Relational Normalization: Contents

- Motivation
- Functional Dependencies
- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form
- Decomposition Algorithms
- Multivalued Dependencies and Fourth Normal Form
- Join Dependencies and Fifth Normal Form
- Critique of Relational Normalization

1

Relational Database Design

- Choose and apply criteria for grouping attributes in relation schemas
- Necessary to characterize some groupings as better than others
- A good design can be obtained by
  - intuiton, craftmanship of designer
  - design in a more expressive data model (such as E-R) and translate to relational
  - thru normalization theory
    - relational **normal forms**
    - important stage in the development of relational theory
    - normal forms can be defined for other data models (e.g., E-R)

2

Relational Database Design: Rationale

- **Logical level**
  - clarity of semantics of data
  - convenience of query formulation
  - information content
  - applies both to base relations and views (virtual relations)
- **Physical level**
  - storage optimization
  - efficiency of access (query evaluation)
  - simplicity of updates
  - applies only to stored base relations

3

- The grouping of attributes in physical relation schemas has a significant effect on storage space

Example of Relational Schema

Employee

SSN	FName	MInit	LName	BDate	Address	Sex	Salary	SuperSSN	DNo

Department

DNumber	DName	DMgr	MgrStartDate

DeptLocation

DNumber	DLocation

Project

PNumber	PName	PLocation	DNumber

WorksOn

PNumber	PName	Hours

Dependent

SSN	DependentName	Sex	BDate	Relationship

Logical Level: Semantics of Relation Attributes

- Meaning (semantics) is associated with attributes = interpretation of values in relation tuples
- The clearer the semantics of a relation, the better the design of the schema
- A plausible set of rules:
  - each tuple should represent one entity or one relationship instance
  - attributes of different entities and relationships should not be mixed in the same relation
  - only foreign keys should be used to refer to other entities (most probably with referential integrity)

- In terms of entities and relationships
  - DeptLocations represents a multivalued attribute of Department
  - WorksOn represents a M-N relationship between Employee and Project
  - Dependent merges a relationship and a less important entity
- There are many other ways to organize the same data
- No claim that the database represents the "real world"

A not so Good Design and an Improved Normalization

EmpDept

EName	SSN	BDate	Address	DNumber	DName	DMgr

Emp

EName	SSN	BDate	Address	DNumber

Dept

DNumber	DName	DMgr

- Clear semantics but poor design: attributes from distinct real-world entities (Employee, Department) are mixed in EmpDept
- Maybe acceptable for views, but causes problems when done with base relations
- Redundant information causes **update anomalies**

A Similar Example

EmpProj

SSN	PNumber	Hours	EName	PName	PLocation	DNumber

Proj

PNumber	PName	PLocation	DNumber

Emp

SSN	EName

WorksOn

SSN	PNumber	Hours

Insertion Anomalies

EmpDept

EName	SSN	BDate	Address	DNumber	DName	DMgr

- Insert a new employee into EmpDept
  - compatibility of department data to be checked with other tuples of EmpDept
  - insert nulls if no department data is input
- Insert a new department without employee into EmpDept
  - tuple with nulls for employee data (but SSN is key!)
  - remove nulls (complex insertion) when the first employee of th department is inserted

Deletion and Modification Anomaliesan

EmpDept

EName	SSN	BDate	Address	DNumber	DName	DMgr

- Delete from EmpDept the last employee of a department
  → lose information or complex update
- Change manager of a department
  → affects several tuples to avoid inconsistency

Spurious Tuples

- Bad design may result in erroneous results for joins

EmpProj

SSN	PNumber	Hours	EName	PName	PLocation

⇓

EmpLocs

EName	PLocation

EmpProj1

SSN	PNumber	Hours	PName	PLocation

- EmpProj ≠ join of its projections EmpLocs and EmpProj1
  - ◇ common attribute (PLocation) is not a key or a foreign key
  - ◇ joining yields more tuples than in EmpProj ("spurions tuples")
- **Lossless-join property**: guarantees meaningful results for joins

EmpProj

SSN	P#	Hours	EName	PName	PLocation
1234	1	32.5	Smith	ProdX	Bellaire
1234	2	7.5	Smith	ProdY	Sugarland
6668	3	40.0	Narayan	ProdZ	Houston
4534	1	20.0	English	ProdX	Bellaire
4534	2	20.0	English	ProdY	Sugarland

⇓

EmpProj1

SSN	P#	Hours	PName	PLocation
1234	1	32.5	ProdX	Bellaire
1234	2	7.5	ProdY	Sugarland
6668	3	40.0	ProdZ	Houston
4534	1	20.0	ProdX	Bellaire
4534	2	20.0	ProdY	Sugarland

EmpLocs

EName	PLocation
Smith	Bellaire
Smith	Sugarland
Narayan	Houston
English	Bellaire
English	Sugarland

Spurious Tuples in EmProj1 ⋈ EmpLocs

EmpProj1 ⋈ EmpLocs

	SSN	P#	Hours	PName	PLocation	EName
	1234	1	32.5	ProdX	Bellaire	Smith
*	1234	1	32.5	ProdX	Bellaire	English
	1234	2	7.5	ProdY	Sugarland	Smith
*	1234	2	7.5	ProdY	Sugarland	English
	6668	3	40.0	ProdZ	Houston	Narayan
*	4534	1	20.0	ProdX	Bellaire	Smith
	4534	1	20.0	ProdX	Bellaire	English
	4534	2	20.0	ProdY	Sugarland	Smith
*	4534	2	20.0	ProdY	Sugarland	English

