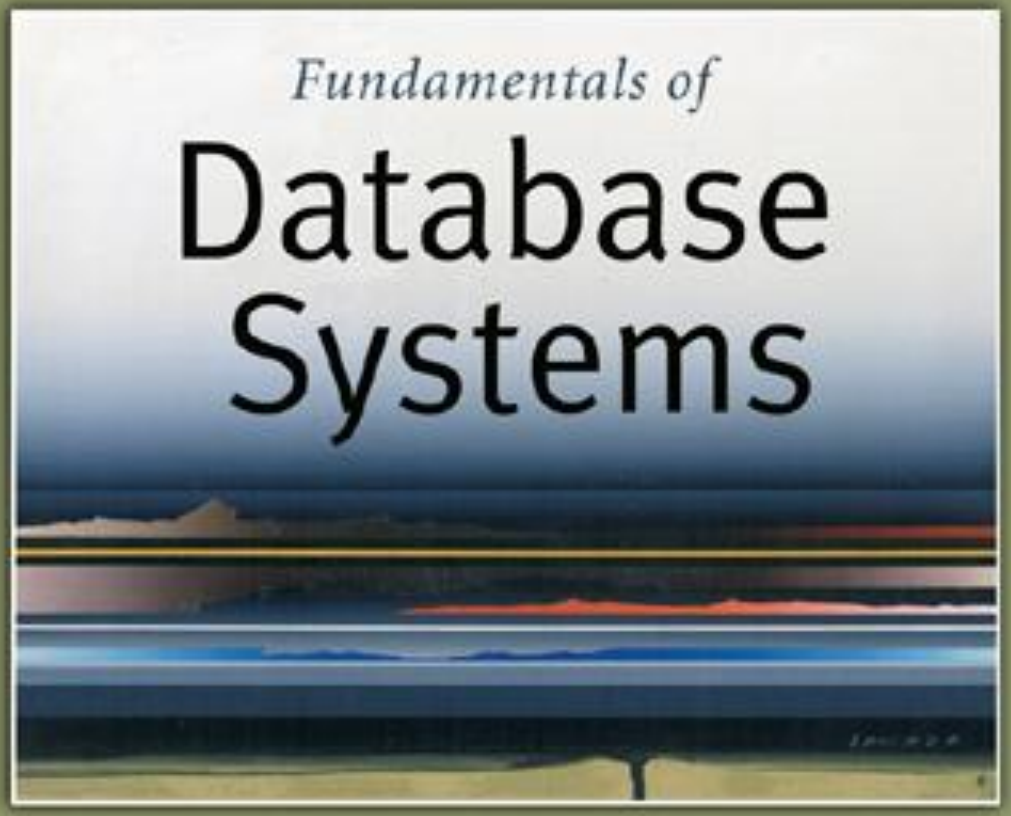
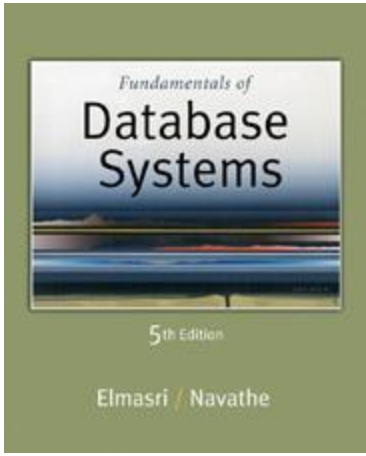


5th Edition

Elmasri / Navathe

Module 4

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, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - 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
  - Entity and relationship attributes should be kept apart as much as possible.
- Bottom Line: *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*

Figure 10.1 A simplified COMPANY relational database schema

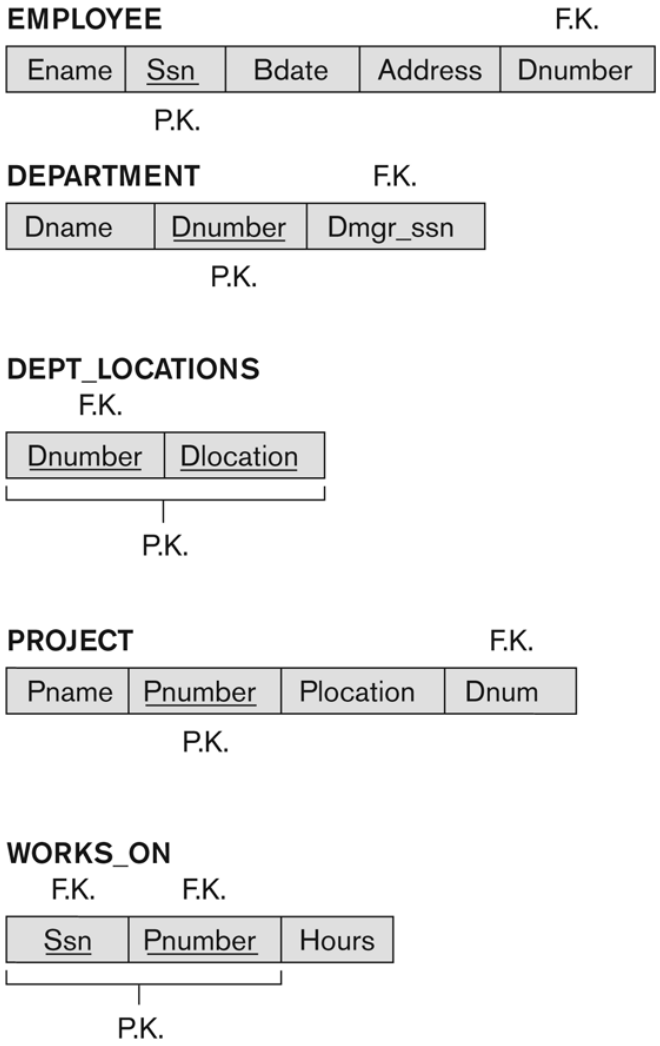


Figure 10.1
A simplified COMPANY relational database schema.

