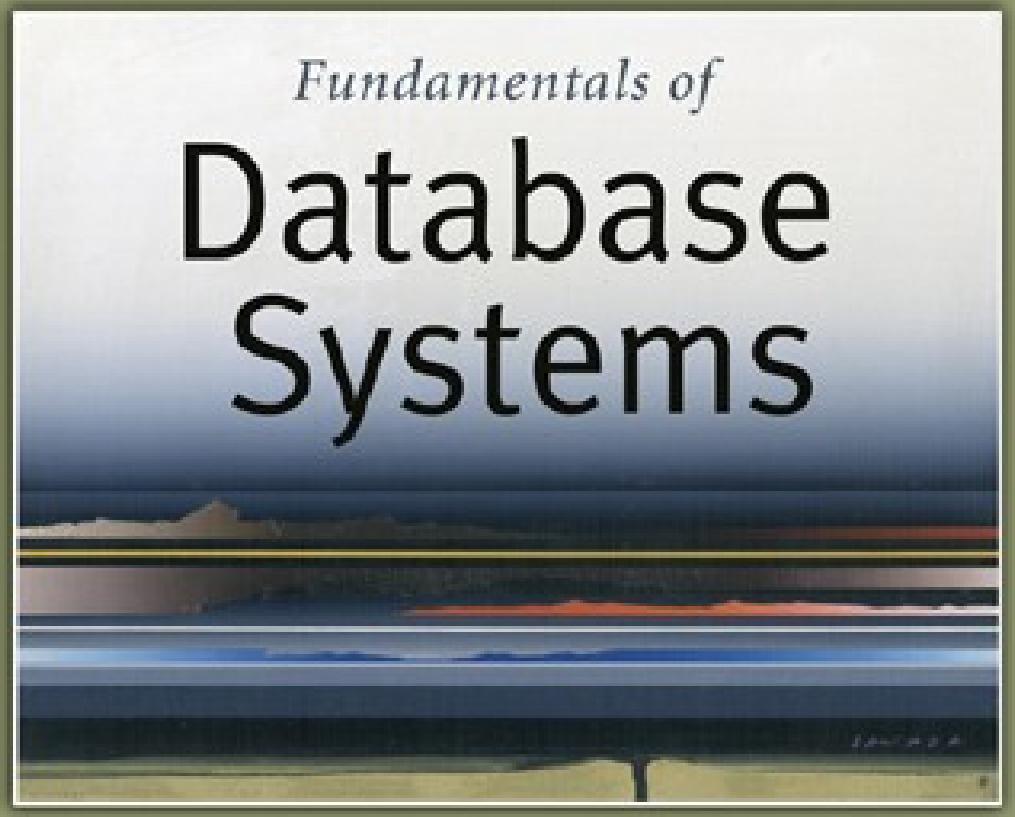
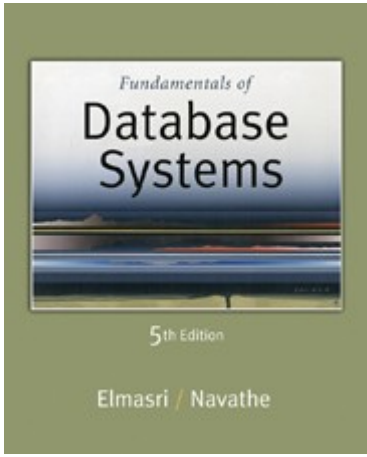


5th Edition

Elmasri / Navathe

Chapter 10

Functional Dependencies and Normalization for Relational Databases



Chapter Outline

- 1 Informal Design Guidelines for Relational Databases
  - 1.1Semantics of the Relation Attributes
  - 1.2 Redundant Information in Tuples and Update Anomalies
  - 1.3 Null Values in Tuples
  - 1.4 Spurious Tuples
- 2 Functional Dependencies (FDs)
  - 2.1 Definition of FD
  - 2.2 Inference Rules for FDs
  - 2.3 Equivalence of Sets of FDs
  - 2.4 Minimal Sets of FDs

Chapter Outline

- 3 Normal Forms Based on Primary Keys
  - 3.1 Normalization of Relations
  - 3.2 Practical Use of Normal Forms
  - 3.3 Definitions of Keys and Attributes Participating in Keys
  - 3.4 First Normal Form
  - 3.5 Second Normal Form
  - 3.6 Third Normal Form
- 4 General Normal Form Definitions (For Multiple Keys)
- 5 BCNF (Boyce-Codd Normal Form)

1 Informal Design Guidelines for Relational Databases (1)

- What is relational database design?
  - The grouping of attributes to form "good" relation schemas
- Two levels of relation schemas
  - The logical "user view" level
  - The storage "base relation" level
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?

Informal Design Guidelines for Relational Databases (2)

- We first discuss informal guidelines for good relational design
- Then we discuss formal concepts of functional dependencies and normal forms
  - - 1NF (First Normal Form)
  - - 2NF (Second Normal Form)
  - - 3NF (Third Normal Form)
  - - BCNF (Boyce-Codd Normal Form)
- Additional types of dependencies, further normal forms, relational design algorithms by synthesis are discussed in Chapter 11

1.1 Semantics of the Relation Attributes

- GUIDELINE 1: Informally, design a relation schema so that it is easy to explain its meaning.
  - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
- *Design a schema that can be explained easily relation by relation.*
- *If the schema is easier to explain the semantics of the relation, the better the relation schema design will be.*