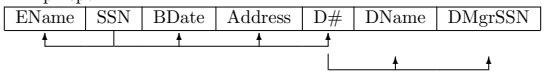
Functional Dependencies (FDs)

- **Definition 1**:
  FD $X \rightarrow Y$ holds in relation $R(A_1, \ldots, A_n)$, with $X, Y \subseteq \{A_1, \ldots, A_n\}$,
  if for every pair of tuples t_1, t_2 such that $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$
- **Definition 2**:
  $X \rightarrow Y$ if there cannot exists different tuples t_1, t_2 such that $t_1[X] = t_2[X]$
- Functional dependencies are constraints, i.e., belong to the schema
  - → cannot be deduced from extension
  - → can only be verified or invalidated on extension (one counterexample is enough to invalidate)

- Definition 1:
  - ◇ all tuples that agree on X also agree on Y
  - ◇ values for X *functionally determine* values for Y
  - ◇ definition applies with $t_1 = t_2$
- **Special case :** if X is a key or superkey then $X \rightarrow Y$ for all $Y \subseteq \{A_1, \ldots, A_n\}$

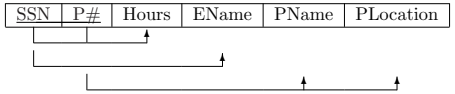
FDs: Example

EmpDept



SSN → EName
SSN → {EName,BDate}
SSN → D#
D# → DMgrSSN

EmpProj



{SSN,P#} → Hours
P# → {PName,PLocation}

FDs: Example, cont.

EmpDept

EName	SSN	BDate	Address	D#	DName	DMgrSSN
Smith	1234	21/07/39	...	1	Research	1234
Narayan	6668	18/01/43	...	1	Research	1234
English	4534	8/05/53	...	2	Account	4534
Wong	9788	30/11/49	...	3	Admin	9788
Zelaya	6677	23/08/60	...	3	Admin	9788

DMgrSSN $\stackrel{?}{\rightarrow}$ D#

EmpProj

SSN	P#	Hours	EName	PName	PLocation
1234	1	32.5	Smith	ProdX	Bellaire
1234	2	7.5	Smith	ProdY	Sugarland
6668	3	40.0	Narayan	ProdZ	Houston
4534	1	20.0	English	ProdX	Bellaire
4534	2	20.0	English	ProdY	Sugarland

PLocation → PName does not hold

Inference of New FDs

- Only most obvious (or most important) dependencies are explicitly specified
- Many other dependencies can be deduced from them
- Closure F^+ of a set F of dependencies:
  - ◇ set of all dependencies in F + those implied by F
  - ◇ dependency $X \rightarrow Y$ is implied by F if
    $X \rightarrow Y$ is valid in all relation instances for which F is valid
  - ◇ $X \rightarrow Y \in$ F iff $F \models X \rightarrow Y$

Armstrong's Inference Rules

A1 (Reflexivity) If $Y \subseteq X$, then $X \to Y$

A2 (Augmentation) If $X \to Y$, then $XZ \to YZ$ (and $XZ \to Y$)

A3 (Transitivity) If $X \to Y$ and $Y \to Z$, then $X \to Z$

A1, A2, A3 form a *sound* and *complete* set of inference rules

Some useful additional inference rules

A4 (Decomposition) If $X \to YZ$, then $X \to Y$ and $X \to Z$

A5 (Union) If $X \to Y$ and $X \to Z$, then $X \to YZ$

A6 (Pseudotransitivity) If $X \to Y$ and $WY \to Z$, then $WX \to Z$

Inference rule: systematic way of constructing functional dependencies in F^+

Example: Deriving FDs

EmpDept

EName	SSN	BDate	Address	D#	DName	DMgrSSN

SSN → EName
SSN → BDate
SSN → Address
SSN → D#
⋮

D# → DName
D# → DMgrSSN
⋮

⇒

SSN → DName
SSN → DMgrSSN
SSN → {EName,DName}
⋮

Example: Deriving FDs, cont.

EmpProj

SSN	P#	Hours	EName	PName	PLocation

P# → P#
P# → PLocation

⇒ P# → {P#,PLocation}

{SSN,P#} → Hours
{SSN,P#} → EName
{SSN,P#} → PName
{SSN,P#} → PLocation
{SSN,P#} → {Hours,EName}
⋮
15 in all

{SSN,P#,Hours} → . . . 7 possibilities
{SSN,P#,EName} → . . . 7 possibilities
{SSN,P#,Hours,EName} → . . . 3 possibilities
⋮

Proof of the Inference Rules

- Viewed as a formal deductive system
  - ◇ a set of given FDs plays the role of axioms
  - ◇ Armstrong's rules: sound and complete inference rules to derive new FDs from the axioms **without** invoking definition of what is an FD
- Structure of formal deductive system is clear and simple
- Armstrong's rules can be proven with the definition of FD
- Derived rules (A4, A5, A6, and others) can be proven from Armstrong's basic rules