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

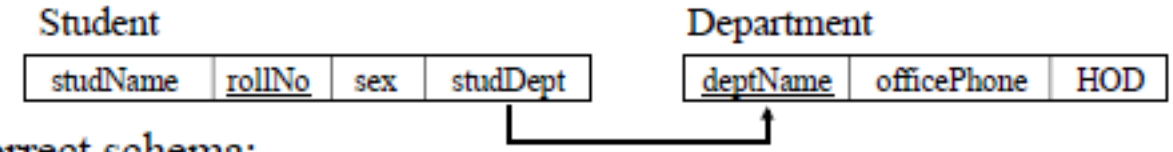
Example on bad schema and anamolies

An Example

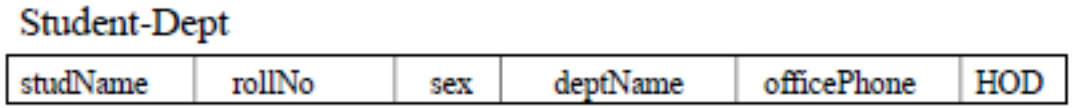
student relation with attributes: studName, rollNo, sex, studDept
department relation with attributes: deptName, officePhone, hod

Several students belong to a department.
studDept gives the name of the student's department.

Correct schema:



Incorrect schema:



What are the problems that arise ?

Problems with bad schema

Redundant storage of data:

Office Phone & HOD info - stored redundantly

- once with each student that belongs to the department
- wastage of disk space

A program that updates Office Phone of a department

- must change it at several places
  - more running time
  - error - prone

Transactions running on a database

- must take as short time as possible to increase transaction throughput

Update Anomalies

Another kind of problems with bad schema
Insertion anomaly:
  No way of inserting info about a new department unless
  we also enter details of a (dummy) student in the department

Deletion anomaly:
  If all students of a certain department leave
  and we delete their tuples,
  information about the department itself is lost

Update Anomaly:
  Updating officePhone of a department
  - value in several tuples needs to be changed
  - if a tuple is missed - inconsistency in data

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.

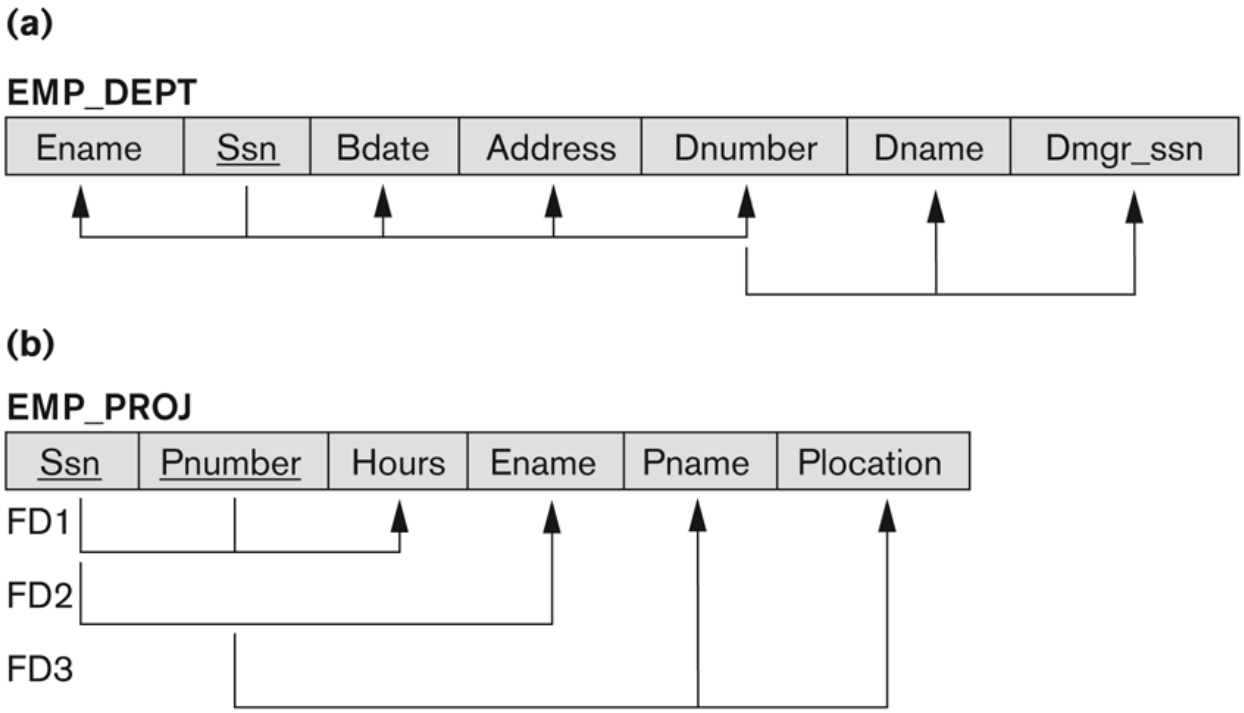


Figure 10.4 Example States for EMP_DEPT and EMP_PROJ

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)
- 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:
  - 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.