Figure 10.1 A simplified COMPANY relational database schema

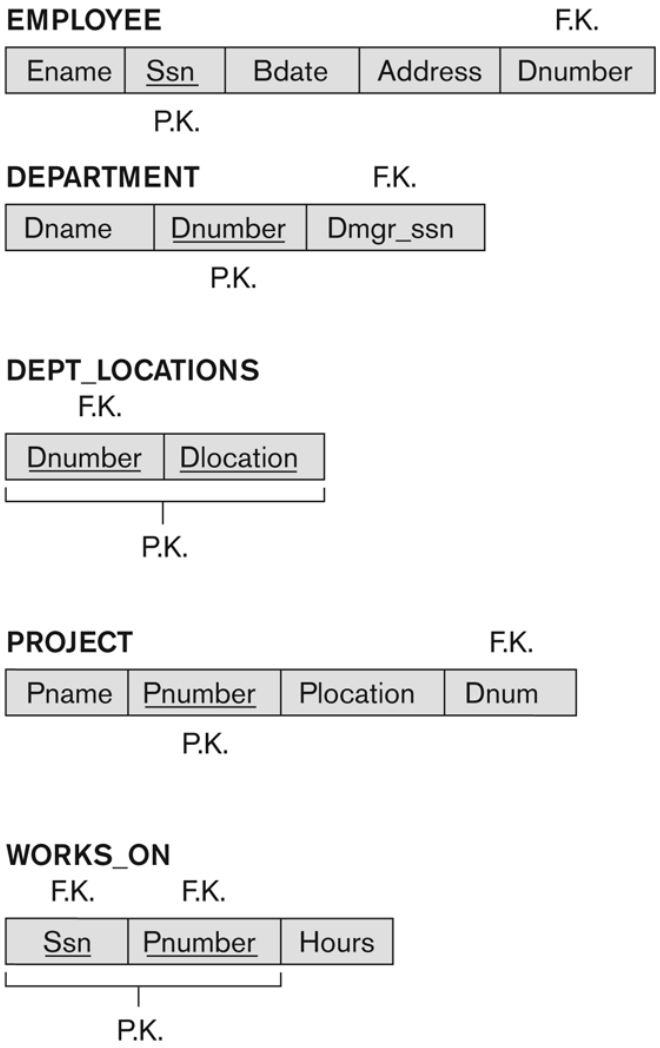


Figure 10.1
A simplified COMPANY relational database schema.

1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

EXAMPLE OF AN INSERT ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Insert Anomaly:
  - Cannot insert a project unless an employee is assigned to it.
- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project.

EXAMPLE OF AN DELETE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Figure 10.3 Two relation schemas suffering from update anomalies

Figure 10.3
Two relation schemas suffering from update anomalies.
(a) EMP_DEPT and
(b) EMP_PROJ.

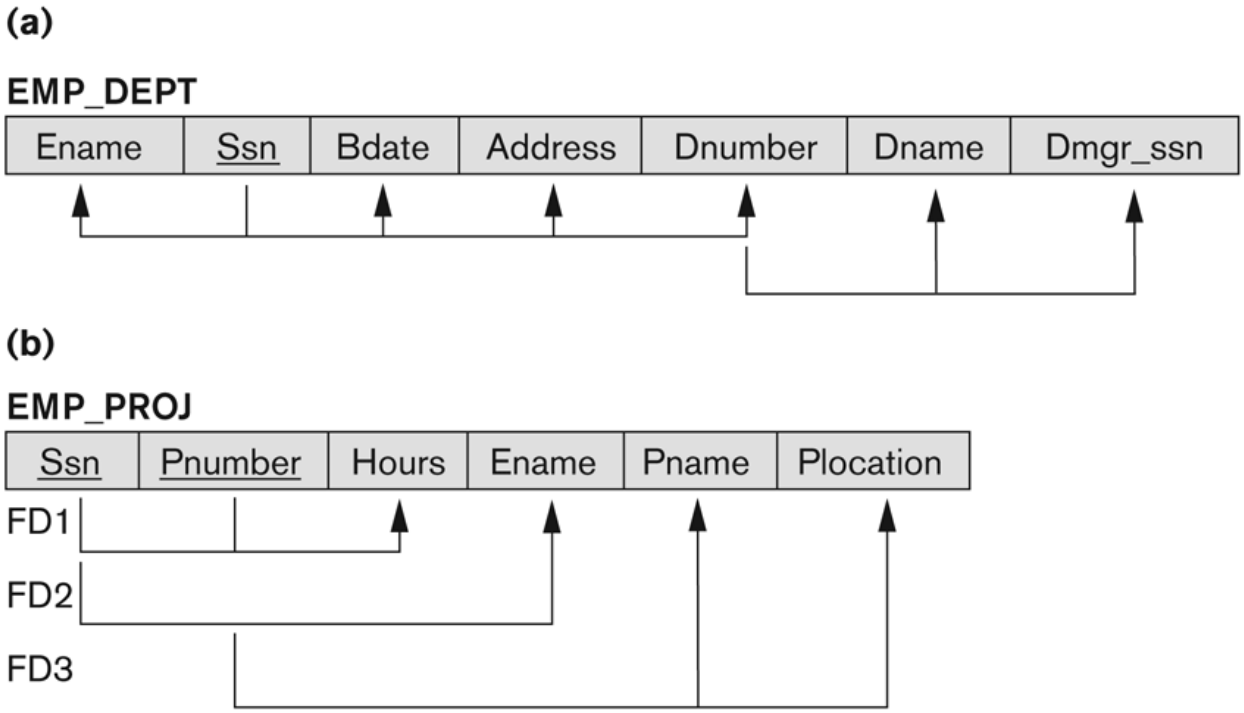


Figure 10.4 Example States for EMP_DEPT and EMP_PROJ

Figure 10.4
Example states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 10.2. These may be stored as base relations for performance reasons.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	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 FireOak, 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

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Guideline to Redundant Information in Tuples and Update Anomalies

- GUIDELINE 2:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

1.3 Null Values in Tuples

- GUIDELINE 3:
  - Relations should be designed such that their tuples will have as few NULL values as possible
  - Attributes that are NULL frequently could be placed in separate relations (with the primary key)
- It is difficult to apply join operation when more NULL are specified, & it is difficult to apply the aggregate functions.
- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

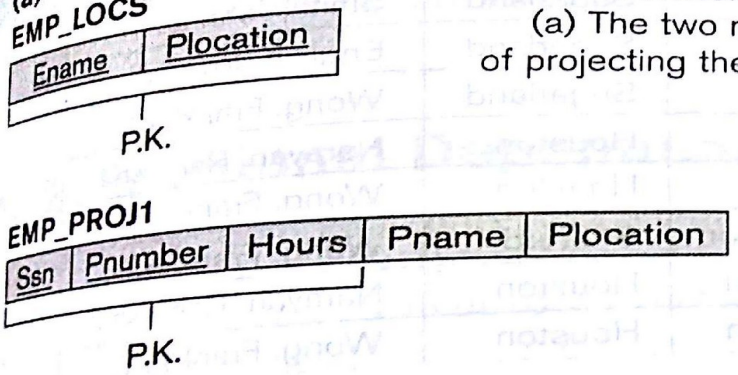
1.4 Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations
- GUIDELINE 4:
  - Design a relation schema so that they can be joined with equality condition on attributes that are (primary key, foreign key).
  - The relations should be designed to satisfy the lossless join condition.
  - No spurious tuples should be generated by doing a natural-join of any relations.