Closure of Attribute Sets

- Closure F (X^+) of a set of attributes X under a set F of FDs:
  - ◇ set of attributes A such that $X \rightarrow A$ can be deduced from F by Armstrong's rules
  - ◇ allows to tell at a glance whether FD $X \rightarrow Y$ follows from F
- Algorithm for determining the closure of X under F

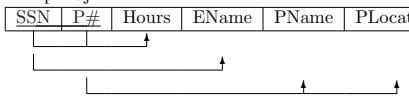
```
X+ := X;
repeat
        oldX+ := X+;
        for each FD Y → Z in F do
                if X+ ⊇ Y then X+ := X+ ∪ Z;
until (X+ = oldX+)
```


- Algorithm starts setting X^+ to all attributes in X: by A1 all these attributes are functionally dependent on X
- Using A3 and A4 we add attributes to X^+, using each FD in F
- We keep going through all dependencies in F (repeat loop) until no more attributes are added to X^+ during a complete cycle (for loop) through the dependencies in F

Closure of Attribute Sets: Example

EmpProj

SSN	P#	Hours	EName	PName	PLocation



$\{SSN\}^+ = \{SSN,EName\}$
$\{P\#\}^+ = \{P\#,PName,PLocation\}$
$\{SSN,P\#\}^+ = \{SSN,P\#,EName,PName,PLocation,Hours\}$

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 $F^+ = G^+$
- F **covers** G if every FD in G can be inferred from F (i.e., if $G^+ \subseteq F^+$)
- F and G are equivalent if F covers G and G covers F
- There is an algorithm for cheking equivalence of sets of FDs

Minimal Sets of FDs

- A set F of FDs is minimal if
  - (1) every FD in F has a single attribute for its RHS
  - (2) no subset of F is equivalent to F
  - (3) no dependency $X \to A$ in F can be replaced by $Y \to A$ with $Y \subset X$ and yield a set of dependencies equivalent to F
- **Minimal cover** of a set F of FDs: minimal set of dependencies F_{min} equivalent to F

> - Every set of FDs has a minimal cover
> - There can be several minimal covers for a set of FDs

Finding Minimal Covers

Algorithm for finding a minimal cover G of F

(1) Set $G := F$

(2) Replace each FD $X \to \{A_1, \ldots A_n\}$ in G by n FDs $X \to A_1, \ldots, X \to A_n$

(3) For each FD $X \to A$ in G
  for each attribute B that is an element of X
    if $(G - \{X \to A\}) \cup \{(X - \{B\}) \to A\}$ is equivalent to G
    then replace $X \to A$ with $(X - \{B\}) \to A$ in G

(4) For each remaining FD $X \to A$ in G
  if $(G - \{X \to A\})$ is equivalent to G
    then remove $X \to A$ from G

Several Minimal Covers: Example

$AB \to C$	$D \to EG$		$AB \to C$	$D \to E$	$CG \to B$
$C \to A$	$BE \to C$	$\Rightarrow$	$C \to A$	$D \to G$	$CG \to D$
$BC \to D$	$CG \to BD$		$BC \to D$	$BE \to C$	$CE \to A$
$ACD \to B$	$CE \to AG$		$ACD \to B$		$CE \to G$

- Two minimal covers

$AB \to C$	$D \to G$
$C \to A$	$BE \to C$
$BC \to D$	$CG \to D$
$CD \to B$	$CE \to G$
$D \to E$	

$AB \to C$	$D \to G$
$C \to A$	$BE \to C$
$BC \to D$	$CG \to B$
$D \to E$	$CE \to G$

First minimal cover

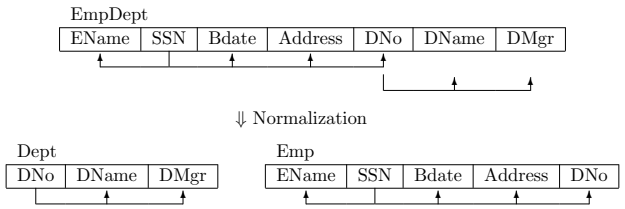
- $CE \rightarrow A$ implied by $C \rightarrow A$
- $CG \rightarrow B$ implied by $CG \rightarrow D$, $C \rightarrow A$, $ACD \rightarrow B$
- $ACD \rightarrow B$ replaced by $CD \rightarrow B$ since $C \rightarrow A$

Second minimal cover

- $CE \rightarrow A$ implied by $C \rightarrow A$
- $CG \rightarrow D$ implied by $CG \rightarrow B$, $BC \rightarrow D$
- $ACD \rightarrow B$ implied by $C \rightarrow A$, $D \rightarrow G$, $CG \rightarrow B$

Normalization

- Process of decomposing unsatisfactory relation schemas into smaller relations without those undesirable aspects (e.g., update anomalies)



- **Normal form** = condition (integrity constraint) to certify that a relation schema is in a particular form

26

Normal Forms

- Several types of normal forms based on different integrity constraints
  - ◇ **Functional Dependencies**
    1st, 2nd, 3rd, Boyce-Codd, Improved 3rd Normal Forms
  - ◇ **Functional and Inclusion Dependencies**
    Inclusion Normal Form
  - ◇ **Functional and Multivalued Dependencies**
    4th Normal Form
  - ◇ **Functional, Multivalued, and Join Dependencies**
    5th Normal Form