Example for generation of Spurious tuples due to bad schema

(a)

EMP_LOCS

Ename	Plocation

P.K.

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

P.K.

Figure 15.5
Particularly poor design for the EMP_PROJ relation in Figure 15.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 15.4 onto the relations EMP_LOCS and EMP_PROJ1.

(b)

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
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

Example for generation of Spurious tuples due to bad schema

	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.

Figure 15.6
Result of applying NATURAL JOIN to the tuples above the dashed lines in EMP_PROJ1 and EMP_LOCS of Figure 15.5. Generated spurious tuples are marked by asterisks.

Spurious Tuples (2)

- There are two important properties of decompositions:
  a) Non-additive or losslessness of the corresponding join
  b) Preservation of the functional dependencies.

- Note that:
  - Property (a) is extremely important and *cannot* be sacrificed.
  - Property (b) is less stringent and may be sacrificed. (See Chapter 11).

2.1 Functional Dependencies (1)

- Functional dependencies (FDs)
  - Are used to specify *formal measures* of the "goodness" of relational designs
  - And keys are used to define **normal forms** for relations
  - Are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes
- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y

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])

Example of FD's

Functional Dependencies – Examples

Consider the schema:

Student(studName, rollNo, sex, dept, hostelName, roomNo)

Since rollNo is a key, rollNo → {studName, sex, dept, hostelName, roomNo}

Suppose that each student is given a hostel room exclusively, then

hostelName, roomNo → rollNo

Suppose boys and girls are accommodated in separate hostels, then

hostelName → sex

Does Sex → hostelName?

FDs are additional constraints that can be specified by designers

2.2 Inference Rules for FDs (1)

- 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

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)

Exercise

Which of the following FD's can be observed as **not** holding on the given data instance?: Employee

EmpId	EmpName	EmpDob	HomeCity	EmpAddress	EmpMobileNum
E101	Rakesh	01-01-1990	Delhi	D1/123	91234-12345
E102	Mahesh	02-04-1991	Pune	P1/456	92345-23456
E103	Vijay	10-10-1990	Chennai	C2/345	93456-34567
E104	Mahesh	05-06-1995	Jaipur	D2/345	91345-12345
E105	Manish	08-09-1980	Delhi	D7/234	94123-23456
E106	Rakesh	19-12-1992	Pune	P5/465	94234-45678
E107	Rakesh	05-06-1995	Jaipur	J2/980	91432-12345
E108	Vijay	02-08-1991	Madurai	C5/123	93421-98765

Empid determines *EmpAddress*

EmpMobileNum determines *HomeCity*

{EmpDob, Homecity} determines *EmpAddress*

{EmpName, HomeCity} determines *EmpAddress*

Inference Rules for FDs (3)

- **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

Computing X^+_F

We compute a sequence of sets $X_0, X_1, \ldots$ as follows:

$X_0 = X$; // X is the given set of attributes

$X_{i+1} = X_i \cup \{A \mid$ there is a FD $Y \rightarrow Z$ in F such that $Y \subseteq X_i$ and $A \in Z\}$

To get new attributes into X_{i+1}, we use Transitive Rule and we can only use that!

Since $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots \subseteq X_i \subseteq X_{i+1} \subseteq \ldots \subseteq R$, and R is finite, There is an integer i such that $X_i = X_{i+1} = X_{i+2} = \ldots$

X^+_F is equal to such X_i.

Computing X^+_F can be done in polynomial time

Exercise on closure

The functional dependencies for a relation R(A, B, C, D, E, F, G, H, I) are

$S = \{ AD \rightarrow G, A \rightarrow F, H \rightarrow B, FB \rightarrow D, D \rightarrow C, I \rightarrow AH \}$.

Which of the following represents I^+ ?

{I, A, H}

{I, A, H, F}

{I, A, H, F, B}

{I, A, H, F, B, C, D, G}

Which is the candidate key of R?

Exercise on closure

The functional dependencies for a relation R ' (A, B, C, D, E, F, G, H, I) are

S ' = {AD → G, A → F, H → B, FB → E, D → C, I → AH}.

Which of the following represents I^+ ?

{I, A, H}

{I, A, H, F, B}

{I, A, H, F, B, C, D, G}

{I, A, H, F, B, E}

Exercise on computing closure

Compute the closure of the following set F of functional dependencies for relation schema $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

List the candidate keys for R.

members of F^+ are found.

Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$.

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$	(decomposition, transitive)
Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$	(union, decomposition, transitive)
Since $A \rightarrow A$, we have	(reflexive)
$A \rightarrow ABCDE$ from the above steps	(union)
Since $E \rightarrow A$, $E \rightarrow ABCDE$	(transitive)
Since $CD \rightarrow E$, $CD \rightarrow ABCDE$	(transitive)
Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCDE$	(augmentative, transitive)
Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.	

Contd..

Therefore, any functional dependency with A, E, BC, or CD on the left hand side of the arrow is in F^+, no matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F^+ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A* \rightarrow \alpha$, $BC* \rightarrow \alpha$, $CD* \rightarrow \alpha$, $E* \rightarrow \alpha$ where α is any subset of $\{A, B, C, D, E\}$. The candidate keys are A, BC, CD, and E.