Figure 10.5

Particularly poor design for the EMP_PROJ relation of Figure 10.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 10.4 onto the relations EMP_LOCS and EMP_PROJ1.

(a)



(b)

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Surgarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Surgarland
Wong, Franklin T.	Surgarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

igure 10.6

esult of applying NATURAL JOIN to the tuples above the dotted lines in EMP_PROJ1
d EMP_LOCS of Figure 10.5. Generated spurious tuples are marked by asterisks.

2.1 Functional Dependencies (1)

- Functional dependencies (FDs)
  - Is a constraint between two sets of attributes from the database. $R(A_1,A_2,A_3\text{------}A_n)$
  - Are used to specify *formal measures* of the "goodness" of relational designs
  - And keys are used to define **normal forms** for relations
- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y.(where X & Y are subsets of relation R).

Functional Dependencies (2)

- X -> 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 -> Y in R specifies a *constraint* on all relation instances r(R)
- Written as X -> Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Examples of FD constraints (1)

- Social security number determines employee name
  - SSN -> ENAME
- Project number determines project name and location
  - PNUMBER -> {PNAME, PLOCATION}
- Employee ssn and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} -> HOURS

Examples of FD constraints (2)

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - (since we never have two distinct tuples with t1[K]=t2[K])

2.2 Inference Rules for FDs (1)

- In real life it is difficult to specify all the FDs for a given situation.
- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold
- 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

- Each dept has one manager so that
  - Dept_no->Mgr_SSN
  - Manager has unique phone number so
  - Mgr_SSN->Mgr_Phone
  - So dependencies together imply
  - dept_no->Mgr_phone no need to specify explicitly.

Inference Rules for FDs (2)

- 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)

Inference Rules for FDs (3)

- **CLOSURE that includes all possible dependencies that can be inferred from the given set F.**
- **Closure is the set of all dependencies that include F as well as all the dependencies that can be inferred from F is called closure of F and denoted as** F^{+}.
- **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

Example

- Consider EMP_DEPT relation in 10.3a
- F={SSN→{Ename,bdate,add,dnum},
  dnumber→{dname,DM_SSN}
  }
  - Other dependencies that can be derived are
  - SSN→SSN, Dnum→Dname,
  - SSN→{ Dname, DM_SSN}

Definitions For each set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F:

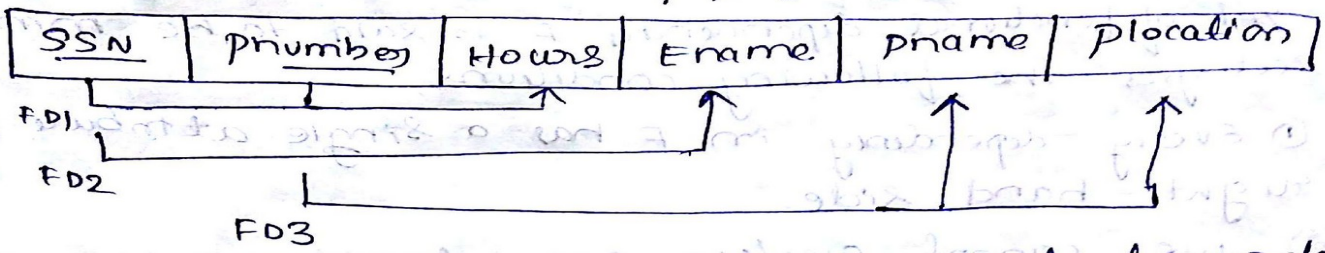
X^+ is called the closure of X under F.

Algorithm 10.1: Determining X^+, the closure of X under F.

```
X+ := X;
Repeat
    old X+ := X+;
    for each functional dependency Y → Z in F do
        if X+ ⊇ Y then X+ := X+ ∪ Z;
until (X+ = old X+);
```

* It starts with setting X^+ to all the attributes in X
* By IR1 WKT all these attributes are functionally dependent on X.
* By using IR3 & IR4 we add attributes to X^+. we keep on continuing all the dependencies in
* Repeat until no more attributes are added to X^+ during a complete cycle.

Eg: Consider relation schema Emp-Proj



Soln: We will specify the set of Functional dependencies that hold on Emp_proj.

$F = \{ SSN \rightarrow ename,\ pnumber \rightarrow \{pname, plocation\},\ \{SSN, pnumber\} \rightarrow Hours \}$.

Using the algorithm 10.1 we calculate the following closure sets with respect to F.

$\{SSN\}^+ = \{SSN, Ename\}$

$\{pnumber\}^+ = \{pnumber, pname, plocation\}$

$\{SSN, pnumber\}^+ = \{SSN, pnumber, Hours, Ename, pname, plocation\}$.

So all the attributes in the right hand side in each line above represents all those attributes are functionally dependent on the set of attributs in the L.H.S.

10.2.3

* Equivalence of Sets of Functional Dependencies

Defn: Two sets of functional dependencies E & F are are equivalent if $E^+ = F^+$: Therefore, equivalence means every FD in E is also inferred from F & every FD in F can be inferred from E, then **E is equivalent to F.**

2.3 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

2.4 Minimal Sets of FDs (1)

- 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 replace any dependency X□A in F with a dependency Y□A where Y is a proper subset of X.

Minimal Sets of FDs (2)

- 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
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set
  - E.g., see algorithms 11.2 and 11.4

Algorithm 10.2: Finding the minimal cover F for a set of dependencies E.

1. Set $F := E$.
2. Replace each functionally dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$.
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.

4. For each remaining functional dependency $X \to A$ in F if $\{F - \{X \to A\}\}$ is equivalent to F, then remove $X \to A$ from F.

Eg: Let the given set of FDs be $E: \{B \to A, D \to A, AB \to D\}$. we have to find the minimum cover of E.

* All the above dependencies are in canonical form so step 1 of 10.3 Alg is completed

Set $F := E$.

$F = \{B \to A, D \to A, AB \to D\}$.

* Step 2 we need to determine if $AB \to D$ has any redundant attribute on the left-hand side.

i.e., $B \to D$ or $A \to D$ ✗

* Since $B \to A$ by augmenting with B on both sides (IR2) we have $BB \to AB$, or $B \to AB$

However $AB \to D$ as given

* By transitive rule (IR3) we get $B \to D$

$B \to AB$
$AB \to D$ } By IR3.
$B \to D$

So $AB \to D$ may be replaced by $B \to D$.

* Now we have a set equivalent to original E, say $\boxed{E': \{B \to A, D \to A, B \to D\}}$ No more redun.