27

How Far to Normalize

- Relations not always normalized to the highest possible form, e.g., for performance reasons
- Higher normals forms
  - ⇒ smaller relations
  - ⇒ more joins in queries
  - ⇒ space/time or query/update tradeoff

28

First Normal Form (1NF)

- Relations as defined in the relational model
- Forbids composite attributes, multivalued attributes, nested relations (relations within relations)
- Relational-model limitation to 1NF:
  - ◇ historical reasons (simplify file management)
  - ◇ on retrospect, a mistake

29

Multivalued attributes

Department

DName	DNumber	DMgr	{DLocations}

Department

DName	DNumber	DMgr	{DLocations}
Research	5	333445555	{Bellaire,Sugarland,Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

⇓ 1NF Normalization

Department

DName	DNumber	DLocations	DMgr
Research	5	Bellaire	333445555
Research	5	Sugarland	333445555
Research	5	Houston	333445555
Administration	4	Stafford	987654321
Headquarters	1	Houston	888665555

30

- Redundancy: for each value of DLocation, other attributes has to be repeated
- Normalized relation is less adequate with respect to real-world intuition

Nested Relations

EmpProj

SSN	EName	Projs	
		PNumber	Hours

EmpProj

SSN	EName	Projs	
		PNumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
999888777	Zelaya, Alicia J.	1	20.0
		2	20.0
453453453	English, Joyce A.	30	30.0
		10	10.0

⇓ 1NF Normalization (unnest operation)

EmpProj1

SSN	EName

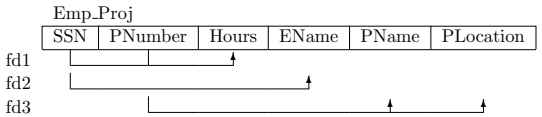
EmpProj2

SSN	PNumber	Hours

31

- Nested relations: value of an attribute of a relation can be a relation
- SSN: primary key of EmpProj
- PNumber: primary key of each nested Projs relation
- Unnest operation transforms the relation into 1NF
- The primary key has to be propagated into the embedded relation

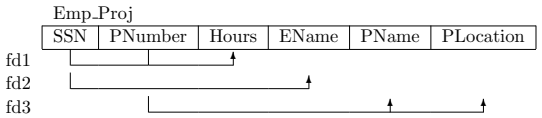
Second Normal Form (2NF)



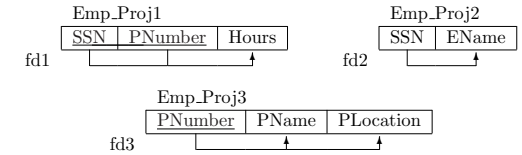
- **Prime attribute:** attribute which is a member of a key
- **Full functional dependency:** an FD $X \rightarrow Z$ where removal of any attribute from X invalidates the dependency
  - ◇ fd1 is a full FD, neither SSN $\rightarrow$ Hours nor PNumber $\rightarrow$ Hours hold
  - ◇ SSN,PNumber $\rightarrow$ EName is not a full FD (i.e., is a **partial dependency**) since SSN $\rightarrow$ EName (fd2) also holds
- Definition: a relation schema R is in 2NF if every nonprime attribute is fully functionally dependent on every key
- A relation where all keys are single attributes is automatically in 2NF

32

Second Normal Form : Example

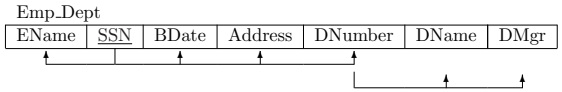


⇓ 2NF Normalization



33

Third Normal Form (3NF)



- $X \rightarrow Z$ is a **transitive functional dependency** if $\exists\, Y$ such that $X \rightarrow Y$ and $Y \rightarrow Z$ (and Z is not a subset of a key)
  - ◇ SSN $\rightarrow$ DMgr is a transitive FD
    (SSN $\rightarrow$ DNumber and DNumber $\rightarrow$ DMgr hold)
  - ◇ SSN $\rightarrow$ EName is a non-transitive FD
    (there is no set of attributes X where SSN $\rightarrow$ X and X $\rightarrow$ EName)

34

Third Normal Form: Definitions

3 definitions for a relation schema R to be in 3NF:

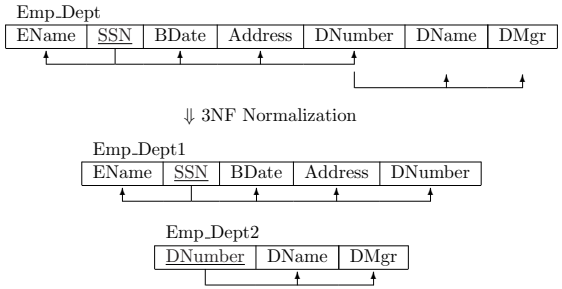
(1) R is in 2NF and no nonprime attribute is transitively dependent on a key

(2) for all $X \rightarrow A \in F^+$
- either X is a superkey,
- or A is a prime attribute

(3) every nonprime attribute is
- fully functionally dependent on every key
- nontransitively dependent on every key