Attribute Closures − An Example

Consider a scheme R and the FDs: (Data redundancy exists in R)

R = (rollNo, name, advisorId, advisorName, courseId, grade)

FDs = { rollNo → name; rollNo → advisorId;
advisorId → advisorName;
rollNo, courseId → grade }

$\{rollNo\}^+ = \{rollNo, name, advisorId, advisorName\}$

$\{rollNo, courseId\}^+ = \{rollNo, name, advisorId, advisorName, courseId, grade\} = R$

So {rollNo, courseId} is the key for R.

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 remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F.

Minimal Dependency/ Minimal cover

Canonical covers or Minimal covers

It is of interest to reduce a set of FDs F into a 'standard' form F′ such that F′ is equivalent to F.

We define that a set of FDs F is in '*minimal form*' if

(i) the rhs of any FD of F is a single attribute

(ii) there are no redundant FDs in F

that is, there is no FD $X \rightarrow A$ in F

s.t $(F - \{X \rightarrow A\})$ is equivalent to F

(iii) there are no redundant attributes on the lhs of any FD in F

that is, there is no FD $X \rightarrow A$ in F s.t there is $Z \subset X$ for which

$F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ is equivalent to F

Minimal Covers

useful in obtaining a lossless, dependency-preserving

decomposition of a scheme R into 3NF relation schemas

Algorithm for finding Minimal Dependency

Algorithm for computing a minimal cover

R – given Schema or set of attributes; F – given set of FDs on R

Step 1: G := F

Step 2: Replace every fd of the form $X \rightarrow A_1A_2A_3\ldots A_k$ in G
by $X \rightarrow A_1$; $X \rightarrow A_2$; $X \rightarrow A_3$; … ; $X \rightarrow A_k$

Step 3: For each fd $X \rightarrow A$ in G do
for each B in X do
if $(G - \{X \rightarrow A\} + \{(X - B) \rightarrow A\})^+ = F^+$ then
replace $X \rightarrow A$ by $(X - B) \rightarrow A$

Step 4: For each fd $X \rightarrow A$ in G do

Minimal Dependency exercuse

Let the given set of FDs be $E: \{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$. We have to find the minimal cover of E.

- All above dependencies are in canonical form (that is, they have only one attribute on the right-hand side), so we have completed step 1 of Algorithm 16.2 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), $B \rightarrow D$. Thus $AB \rightarrow D$ 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.
- Therefore, the minimal cover of E is $\{B \rightarrow D, D \rightarrow A\}$.

Example: Finding a Minimal Cover

- **Example:** Find a **minimal cover** for the following set of **functional dependencies:**

```
Relation R = (A, B, C, D, E, F)

°F1 = { A  → C
        AC → D
        E  → ADF }
```

Result:

```
°Fmin = { A  → C
          A  → D
          E  → A
          E  → F }
```

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

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

- **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
- 4NF
  - based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs
  - Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation;)

3.2 Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - (usually up to 3NF, BCNF or 4NF)
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

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]

- 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.

3.2 First Normal Form

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

- Considered to be part of the definition of relation

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.

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

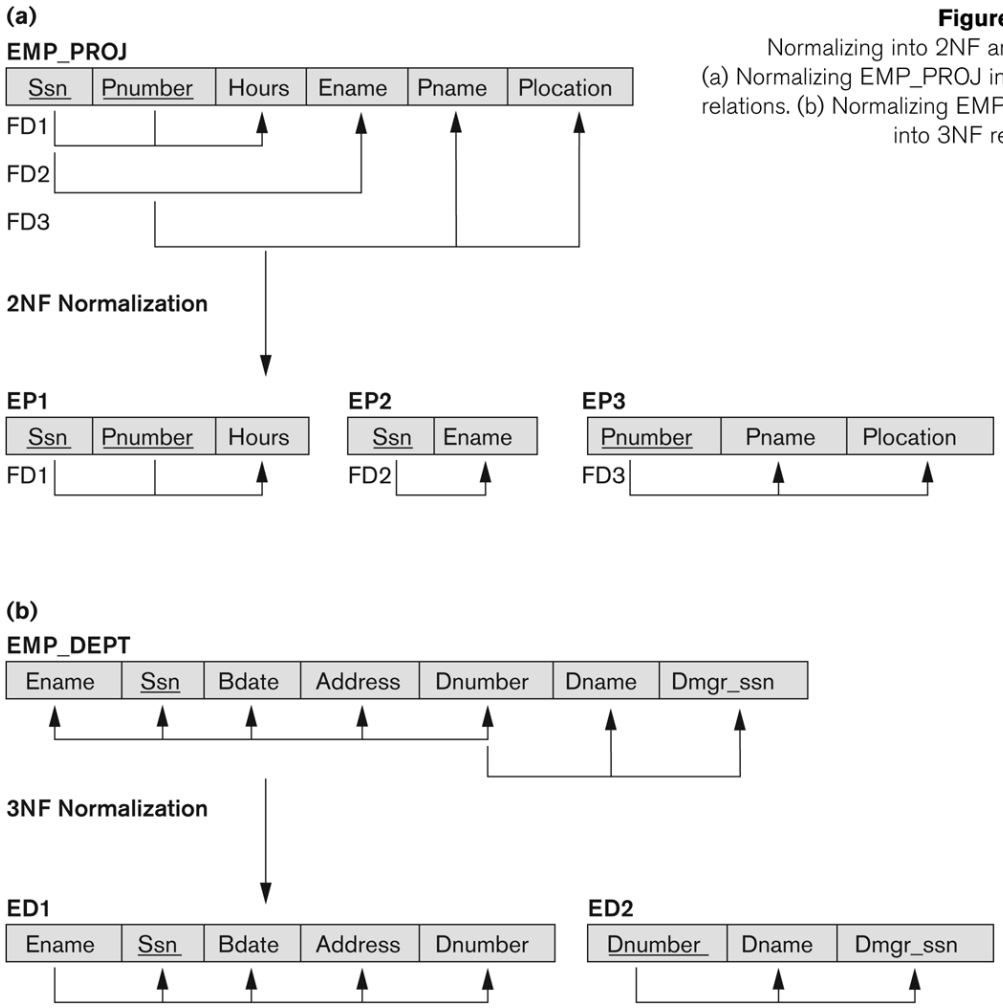


Figure 10.10
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Figure 10.11 Normalization into 2NF and 3NF