* Identify redundant FD in E' by using transitive rule. $B \to D$ $D \to A$ So $B \to A$ is redundant

(6)

Problems on minimal cover FD

1. Consider the FDs { A□B,B□C,A□C} find the minimal cover.
2. Consider the relation R(A,B,C,D,E) and functional dependencies are F={A□D,BC□AD,C□B,E□A,E□D} find the minimal cover F.
3. Consider the FDs F={AB□C,C□A,BC□D,ACD□B,D□G,D□E,D□G,BE□C,CG□B,CG□D,CE□A,CE□G} find the minimal cover by using minimal cover algorithm.

Problems on Equivalence between sets of Functional dependencies.

1. Consider two sets of FDs, F and G,

F = {A -> B, B -> C, AC -> D} and

G = {A -> B, B -> C, A -> D}

Are F and G equivalent?

2. Consider the relation R(A,C,D,E,H) find F=G where FDs are F={A□C,AC□D,E□AD,E□H} G={A□CD,E□AH}

3. Given below are two sets of FDs for a relation R(A,B,C,D,E) are they equivalent FDs are

F={A->B,AB□C,D□AC,D□E}

G={A□BC,D□AE}

3 Normal Forms Based on Primary Keys

- 3.1 Normalization of Relations
- 3.2 Practical Use of Normal Forms
- 3.3 Definitions of Keys and Attributes Participating in Keys
- 3.4 First Normal Form
- 3.5 Second Normal Form
- 3.6 Third Normal Form

3.1 Normalization of Relations (1)

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
  - It is the process of analysing the given relation schema based on their FDs and Primary keys to achieve the property like
    - Minimising redundancy
    - Minimising insertion, deletion, updation anomalies.
- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

Normalization of Relations (2)

- 2NF, 3NF, BCNF
  - based on keys and FDs of a relation schema
- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation; Chapter 11)
- Loossless join property gurantees that no spurious tuples generation.
- Each FD is represented in some individual relation resulting after decomposition.

3.3 Definitions of Keys and Attributes Participating in Keys (1)

- A **superkey** of a relation schema R = {A1, A2, ..., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]
- Eg:

Manager_ID	Name	Title	Dept_ID

- If we combine the Manager_ID column value to any other given column val then we ll have unique set of values
- Eg : (Manager_ID,Dept_ID) } (Super Key)
- Eg2:(Manager_ID,Name)
- Minimal super key a key with only minimum num of columns that uniquely identify the row.
- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

Definitions of Keys and Attributes Participating in Keys (2)

- If a relation schema has more than one key, each is called a **candidate** key.
  - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.
- Eg: WORKS_ON

SSN	Pnumber	Hours

EMPLOYEE

EID	ENAME	DOB	SSN	DEPTID	MGRID

- **In the above table there are** two unique keys they are EID & SSN so that there are two candidate keys.

FINDING CANDIDATE KEYS

- 1.GIVEN RELATION(A,B,C,D,E,F)
- Functional dependencies are X={A□C,C□D,D□B,E□F} find the candidate keys.
- 2.R(A,B,C,D,E,F,G,H) FDs are
- { CH□G,A□BC,B□CFH,E□A,F□EG}
- Find the candidate keys using closure.

3.2 First Normal Form

- Disallows
  - composite attributes
  - multivalued attributes
  - **nested relations**; attributes whose values for an *individual tuple* are non-atomic

- Eg: nested relation
- EMP_PROJ(SSN,ENAME,{PROJS(PNUMBER,HOURS)})

Figure 10.8 Normalization into 1NF

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Figure 10.9 Normalization nested relations into 1NF

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 10.9
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Eg3: MULTILEVEL NESTING

- IF MORE THAN ONE MULTIVALUED ATTRIBUTE IN one relation must be handled carefully.
- PERSON(SSN#,{car_lic#},{phone#})
- A person has multiple cars, multiple phones.
- How to normalize into 1 normal form.?????
- Solution1:PER_INF(SSN#,car_lic#,phone#) by filling with redundant values.
- Solution2: p1(ssn#,car_lic#)
- P2(ssn#,phone#)

3.3 Second Normal Form (1)

- Uses the concepts of **FDs, primary key**
- Definitions
  - **Prime attribute:** An attribute that is member of the primary key K
  - **Full functional dependency:** a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more
- Examples:
  - {SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold
  - {SSN, PNUMBER} -> ENAME is not a full FD (it is called a partial dependency) since SSN -> ENAME also holds