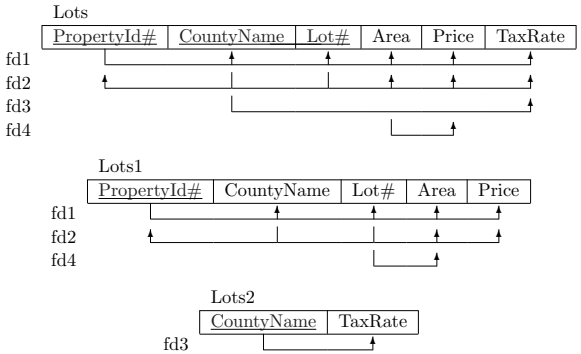
35

Third Normal Form: Example



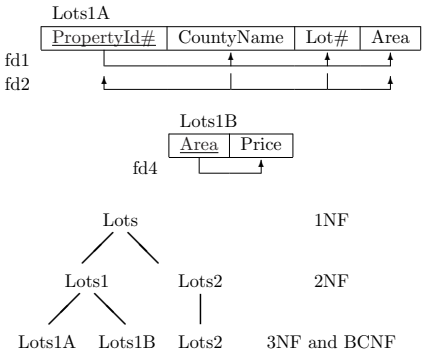
36

Normalization: Example



37

Normalization: Example, cont.



38

Boyce-Codd Normal Form (BCNF)

- A relation schema R is in BCNF if, for all $X \rightarrow A \in F^+$, X is a superkey of R (and $A \notin X$)
- BCNF is an improved 3NF : all dependencies result from keys
- A relation with 2 attributes is automatically in BCNF
- Most 3NF relations are also in BCNF
- Intuition of 3NF/BCNF: **FDs concern the key, the whole key, and nothing but the key**

39

A relation in 3NF but not in BCNF

PatVisit

Patient	Hospital	Doctor

PatVisit

Patient	Hospital	Doctor
Smith	Alachua	Atkinson
Lee	Shands	Smith
Marks	Alachua	Atkinson
Marks	Shands	Shaw
Rao	North Florida	Nefzger

⇓ BCNF Normalization

PatDoctor

Patient	Doctor
Smith	Atkinson
Lee	Smith
Marks	Atkinson
Marks	Shaw
Rao	Nefzger

DoctHosp

Doctor	Hospital
Atkinson	Alachua
Smith	Shands
Shaw	Shands
Nefzger	North Florida

- PatVisit has 2 keys: (Patient,Hospital) and (Patient,Doctor)
- To reach BCNF without losing information an inter-relation constraint is needed, namely (Patient, Hospital) → Doctor

Relational Decomposition

- Normalization decomposes relation schemas with undesirable aspects into smaller relations
- Consider a relation schema $R(A_1, \ldots, A_n)$ and a set of dependencies F
- Goal: produce a decomposition D of R into m relation schemas $D = \{R_1, \ldots, R_m\}$ where each R_i contains a subset of $\{A_1, \ldots, A_n\}$, and
  - ◇ every attribute A_i in R appears in at least one R_i
  - ◇ each relation R_i is at least in BCNF or in 3NF
- Extreme decomposition approach: start with a universal relation schema containing all the DB attributes and a set of FD
- ⇒ **Universal relation assumption** is needed: every attribute is unique, i.e., attributes with the same name in different relations have the same meaning

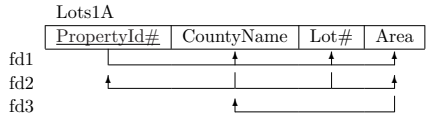
Relational Decomposition, cont.

- Requiring each individual relation to be in a given normal form does not alone guarantee a good design
- BCNF (or 3NF) measure "goodness" for individual relations based on their keys and functional dependencies
- A set of relations must possess additional properties to ensure a good design
  - ◇ dependency preservation
  - ◇ lossless (nonadditive) join
- In traditional approach to relational database design, dependency preservation is required because of the weak support for integrity constraints by DBMSs

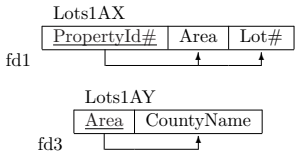
Dependency Preservation

- Consider a relation schema $R(A_1, \ldots, A_n)$, a set of dependencies F, and a decomposition D of R into m relation schemas $D = \{R_1, \ldots, R_m\}$
- Dependency preservation: each FD $X \rightarrow Y$ of F should appear explicitly or be inferrable in one relation schema R_i
- Otherwise, to preserve information, inter-relation FDs are needed (i.e., dependencies that hold on a join of several relations of the decomposition)

Dependency Preservation: Example



⇓ BCNF Normalization with loss of a key



- An inter-relation constraint must express that (Lot#,CountyName) is the key of the join of Lots1AX and Lots1AY

Dependency Preservation: Formalization

- A decomposition D must preserve the dependencies: collection of all dependencies that hold on individual relations R_i must be *equivalent* to F
- Formally
  - ◇ Projection $\prod_F(R_i)$ of F on R_i: set of FDs $X \rightarrow Y$ in F^+ such that $(X \cup Y) \subseteq R_i$ (their left- and right-hand side attributes are in R_i)
  - ◇ A decomposition $D = \{R_1, \ldots, R_m\}$ is dependency-preserving if $(\prod_F(R_1) \cup \ldots \cup \prod_F(R_m))^+ = F^+$
- Dependency preservation enables checking that FDs in F hold by checking them on each relation R_i individually

Dependency Preserving Decomposition Algorithm

Find a dependency preserving decomposition $D = \{R_1, \ldots, R_m\}$ of a relation R w.r.t. a set of FDs F such that each R_i is in 3NF