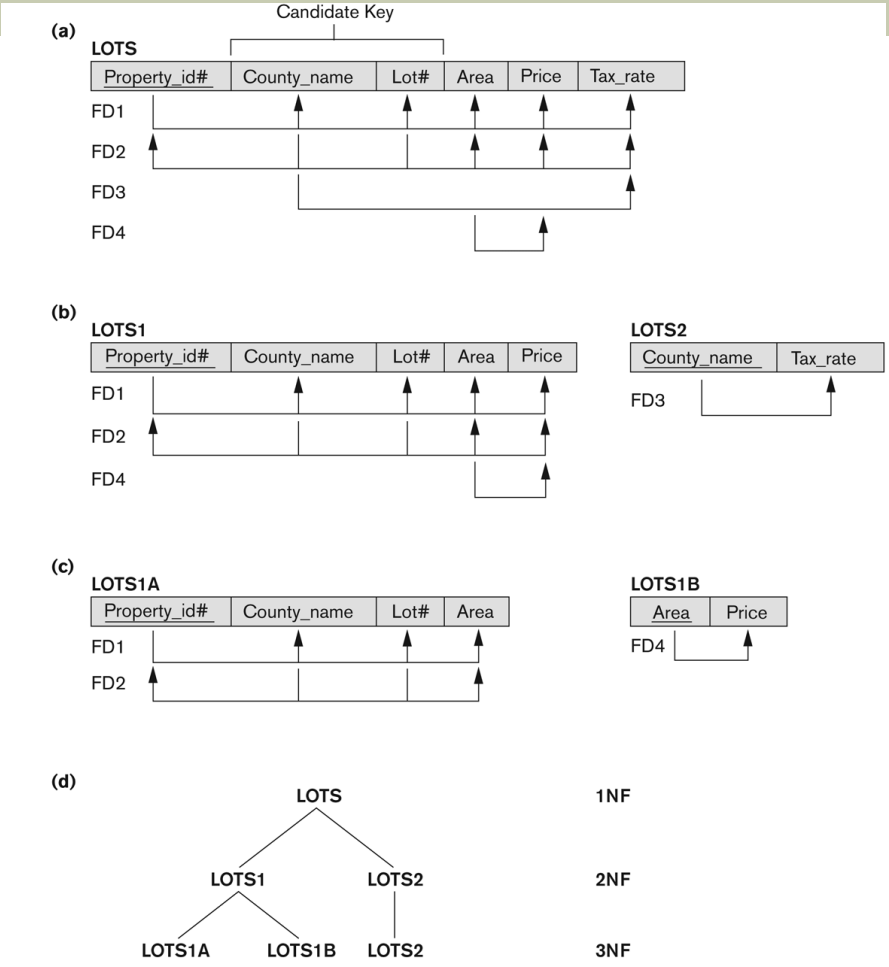


Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

Example

Functional Dependencies – Examples

Consider the schema:

Student(studName, rollNo, sex, dept, hostelName, roomNo)

Since rollNo is a key, rollNo → {studName, sex, dept, hostelName, roomNo}

Suppose that each student is given a hostel room exclusively, then

hostelName, roomNo → rollNo

Suppose boys and girls are accommodated in separate hostels, then

hostelName → sex

Does Sex → hostelName?

FDs are additional constraints that can be specified by designers

Example 1: 2NF

student(rollNo, name, dept, sex, hostelName, roomNo, admitYear)

Assumptions:

Each student is allotted a single-occupancy room.
A room is identified by values of attributes hostelName, roomNo.
Boys and girls are accommodated in separate hostels.

Keys: rollNo, (hostelName, roomNo)

Not in 2NF as hostelName → sex

Decompose:

student(rollNo, name, dept, hostelName, roomNo, admitYear)
hostelDetail(hostelName, sex)

- These are both in 2NF

Example 2: 2NF

book(authorName, title, authorAffiliation, ISBN, publisher, pubYear)

Assumptions: A book has exactly one author.
Author can be uniquely identified by value of attribute authorName
AuthorAffiliation is the organization to which the author is *currently* associated with.
An author is associated with *exactly one* organization at any time.

Keys: (authorName, title), ISBN
Not in 2NF as authorName → authorAffiliation
(authorAffiliation is not fully functionally dependent on the first key)

Decompose:
book(authorName, title, ISBN, publisher, pubYear)
authorInfo(authorName, authorAffiliation) -- both in 2NF

Transitive Dependencies

Transitive dependency:

An FD $X \rightarrow Y$ in a relation schema R for which there is a set of attributes $Z \subseteq R$ such that

$X \rightarrow Z$ and $Z \rightarrow Y$ and Z is not a subset of any key of R

studentDept(rollNo, name, dept, hostelName, roomNo, headDept)

Keys: rollNo, (hostelName, roomNo)

rollNo → dept; dept → headDept hold

So, rollNo → headDept is a transitive dependency

Head of the dept of dept D is stored redundantly in every tuple where D appears.