Second Normal Form (2)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

- R can be decomposed into 2NF relations via the process of 2NF normalization

Figure 10.10 Normalizing into 2NF and 3NF

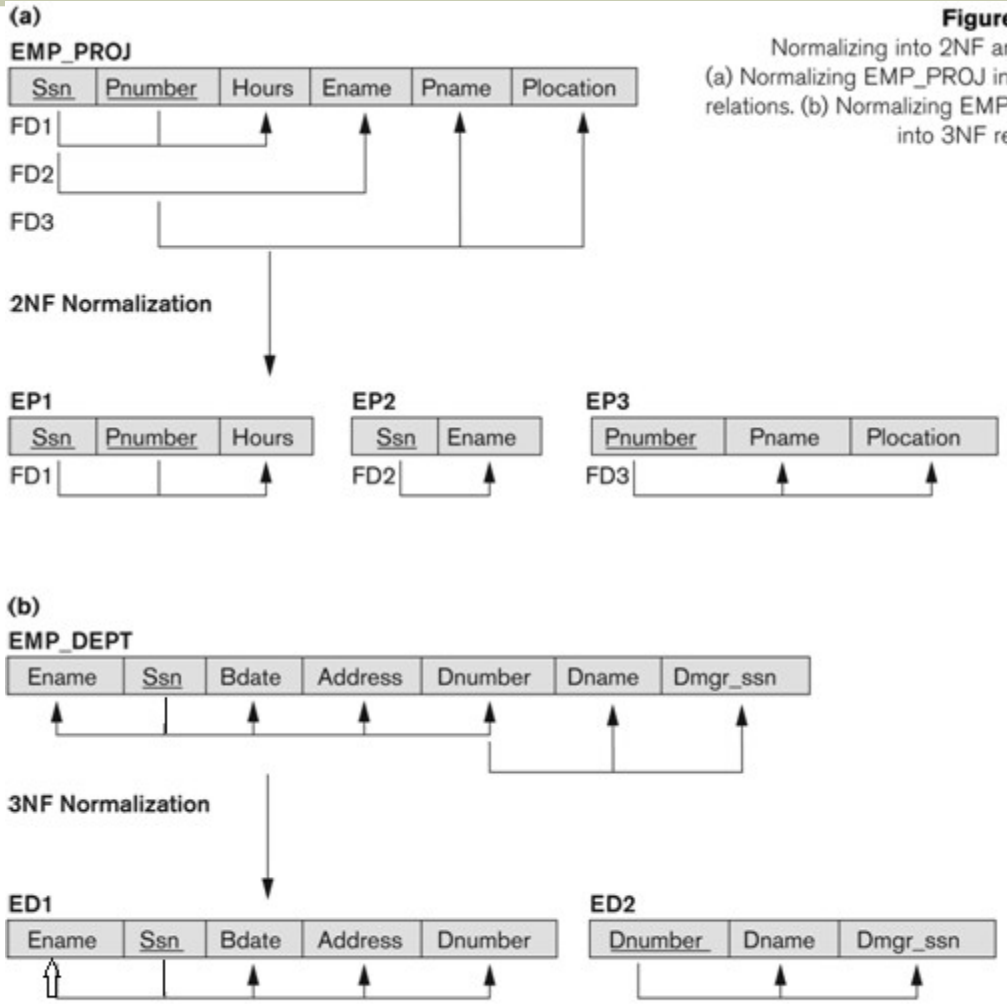


Figure 10.10
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Figure 10.11 Normalization into 2NF and 3NF

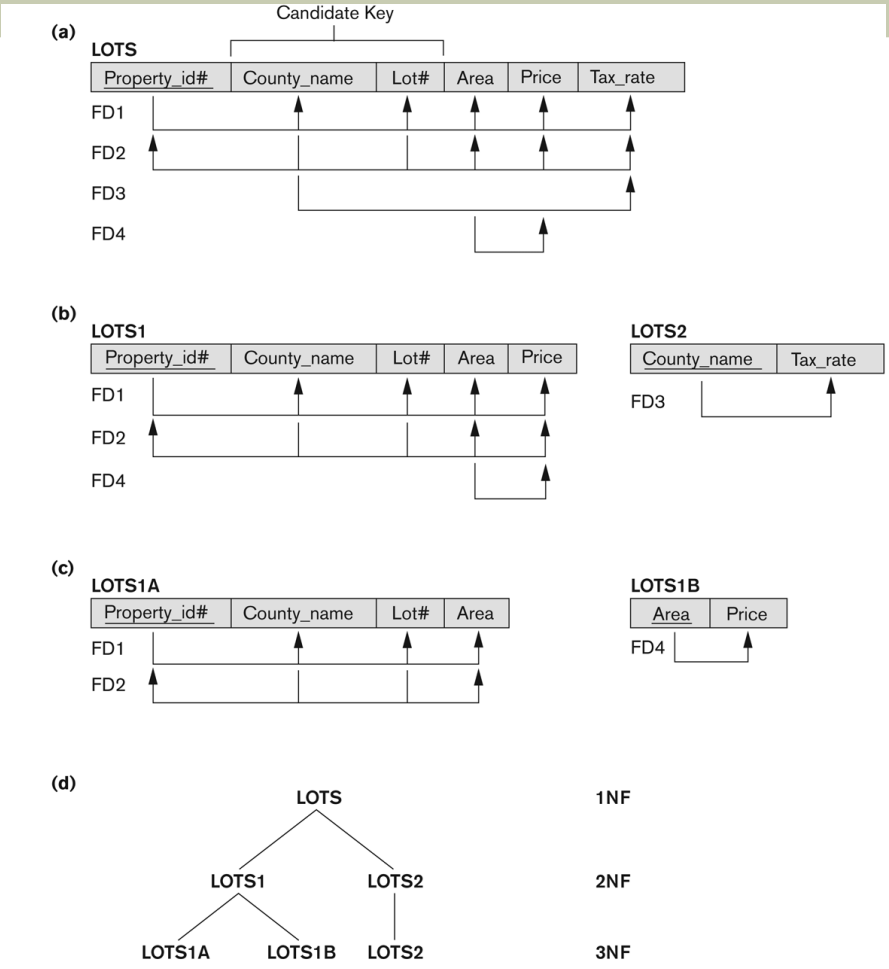


Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

3.4 Third Normal Form (1)

- Definition:
  - **Transitive functional dependency:** a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z
- Examples:
  - SSN -> DMGRSSN is a **transitive** FD
    - Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold
  - SSN -> ENAME is **non-transitive**
    - Since there is no set of attributes X where SSN -> X and X -> ENAME

Third Normal Form (2)

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
  - In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.
  - When Y is a candidate key, there is no problem with the transitive dependency .
  - E.g., Consider EMP (SSN, Emp#, Salary).
    - Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

5 BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD X -> A** holds in R, then **X is a superkey** of R
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- The goal is to have each relation in BCNF (or 3NF)

Figure 10.12 Boyce-Codd normal form

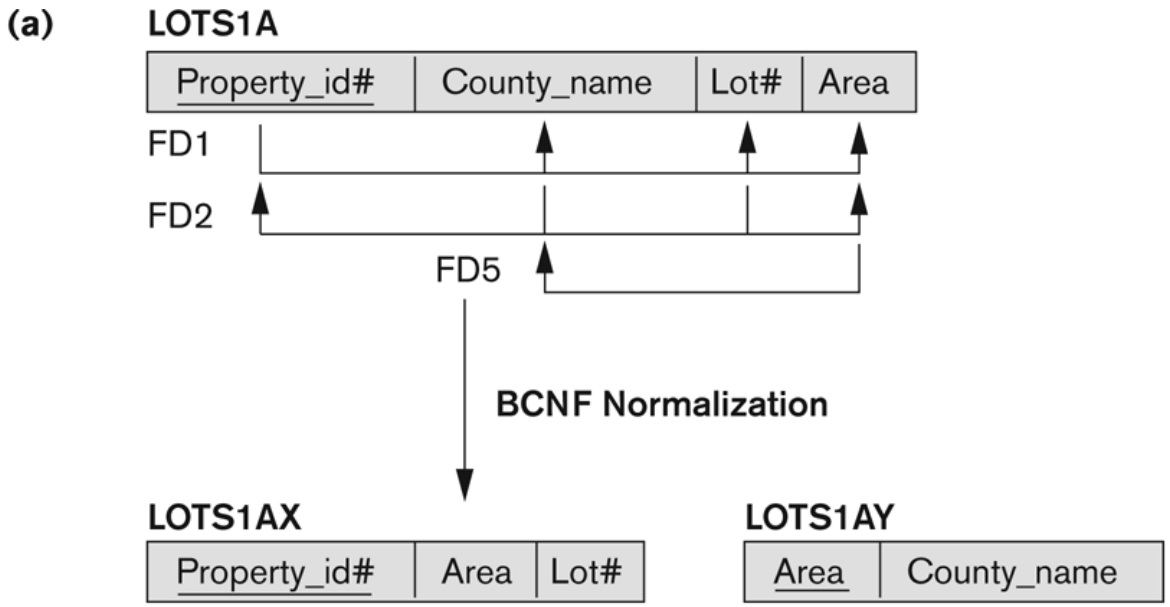


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Figure 10.13 a relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 10.13
A relation TEACH that is in 3NF but not BCNF.