(1) Find a minimal cover G of F

(2) For each X and FD $X \rightarrow A$ in G

create relation R_i in D with attributes $\{X \cup A_1 \cup \ldots \cup A_k\}$ where $X \rightarrow A_1, \ldots, X \rightarrow A_k$ are the only FDs in G with X as left-hand side (X is the key of R_i)

(3) If attributes in G are not placed in any R_j, create another relation (all-key, without FD) in D for those attributes

- There is no similar algorithm to reach BCNF
- There are several minimal covers in general ⇒ nondeterminism
- Not guaranteed to be lossless (nonadditive)

Lossless (Nonadditive) Join Property

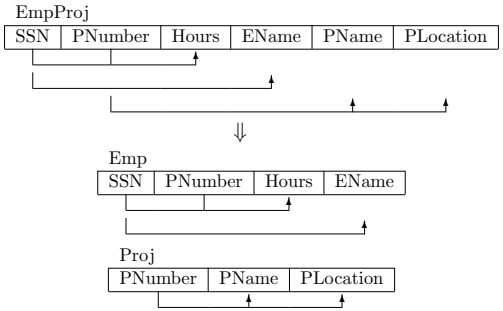
- Ensures that no spurious tuples appear when relations in the decomposition are joined
- Decomposition $D = \{R_1, \ldots, R_m\}$ of R has the lossless-join property w.r.t. a set F of FDs if, for *every* relation instance $r(R)$ whose tuples satisfy all the FDs in F:

  $$\Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_m}(r) = r$$

  (This is the general form of join dependency, see later)
- Ensures that whenever a relation instance $r(R)$ satisfies F, no spurious tuples are generated by joining the decomposed relations $r(R_i)$
- Necessary to generate meaningful results for queries involving joins
- There exists an algorithm for testing whether a decomposition D satisfies the lossless-join property with respect to a set F of FDs

Properties of the Nonadditive Join (1)

Decomposition $D = \{R_1, R_2\}$ of R has the nonadditive-join property w.r.t. F iff either $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ or $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ is in F^+

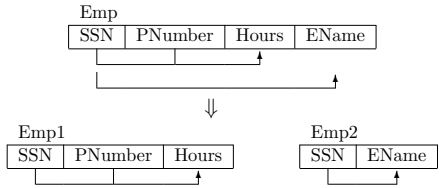


Properties of the Nonadditive Join (2)

If $D = \{R_1, \ldots, R_m\}$ of R has the nonadditive-join property w.r.t. F, and $D_i = \{Q_1, \ldots, Q_k\}$ of R_i has the nonadditive-join property w.r.t. $\prod_{R_i}(F)$, then

$$D = \{R_1, \ldots, R_{i-1}, Q_1, \ldots, Q_k, R_{i+1}, \ldots, R_m\}$$

has the nonadditive join property w.r.t. F



Lossless Join Decomposition Algorithm

Decompose R into a lossless join decomposition $D = \{R_1, \ldots, R_m\}$ w.r.t. F such that each R_i is in BCNF

(1) Set $D := \{R\}$

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

  begin
  find an FD $X \rightarrow Y$ in Q that violates BCNF;
  replace Q in D by two relations $(Q - Y)$ and $(X \cup Y)$
  end

- Does not necessarily preserve the FDs
- The algorithm is very simple: normalizing up to BCNF = break relations according to FDs

Combined Decomposition Algorithm

Produce a lossless join and dependency-preserving decomposition into 3NF

(1) Find a minimal cover G of F

(2) For each X in an FD $X \rightarrow A$ in G

create a relation in D with attributes $\{X \cup A_1 \cup \ldots \cup A_k\}$ where $X \rightarrow A_1, \ldots, X \rightarrow A_k$ are the only FDs in G with X as left-hand side (X is the key of this relation)

(3) If none of the relations in D contains a key of R, create one more relation schema in D that contains attributes that form a key of R

- Step 3 of previous algorithm is not needed because the key will include any unplaced attributes (i.e., attributes not participating in any FD)

Algorithm for Finding a Key

Finding a key F for a relation schema R based on a set F of FDs

(1) Set $K := R$

(2) For each attribute A in K

compute $(K - A)^+$ with respect to F
if $(K - A)^+$ contains all attributes in R then set $K := K - \{A\}$;

- Determines only one key out of the possible candidates keys for R
- Key returned depends on the order in which attributes are removed

Decomposition Example

Stock

Model#	Serial#	Price	Color	Name	Year

Dependencies

{M,S} → {P,C} {M} → {N}
{S} → {Y} {N,Y} → {P}

Minimal Cover

{M,S} → {C} {M} → {N}
{S} → {Y} {N,Y} → {P}

Stock1

Model#	Serial#	Color

Stock2

Model#	Name

Stock3

Serial#	Year

Stock4

Name	Year	Price

Multivalued Dependencies

- Multivalued dependencies are a consequence of requiring 1NF

Course	Teacher	Text
Physics	Green Brown Black	Mechanics Thermodynamics
Math	White	Algebra Geometry

- Semantics: every teacher who teaches a course uses all the texts for that course (independence of Teacher and Text)
- For two or more multivalued *independent* attributes, every value of one of the attributes must be repeated with every value of the other attribute to keep the relation consistent

Multivalued Dependencies (cont.)