Relation is in 2NF but redundancy still exists.

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

Normal Forms – 3NF

Relation schema R is in 3NF if it is in 2NF and no non-prime attribute of R is transitively dependent on any key of R

studentDept(rollNo, name, dept, hostelname, roomNo, headDept) is not in 3NF

Decompose: student(rollNo, name, dept, hostelName, roomNo)
deptInfo(dept, headDept)

both in 3NF

Redundancy in data storage - removed

Another definition of 3NF

Relation schema R is in 3NF if for any nontrivial FD $X \rightarrow A$
either (i) X is a superkey or (ii) A is prime.

Suppose some R violates the above definition
$\Rightarrow$ There is an FD $X \rightarrow A$ for which both (i) and (ii) are false
$\Rightarrow$ X is not a superkey and A is non-prime attribute

Two cases arise:

1) X is contained in a key – A is not fully functionally dependent on this key
   - violation of 2NF condition
2) X is not contained in a key
   $K \rightarrow X$, $X \rightarrow A$ is a case of transitive dependency
   (K – any key of R)

Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

4 General Normal Form Definitions (For Multiple Keys) (1)

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on *every* key of R

General Normal Form Definitions (2)

- Definition:
  - **Superkey** of relation schema R - a set of attributes S of R that contains a key of R
  - A relation schema R is in **third normal form (3NF)** if whenever a FD X -> A holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R
- NOTE: Boyce-Codd normal form disallows condition (b) above

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

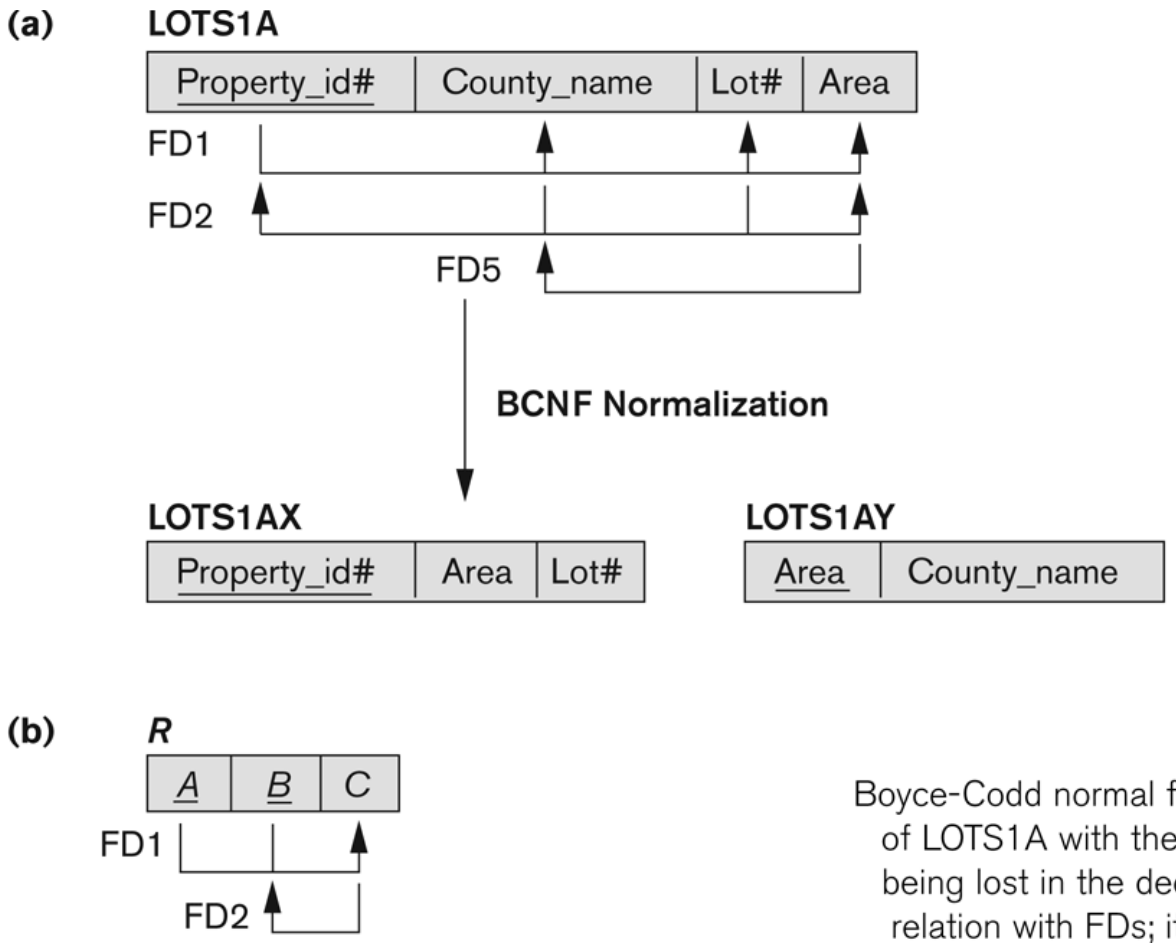


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

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.

Motivating example for BCNF

gradeInfo (rollNo, studName, course, grade)

Suppose the following FDs hold:

1) rollNo, course → grade
2) studName, course → grade
3) rollNo → studName
4) studName → rollNo

Keys:

(rollNo, course)
(studName, course)

(Assumption: No two students have the same name)

For 1, 2 lhs is a key. For 3, 4 rhs is prime; so gradeInfo is in 3NF

But studName is stored redundantly along with every course being done by the student.

