Decompose R1 into R11(B, C) and R12(C,A)

Test with NJB