Course	Teacher	Text
Physics	Green	Mechanics
Physics	Green	Thermodynamics
Physics	Brown	Mechanics
Physics	Brown	Thermodynamics
Physics	Black	Mechanics
Physics	Black	Thermodynamics
Math	White	Algebra
Math	White	Geometry

- This relation is in BCNF since it is all-key
- A set of attributes X *multidetermines* a set of attributes Y if the value of X determines a set of values for Y (*independently* of any other attributes)
- A multivalued dependency (MVD) is written as $X \rightarrow\rightarrow Y$
- In the example, Course $\rightarrow\rightarrow$ Teacher and Course $\rightarrow\rightarrow$ Text

Intuition of MVDs

- $X \rightarrow\rightarrow Y$ is sometimes paraphrased as "X multidetermines Y" or "a given X-value determines a set of Y-values"
- But this is really not precise enough, this is dangerously close to the paraphrase of a many-to-many relationship or a multivalued attribute
- $Z = R - (X \cup Y)$ is also involved
- If $X \rightarrow\rightarrow Y$ holds, then $X \rightarrow\rightarrow Z$ also holds
- A better, more intuitive notation is $X \rightarrow\rightarrow Y \mid Z$
- $X \rightarrow\rightarrow Y \mid Z$ implies that a value of X determines a **set of values** of Y **independently from** the values of Z

- In the example, Course $\rightarrow\rightarrow$ Teacher | Text: each course is associated with a set of teachers and with a set of texts, and these sets are independent of each other

Multivalued Dependencies: Formal Definition

- Consider a relation schema R, X and Y are subsets of attributes in R, $Z = R - (X \cup Y)$
- $X \rightarrow\rightarrow Y$ holds in R if, whenever tuples t_1 and t_2 exist in an instance $r(R)$ with $t_1[X] = t_2[X]$, then tuples t_3 and t_4 also exist in $r(R)$ such that
  - $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
  - $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$
  - $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

	X	Y	Z
t_1	a	b_1	c_1
t_2	a	b_1	c_2
t_3	a	b_2	c_1
t_4	a	b_2	c_2

- A MVD $X \rightarrow\rightarrow Y$ is **trivial** if either $Y \subseteq X$ or $(X \cup Y) = R$
  - ◇ always holds according to the MVD definition
- FDs are special cases of MVDs: If $X \rightarrow Y$ holds, then $X \rightarrow\rightarrow Z$ also holds

Inference Rules for FDs and MVDs

I1 $Y \subseteq X \Rightarrow X \to Y$

I2 $X \to Y \Rightarrow XZ \to YZ$

I3 $X \to Y$ and $Y \to Z \Rightarrow X \to Z$

I4 $X \to Y \Rightarrow X \to Z$ where $Z = R - (X \cup Y)$

I5 $X \to\to Y$ and $Z \subseteq W \Rightarrow WX \to\to YZ$

I6 $X \to\to Y$ and $Y \to\to Z \Rightarrow X \to\to (Z - Y)$

I7 $X \to Y \Rightarrow X \to\to Z$

I8 $X \to\to Y$ and $W \to Z$ (for $Z \subseteq Y$, $W \cap Y = \emptyset$ and $W \cap Z = \emptyset$) $\Rightarrow X \to Z$

Motivation for 4NF

- A relational schema with non-trivial MVDs is not a good design
- Update anomalies : for a new teacher of Physics, we must insert two tuples

Course	Teacher	Text
Physics	Green	Mechanics
Physics	Green	Thermodynamics
Physics	Brown	Mechanics
Physics	Brown	Thermodynamics
Physics	Black	Mechanics
Physics	Black	Thermodynamics
Math	White	Algebra
Math	White	Geometry

A Sound and Complete Set of Inference Rules for FDs and MVDs

To compute the closure of a set F of functional and multivalued dependencies (F^+)

I1 (Reflexivity for FDs) If $Y \subseteq X$, then $X \to Y$

I2 (Augmentation for FDs) If $X \to Y$, then $XZ \to YZ$

I3 (Transitivity for FDs) If $X \to Y$ and $Y \to Z$, then $X \to Z$

I4 (Complementation for MVDs)
If $X \to Y$, then $X \to Z$ where $Z = R - (X \cup Y)$

I5 (Augmentation for MVDs)
If $X \to\to Y$ and $Z \subseteq W$, then $WX \to\to YZ$

I6 (Transitivity for MVDs)
If $X \to\to Y$ and $Y \to\to Z$, then $X \to\to (Z - Y)$

I7 (Replication)
If $X \to Y$, then $X \to\to Z$

I8 (Coalescence for MVDs)
If $X \to\to Y$ and $W \to Z$, for $Z \subseteq Y$, $W \cap Y = \emptyset$ and $W \cap Z = \emptyset$, then $X \to Z$

- This relation represents two independent 1:N relationships

Course	Teacher
Physics	Green
Physics	Brown
Physics	Black
Math	White

Course	Text
Physics	Mechanics
Physics	Thermodynamics
Math	Algebra
Math	Geometry

Definition of 4NF

- A relation schema R is in 4NF w.r.t. a set of FDs and MVDs F if, for every nontrivial multivalued dependency $X \rightarrow\!\!\!\rightarrow Y$ in F^+, X is a superkey of R
- Since every MVD is an FD, 4NF implies BCNF
- In other words, a relation is in 4NF if it is in BCNF and if every nontrivial MVD is also an FD
- If all dependencies in F are FDs, the definition of 4NF reduces to that of BCNF
- Although many relations in BCNF but not in 4NF are all-key (they have no FD), this is not necessarily so