Boyce - Codd Normal Form (BCNF)

Relation schema R is in BCNF if for every nontrivial
FD X → A, X is a *superkey* of R.

In gradeInfo, FDs 3, 4 are nontrivial but lhs is not a superkey
So, gradeInfo is not in BCNF

Decompose:

gradeInfo (rollNo, course, grade)
studInfo (rollNo, studName)

Redundancy allowed by 3NF is disallowed by BCNF

BCNF is stricter than 3NF
3NF is stricter than 2NF

Decomposition of Relation Schema

Decomposition of a relation schema

If R doesn't satisfy a particular normal form,
we decompose R into smaller schemas

What's a decomposition?

$R = (A_1, A_2, \ldots, A_n)$

$D = (R_1, R_2, \ldots, R_k)$ st $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \ldots \cup R_k$

(R_i's need not be disjoint)

Replacing R by $R_1, R_2, \ldots, R_k$ is the process of decomposing R

Ex: gradeInfo (rollNo, studName, course, grade)

R_1: gradeInfo (rollNo, course, grade)

R_2: studInfo (rollNo, studName)

Desirable Properties of Decompositions

Not all decomposition of a relational scheme R are useful

We require two properties to be satisfied

(i) Lossless join property
- the information in an instance r of R must be preserved in the instances $r_1, r_2, \ldots, r_k$ where $r_i = \Pi_{R_i}(r)$

(ii) Dependency preserving property
- if a set F of dependencies hold on R it should be possible to enforce F on an instance r by enforcing appropriate dependencies on each r_i

Lossless join property

F – set of FDs that hold on R
R – decomposed into $R_1, R_2, \ldots, R_k$
Decomposition is *lossless* wrt F if
for every relation instance r on R satisfying F,
$r = \Pi_{R_1}(r) * \Pi_{R_2}(r) * \ldots * \Pi_{R_k}(r)$

Lossless joins are also called non-additive joins

Original info is distorted

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

r:

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_1	c_3

Lossy join

r_1:

A	B
a_1	b_1
a_2	b_2
a_3	b_1

r_2:

B	C
b_1	c_1
b_2	c_2
b_1	c_3

$r_1 * r_2$:

A	B	C
a_1	b_1	c_1
a_1	b_1	c_3
a_2	b_2	c_2
a_3	b_1	c_1
a_3	b_1	c_3

Spurious tuples

Dependency Preserving Decompositions

Decomposition $D = (R_1, R_2, \ldots, R_k)$ of schema R *preserves* a set of dependencies F if

$$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F) \cup \ldots \cup \Pi_{R_k}(F))^+ = F^+$$

Here, $\Pi_{R_i}(F) = \{ (X \rightarrow Y) \in F^+ \mid X \subseteq R_i, Y \subseteq R_i \}$
(called projection of F onto R_i)

Informally, any FD that logically follows from F must also logically follow from the union of projections of F onto R_i's
Then, D is called dependency preserving.

An example

Schema $R = (A, B, C)$
FDs $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Decomposition $D = (R_1 = \{A, B\}, R_2 = \{B, C\})$
$\Pi_{R_1}(F) = \{A \rightarrow B, B \rightarrow A\}$
$\Pi_{R_2}(F) = \{B \rightarrow C, C \rightarrow B\}$

$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F))^+ = \{A \rightarrow B, B \rightarrow A,$
$B \rightarrow C, C \rightarrow B,$
$A \rightarrow C, C \rightarrow A\} = F^+$

Hence Dependency preserving

Testing for lossless decomposition property(6/6)

R – given schema. F – given set of FDs

The decomposition of R into R_1, R_2 is lossless wrt F if and only if
either $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ belongs to F^+ or
$R_1 \cap R_2 \rightarrow (R_2 - R_1)$ belongs to F^+

Example:
gradeInfo (rollNo, studName, course, grade)
with FDs = {rollNo, course → grade; studName, course → grade;
rollNo → studName; studName → rollNo}
decomposed into
grades (rollNo, course, grade) and studInfo (rollNo, studName)
is lossless because
rollNo → studName

A property of lossless joins

D_1: (R_1, R_2,…, R_K) lossless decomposition of R wrt F

D_2: (R_{i1}, R_{i2},…, R_{ip}) lossless decomposition of R_i wrt $F_i = \Pi_{R_i}(F)$

Then

D = (R_1, R_2, … , R_{i-1}, R_{i1}, R_{i2}, …, R_{ip}, R_{i+1},…, R_k) is a lossless decomposition of R wrt F

This property is useful in the algorithm for BCNF decomposition

Algorithm for BCNF decomposition

R – given schema. F – given set of FDs

D = {R} // initial decomposition
while there is a relation schema R_i in D that is not in BCNF do
{ let $X \rightarrow A$ be the FD in R_i violating BCNF;
Replace R_i by $R_{i1} = R_i - \{A\}$ and $R_{i2} = X \cup \{A\}$ in D;
}

Decomposition of R_i is lossless as

$$R_{i1} \cap R_{i2} = X,\ R_{i2} - R_{i1} = A \text{ and } X \rightarrow A$$

Result: a lossless decomposition of R into BCNF relations

Dependencies may not be preserved (2/2)

Consider the schema: R (A, B, C)
with the FDs F: AB → C and C → B
Keys: AB, AC – relation in 3NF (all attributes are prime)
– Relation is not in BCNF as C → B and C is not a key

Decomposition given by algorithm: R_1: CB R_2: AC
Not dependency preserving as $\Pi_{R_1}(F)$ = trivial dependencies
$\Pi_{R_2}(F) = \{C \rightarrow B\}$
Union of these does not entail AB → C

All possible decompositions: {AB, BC}, {BA, AC}, {AC, CB}
Only the last one is lossless!