Achieving the BCNF by Decomposition (1)

- Two FDs exist in the relation TEACH:
  - fd1: { student, course} -> instructor
  - fd2: instructor -> course
- {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 10.12 (b).
  - So this relation is in 3NF *but not in* BCNF
- A relation **NOT** in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.
  - (See Algorithm 11.3)

Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
  - {student, instructor} and {student, course}
  - {course, instructor } and {course, student}
  - {instructor, course } and {instructor, student}
- All three decompositions will lose fd1.
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).
- A test to determine whether a binary decomposition (decomposition into two relations) is non-additive (lossless) is discussed in section 11.1.4 under Property LJ1. Verify that the third decomposition above meets the property.

Minimal Cover for a Set of FDs

Minimal Cover for a Set of FDs

- ***Minimal cover*** **G for a set of FDs F:**
  - Closure of F = Closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- **Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.**
- **e.g., A $\rightarrow$ B, ABCD $\rightarrow$ E, EF $\rightarrow$ GH, ACDF $\rightarrow$ EG has the following minimal cover:**
  - A $\rightarrow$ B, ACD $\rightarrow$ E, EF $\rightarrow$ G and EF $\rightarrow$ H
- **Minimal Cover implies Lossless-Join, Defendency Preserving Decomposition.**
  - Start with M.C. of F, do the decomposition from Minimal Cover of F

Functional dependencies

Our goal:

given a set of FD set, F, find an alternative FD set, G that is:

smaller

equivalent

Bad news:

Testing F=G (F+ = G+) is computationally expensive

Good news:

Canonical Cover algorithm:

given a set of FD, F, finds minimal FD set equivalent to F

Minimal: can't find another equivalent FD set w/ fewer FD's

Minimal cover

F = { AB -> C, C -> A, BC -> D, ACD -> B, D -> E, D -> G, BE -> C, CG -> B,
CG -> D, CE -> A, CE -> G}

Notice that we have single attribute on the RHS in all FDs, we need to look for extraneous (redundant) attributes on the LHS and also look for FDs that are redundant.

Canonical Cover Algorithm

Given:

F = { A → BC,
B → CE,
A → E,
AC→ H,
D → B}

- Fc = F
- No G that is equivalent to F and is smaller than Fc

Determines canonical cover of F:

Fc = { A → BH,
B → CE,
D → B}

Another example:

F = { A → BC,
B → C,
A → B,
AB→ C,
AC → D}

⟶ CC Algorithm ⟶ Fc = { A → BD,
B →C}

Canonical Cover Algorithm

Basic Algorithm

ALGORITHM CanonicalCover (X: FD set)
BEGIN
REPEAT UNTIL STABLE
(1) Where possible, apply Additivity rule (A's axioms)
(e.g., A →BC, A→CD becomes A→BCD)
(2) remove "extraneous attributes" from each FD
(e.g., AB→C, A→B becomes
A→B, B→C
i.e., A is extraneous in AB→C)

Extraneous Attributes

(1) Extraneous is RHS?

e.g.: can we replace A → BC with A→C?

(i.e. Is B extraneous in A →BC?)

(2) Extraneous in LHS ?

e.g.: can we replace AB →C with A → C ?

(i.e. Is B extraneous in AB→C?)

Simple but expensive test:

1. Replace A → BC (or AB → C) with A → C in F

F2 = F - {A →BC} U {A →C}

or

F - {AB→C} U {A →C}

2. Test if F2+ = F+ ?

if yes, then B extraneous

Extraneous Attributes

A. RHS: Is B extraneous in A →BC?

step 1: F2 = F - {A →BC} U {A →C}
step 2: F+ = F2+ ?

To simplify step 2, observe that F2+ $\subseteq$ F+

Why? Have effectively removed A→B from F i.e., not new FD's in F2+)

When is F+ = F2+ ?

Ans. When (A→B) in F2+

Idea: if F2+ includes: A→B and A→C, then it includes A→BC

Extraneous Attributes

B. LHS: Is B extraneous in A B→C ?

step 1: F2 = F - {AB →C} U {A →C}
step 2: F+ = F2+ ?

To simplify step 2, observe that F+ $\subseteq$ F2+

Why? A→C "implies" AB→C. therefore all FD's in F+ also in F2+. i.e., there may be new FD's in F2+)
But AB→C does not "imply" A→C

When is F+ = F2+ ?

Ans. When (A→C) in F+

Idea: if F+ includes: A→C then it will include all the FD's of F2+.

Extraneous attributes

A. RHS :

Given F = {A→BC, B→C} is C extraneous in A →BC?

why or why not?

Ans: yes, because

A→C in { A→B, B→C}+

Proof. 1. A→ B
2. B →C
3. A→C transitivity using Armstrong's axioms

Extraneous attributes

B. LHS :

Given F = {A→B, AB→C} is B extraneous in AB →C?

why or why not?

Ans: yes, because

A→C in F+

Proof. 1. A→ B
2. AB →C
3. A→C using pseudotransitivity on 1 and 2

Actually, we have AA→C but {A, A} = {A}

Canonical Cover Algorithm

ALGORITHM CanonicalCover (F: set of FD's)
BEGIN
REPEAT UNTIL STABLE
(1) Where possible, apply Additivity rule (A's axioms)

(2) Remove all extraneous attributes:
a. Test if B extraneous in A→ BC
(B extraneous if
(A→B) in (F - {A→BC} U {A→C})+)
b. Test if B extraneous in AB→C
(B extraneous in AB→C if
(A→C) in F+)

Canonical Cover Algorithm

Example: determine the canonical cover of

$F = \{A \rightarrow BC, B \rightarrow CE, A \rightarrow E\}$

Iteration 1:

a. $F = \{ A \rightarrow BCE, B \rightarrow CE\}$

b. Must check for upto 5 extraneous attributes

- B extraneous in $A \rightarrow BCE$? No
- C extraneous in $A \rightarrow BCE$?

  yes: $(A \rightarrow C)$ in $\{ A \rightarrow BE, B \rightarrow CE\}$

  1. $A \rightarrow BE$ -> 2. $A \rightarrow B$ -> 3. $A \rightarrow CE$ -> 4. $A \rightarrow C$
- E extraneous in $A \rightarrow BE$?

Canonical Cover Algorithm

Example: determine the canonical cover of

F = {A →BC, B→CE, A→E}

Iteration 1:

a. F = { A→BCE, B→CE}

b. Must check for upto 5 extraneous attributes

- B extraneous in A→BCE? No
- C extraneous in A → BCE? Yes
- E extraneous in A→BE?

  1. A→B -> 2. A→CE -> A→E

- E extraneous in B→CE No
- C extraneous in B→CE No

Iteration 2:

a. F = { A → B, B→ CE}

b. Extraneous attributes:

- C extraneous in B → CE No
- E extraneous in B →CE No

DONE

Canonical Cover Algorithm

Find the canonical cover of

F = { A → BC,
B → CE,
A → E,
AC → H,
D → B}

Ans: Fc= { A→BH, B→ CE, D→B}

Canonical Cover Algorithm

Find two different canonical covers of:

$$F = \{ A \rightarrow BC,\quad B \rightarrow CA,\quad C \rightarrow AB \}$$

Ans:

$$Fc1 = \{ A \rightarrow B,\ B \rightarrow C,\ C \rightarrow A \}$$

and

$$Fc2 = \{ A \rightarrow C,\ B \rightarrow A,\ C \rightarrow B \}$$

Minimal cover (example discussed in class)

F = { AB -> C, C -> A, BC -> D, ACD -> B, D -> E, D -> G, BE -> C, CG -> B,
CG -> D, CE -> A, CE -> G}

Notice that we have single attribute on the RHS in all FDs, we need to look for extraneous (redundant) attributes on the LHS and also look for FDs that are redundant.

Apply either of the rules below to find extraneous attributes on the LHS:

1. In an FD $\{\alpha \rightarrow \beta\}$ from set F, attribute $A \in \alpha$ is extraneous in α
   if F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
2. To test if attribute $A \in \alpha$ is extraneous in α
   Step 1: compute $(\{\alpha\} - A)+$ using the dependencies in F
   Step 2: check that $(\{\alpha\} - A)+$ contains A; if it does, A is extraneous

---

* Consider ACD -> B to see if any of the three attributes on the LHS is extraneous. Start with A.

Applying Rule 2, we need to compute ({ACD} – A)+ using the dependencies in F and check if the result contains A; if it does, attribute A is extraneous.
Computing CD+, we get ACDEGB which contains A and therefore, A is extraneous. Also the closure contains B, which tells us that CD -> B holds.

Alternatively, using Rule 1, we need to prove that F logically implies CD -> B in place of ACD -> B. To show that we can get CD -> B from F,

Start with C -> A (given in F)
CD -> ACD (augment with CD)
ACD -> B (given in F)
CD -> B (transitivity)

* Look for redundant FDs in set F.

- is CE -> A redundant? Yes, since we have C -> A in F, we can get CE -> A through augmentation.
- Is CG -> B redundant? Yes, since we have CG -> D, C -> A, and ACD -> B in F.

No more redundant FDs in F.

Minimal Cover 1: {AB -> C, **C -> A**, BC -> D, **CD -> B**, D -> E, D -> G, BE -> C,
CG -> D, CE -> G}

Minimal Cover 2 : {AB -> C, **C -> A**, **BC -> D**, D -> E, **D -> G**, BE -> C,
CG -> B, CE -> G}
Obtained by eliminating redundant FDs, CE -> A, CG -> D, ACD -> B

- is CE -> A redundant? Yes, since we have C -> A in F, we can get CE -> A through augmentation.
- is CG -> D redundant? Yes, since CG -> B, BC -> D in F
- is ACD -> B redundant? Yes, since D -> G, CG -> B

The difference between the two covers is that in the first one, we chose to keep **CG -> D** which helps us to infer CG -> B making it redundant. In the second one, since we kept **CG -> B**, we

could eliminate CG -> D and ACD -> B as redundant. We need one of {**CG -> D, CG -> B**} in the minimal cover – can not eliminate both.

Equivalence between sets of Functional Dependencies:

Consider two sets of FDs, F and G,

F = {A -> B, B -> C, AC -> D} and

G = {A -> B, B -> C, A -> D}

Are F and G equivalent?

To determine their equivalence, we need to prove that F+ = G+. However, since computing F+ or G+ is computationally expensive we take a short cut. We can conclude that F and G are equivalent, if we can prove that all FDs in F can be inferred from the set of FDs in G and vice versa.

For our example above, let us see if we can infer all FDs in F using G; we will use attribute closure to determine that.

Take the attributes from the LHS of FDs in F and compute attribute closure for each using FDs in G:

A+ using G = ABCD; A -> A; **A -> B**; A -> C; A -> D;

B+ using G = BC; B -> B; **B -> C**;

AC+ using G = ABCD; AC -> A; AC -> B; AC -> C; **AC -> D**;

Notice that all FDs in F (highlighted) can be inferred using FDs in G.

To see if all FDs in G are inferred by F, compute attribute closure for attributes on the LHS of FDs in G using FDs in F:

A+ using F = ABCD; A -> A; **A -> B**; A -> C; **A-> D**;

B+ using F = BC; B -> B; **B -> C**;

Since all FDs in F can be obtained from G and vice versa, we conclude that F and G are equivalent.

A Counter Example:

F = {A -> B, A ->C}

G = {A -> B, B -> C}

F and G are not equivalent, because B -> C in G is not inferred from the FDs in F.

A+ using G = ABC;

Whereas,

A+ using F = ABC; but B+ using F = B, indicating B -> C in G is not inferred using the FDs from F.