Lossless and dependency-preserving decomposition doesn't exist.

Multi-valued Dependencies (MVDs) and 4NF

studCoursesAndFriends(rollNo,courseNo,frndEmailAddr)

A student enrolls for several courses and has several friends whose email addresses we want to record.

If rows (CS05B007, CS370, shyam@gmail.com) and
(CS05B007, CS376, radha@yahoo.com) appear then
rows (CS05B007, CS376, shyam@gmail.com)
(CS05B007, CS370, radha@yahoo.com) should also appear!

For, otherwise, it implies that having "shyam" as a friend has something to do with doing course CS370!

Causes a huge amount of data redundancy!
Since there are no non-trivial FD's, the scheme is in BCNF

We say that MVD rollNo $\rightarrow\!\!\!\rightarrow$ courseNo holds
(read as rollNo *multi-determines* courseNo)
By symmetry, rollNo $\rightarrow\!\!\!\rightarrow$ frndEmailAddr also holds

More about MVDs

Consider studCourseGrade(rollNo,courseNo,grade)

Note that rollNo →→ courseNo *does not* hold here even though courseNo is a multi-valued attribute of a student entity

If (CS05B007, CS370, A)

(CS05B007, CS376, B) appear in the data then

(CS05B007, CS376, A)

(CS05B007, CS370, B) will not appear !!

Attribute 'grade' depends on (rollNo,courseNo)

MVD's arise when two or more *unrelated* multi-valued attributes of an entity are sought to be represented together in a scheme.

More about MVDs

Consider

studCourseAdvisor(rollNo,courseNo,advisor)

Note that rollNo $\rightarrow\!\!\!\rightarrow$ courseNo *holds* here

If (CS05B007, CS370, Dr Ravi)

(CS05B007, CS376, Dr Ravi) appear in the data then swapping courseNo values gives rise to existing rows only.

But, since rollNo $\rightarrow$ advisor and (rollNo, courseNo) is the key, this gets caught in checking for 2NF itself.

MVD Definition

Consider a scheme R(X, Y, Z),

An MVD X $\rightarrow\rightarrow$ Y holds on R if, for in any instance of R,

the presence of two tuples

(xxx, y1y1y1, z1z1z1) and

(xxx, y2y2y2, z2z2z2)

guarantees the presence of tuples

(xxx, y1y1y1, z2z2z2) and

(xxx, y2y2y2, z1z1z1)

Note that every FD on R is also an MVD!

- the notion of MVD's generalizes the notion of FD's

Alternative definition of MVDs

Consider R(X,Y,Z)

Suppose that $X \twoheadrightarrow Y$ and by symmetry $X \twoheadrightarrow Z$

Then, decomposition D = (XY, XZ) of R should be lossless

That is, for any instance r on R, $r = \Pi_{XY}(r) * \Pi_{XZ}(r)$

MVDs and 4NF

An MVD $X \twoheadrightarrow Y$ on scheme R is called *trivial* if either $Y \subseteq X$ or $R = X \cup Y$. Otherwise, it is called *non-trivial*.

4NF: A relation R is in 4NF if it is in BCNF and for every nontrivial MVD $X \twoheadrightarrow A$, X must be a superkey of R.

studCourseEmail(rollNo,courseNo,frndEmailAddr)

is not in 4NF as

rollNo $\twoheadrightarrow$ courseNo and

rollNo $\twoheadrightarrow$ frndEmailAddr

are both nontrivial and rollNo is not a superkey for the relation

Join Dependencies and 5NF

A join dependency (JD) is generalization of an MVD

A JD $JD(R_1,R_2,\ldots,R_k)$ is said to hold on schema R if

for every instance $r = *(\Pi_{R1}(r), \Pi_{R2}(r), \ldots, \Pi_{Rk}(r))$

Here, $R = R_1 \cup R_2 \cup \ldots \cup R_k$ and Natural join * is a multi-way join.

A JD is difficult to detect in practice. It occurs in rare situations.

A relational scheme is said to be in 5NF wrt to a set of FDs, MVDs and JDs if it is in 4NF and for every non-trivial $JD(R_1,R_2,\ldots,R_k)$, each R_i is a superkey.

Join Dependencies – An Example

Consider the following relation:

studProjSkill(rollNo, skill, project) and the three relations

studSkill(rollNo, skill) // who has what skill

studProj(rollNo, project) // who is interested in what project

skillProj(project, skill) // which project requires what skills

Suppose there is a rule that:

If a student r1 has skill s1, and r1 is interested in project p1and project p1 requires skill s1 then (r1, s1, p1) *must be* in studProjSkill

In other words, studProjSkill = * (studSkill, studProj, skillProj)

Then, we say JD(studSkill, studProj, skillProj) holds

Example - Observations

rollNo	skill
r1	s1
r1	s2

Size <= rs

rollNo	project
r1	p1
r1	p2

Size <= rp

project	skill
p1	s1
p2	s3

Size <= sp

rollNo	project	skill
r1	p1	s1

Size <= rps

There are no MVDs in 3-column table

#students = r, #projects = p, #skills =s

rps >> rp + sp + rs

Huge amount of data redundancy exists