Practice Problem Set #5

CSC 261/461 (Database Systems), Spring 2018,
University of Rochester
02/28/2018

Problem 1

`Refer: Section 15.1.2`
Let $F = \{A \to C, AC \to D, E \to AD, E \to H\}$.
Let $G = \{A \to CD, E \to AH\}$.
Show that:

1. G covers F
2. F covers G
3. F and G are equivalent

Solution:

1. Show: G covers F or ($F \subseteq G^+$)

   let's check each FD in F:

   - $A \to C$
     So, let's find A^+ in G.
     A^+ in G = $\{ACD\}$ which includes C. So, continue;
   - $AC \to D$
     Let's get AC^+ in G,
     AC^+ in G = $\{ACD\}$ which contains D. So, continue;
   - $E \to AD$, $E \to H$
     Let's get E^+ in G,
     E^+ in G = $\{EACDH\}$ which contains AD and H. So, Done;

   We found that every dependency in F can be inferred from G.
   So, we can say that G covers F.

2. Show: F covers G or ($G \subseteq F^+$)

   let's check each FD in G:

   - $A \to CD$
     So, let's find A^+ in F.
     A^+ in F = $\{ACD\}$ which includes CD. So, continue;
   - $E \to AH$
     Let's get E^+ in F,
     E^+ in F = $\{EACDH\}$ which contains AH. So, continue;

   We found that every dependency in G can be inferred from F.
   So, we can say that F covers G .

3. As G covers F (from 1) and F covers G (from 2), so, F and G are equivalent ($F \equiv G$).

Problem 2

Refer: Section 15.1.3
Let the relation R be R(A,B,C,D,E) and the given set of FDs be
$F : \{A \to D, BC \to AD, C \to B, E \to A, E \to D\}$.
Find the minimal cover of F

Solution:

- Step 1: Convert all dependencies in canonical form (that is, they have only one attribute on the right hand side).

  $\{A \to D, BC \to A, BC \to D, C \to B, E \to A, E \to D\}$

- Step 2: If left-hand-side contains multiple attributes, check if any of those can be removed. We have $BC \to A$, so we will check if $B \to A$ or $C \to A$ is sufficient. For this, we need to get B^+ and C^+. $B^+ = \{B\}$, and $C^+ = \{ABCD\}$. As, B^+ does not have A, so, C canot be removed. But C^+ contains A. So, We do not need B and $C \to A$ is sufficient.

  We get:
  $\{A \to D, C \to A, BC \to D, C \to B, E \to A, E \to D\}$

  Similarly, we can get rid of B from $BC \to D$ as C^+ contains D
  We get:
  $\{A \to D, C \to A, C \to D, C \to B, E \to A, E \to D\}$

- Step 3: This is the fun part. We will check if we can get rid of any of these dependencies completely. For that, we will iterate over each FD, $X \to Y$. Calculate X^+ without considering $X \to Y$. If we find $Y \subset X^+$, we can infer $X \to Y$ is redundant (because we found some other FD/FDs Z such that, $X \to Z$ and $Z \to Y$).

  So, let's do it.

  - $A \to D$:
    Calcualte A^+ without $A \to D$:
    We get: $A^+ = \{A\}$.. So, A^+ does not contain D. We cannot remove it.
  - $C \to A$:
    Calcualte C^+ without $C \to A$:
    We get: $C^+ = \{CBD\}$.. So, C^+ does not contain A. We cannot remove it.
  - $C \to D$:
    Calcualte C^+ without $C \to D$:
    We get: $C^+ = \{CAD\}$.. So, C^+ contains D. We **can** remove it.

    Current cover: $\{A \to D, C \to A, C \to B, E \to A, E \to D\}$
  - $C \to B$:
    Calcualte C^+ without $C \to B$:
    We get: $C^+ = \{CAD\}$.. So, C^+ does not contain B. We cannot remove it.
  - $E \to A$:
    Calcualte E^+ without $E \to A$:
    We get: $E^+ = \{ED\}$.. So, E^+ does not contain A. We cannot remove it.
  - $E \to D$:
    Calcualte E^+ without $E \to D$:

We get: $E^+ = \{EAD\}$.. So, E^+ contains D. We **can** remove it.

Current cover: $\{A \to D, C \to A, C \to B, E \to A\}$

So, minimal cover is:
$\{A \to D, C \to A, C \to B, E \to A\}$

or

$\{A \to D, C \to AB, E \to A\}$

Problem 3

Let the relation schema R(A,B,C,D) is given. For each of the following set of FDs do the following:

i) indicate all the BCNF violations. Do not forget to consider FD's that are not in the given set. However, it is not required to give violations that have more than one attribute on the right side.

ii) Decompose the relations, as necessary, into collections of relations that are in BCNF.

1. FDs AB→ C, C→D, and D→A

2. FDs B→C, B→D

3. FDs AB→C, BC→D, CD→A, AD→B

Solution:

1. There are 14 nontrivial dependencies, They are: C→A, C→D, D→A, AB→D, AB→ C, AC→D, BC→A, BC→D, BD→A, BD→C, CD→A, ABC→D, ABD→C, and BCD→A.

   There are three keys — AB, BC, and BD. Thus, any dependency above that does not have one of these pairs on the left is a BCNF violation. These are: C→A, C→D, D→A, AC→D, and CD→A.

   One choice is to decompose using the violation C→D. Using the above FDs, we get ACD (Because of C→D and C→A) and BC as decomposed relations. BC is surely in BCNF, since any two-attribute relation is. we discover that ACD is not in BCNF since C is its only key. We must further decompose ACD into AD and CD. Thus, the three relations of the decomposition are BC, AD, and CD.

2. By computing the closures of all 15 nonempty subsets of ABCD, we can find all the nontrivial FDs. They are B→C, B→D, AB→C, AB→D, BC→D, BD→C, ABC→D and ABD→C. From the closures we can also deduce that the only key is AB. Thus, any dependency above that does not contain AB on the left is a BCNF violation. These are: B→C, B→D, BC→D and BD→C.

   One choice is to decompose using the violation B→C. Using the above FDs, we get BCD and AB as decomposed relations. AB is surely in BCNF, since any two-attribute relation is. we discover that BCD is in BCNF since B is its only key and the projected FDs all have B on the left side. Thus the two relations of the decomposition are AB and BCD.

3. There are 12 nontrivial dependencies, including the four given ones and the eight derived ones. They are AB→C, BC→D, CD→A, AD→B, AB→D, AD→C, BC→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

We also found out that the keys are AB, AD, BC, and CD. Thus, any dependency above that does not have one of these pairs on the left is a BCNF violation. However, all of the FDs contain a key on the left so there are no BCNF violations. No decomposition is necessary since all the FDs do not violate BCNF.