4NF: Example

Course	Teacher	Text

- Dependencies
  - Course $\rightarrow\!\!\!\rightarrow$ Teacher | Text
  - (Teacher,Text) $\rightarrow$ Course
    (a teacher does not use the same text in more than one course)
- {Teacher,Text} is a key
- Relation is in BCNF but not in 4NF
- Breaking it to produce 4NF relations does not preserve the dependency

Decomposition in 4NF

- Given a MVD $X \rightarrow\!\!\!\rightarrow Y$ that holds in a schema R, the decomposition into $R_1 = (X \cup Y)$ and $R_2 = (R - Y)$ has the nonadditive-join property
- The converse also holds
- Thus, a decomposition $D = \{R_1, R_2\}$ of R has the nonadditive join property with respect to F if and only if either:
  - $(R_1 \cap R_2) \rightarrow\!\!\!\rightarrow (R_1 - R_2)$ holds in F^+, or
  - $(R_1 \cap R_2) \rightarrow\!\!\!\rightarrow (R_2 - R_1)$ holds in F^+
- Actually, if one of these holds, so does the other

Lossless Join Decomposition in 4NF

Algorithm for lossless join decomposition of R into 4NF relations w.r.t. a set of FDs and MVDs

(1) Set $D := \{R\}$

(2) While there is a relation schema Q in D that is not in 4NF do
    begin
    choose one Q in D that is not in 4NF;
    find a nontrivial MVD $X \rightarrow\!\!\!\rightarrow Y$ in Q that violates 4NF;
    replace Q in D by two relations $(Q - Y)$ and $(X \cup Y)$
    end;

- Does not necessarily preserve FDs
  - in the example above the FD (Teacher,Text) $\rightarrow$ Course is lost

A Sufficient Condition for Testing 4NF

- Given a relation schema $R(U)$, a subset C of U is a **cut** if every key of R has a non-empty intersection with C and a nonempty intersection with $U - C$

EmpProj

Emp	Proj	Loc
Smith	P1	FL
Smith	P2	CA
Smith	P3	AZ
Walton	P1	CA
Walton	P2	AZ

- Assume that EmpProj has the keys {Emp,Proj} (an employee works for a project in only one location) and {Emp,Loc} (an employee works in one location for only one project)
- In EmpProj there is only one cut : {Proj,Loc}
- Suppose that another key is added: {Proj,Loc} (given a project and a location, only one employee is attached to them)
- Now, relation EmpProj has no cut

Theorem: If a relation schema is in BCNF and has no cut, then it is in 4NF

Corollary: If a relation schema is in BCNF and has a simple (non composite) key, then it is in 4NF
(a relation with a simple key has no cut)

Embedded Multivalued Dependencies

- FDs are preserved when adding or suppressing an attribute (provided it is not involved in the FD)
- On the contrary, some MVDs are expected to hold after projection but are not explicit as MVD before the projection

Regist

Course	Stud	Preq	Year
CS402	Jones	CS311	1988
CS402	Smith	CS401	1989

FD: {Stud,Preq} → Year

- Course →→ Stud does not hold: (CS402,Jones,CS401,1989) ∉ Regist

Embedded Multivalued Dependencies (cont.)

Regist1

Stud	Preq	Year

Regist2

Course	Stud	Preq

- Course →→ Stud (and Course →→ Preq) hold in Regist2: every student enrolled in a course is required to have taken each prerequisite for the course
- Corresponding constraint in the original relation Regist is an **embedded multivalued dependency**
- It is written Course →→ Stud|Preq, meaning that the dependency holds in the projection $\pi_{\text{Course,Stud,Preq}}(\text{Regist})$
- Regist2 is not in 4NF, and it should be decomposed into relations (Course,Stud) and (Course,Preq)

Join Dependencies

- There are relations where a nonadditive-join decomposition can only be realized with more than two relation schemas

Supply

Supplier	Part	Proj
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Smith	Bolt	ProjY

- If a supplier s supplies part p, a project j uses part p, and the supplier s supplies at least one part to project j, then supplier s also supplies part p to project j
- Relation is "all key", involves no nontrivial FDs or MVDs ⇒ is in 4NF

Join Dependencies, cont.

Sup_Part

Supplier	Part
Smith	Bolt
Smith	Nut
Adamsky	Bolt

Part_Proj

Part	Proj
Bolt	ProjX
Nut	ProjY
Bolt	ProjY

Sup_Proj

Supplier	Proj
Smith	ProjX
Smith	ProjY
Adamsky	ProjY

Join over Part

Supplier	Part	Proj
Smith	Bolt	ProjX
Smith	Bolt	ProjY
Smith	Nut	ProjY
Adamsky	Bolt	ProjX
Adamsky	Bolt	ProjY

spurious

Join over Supplier and Proj

Original relation Supply

- Does not hold if last tuple is removed from Supply

Join Dependencies

- The constraint in the schema is equivalent to say

  if $\langle s_1, p_1, j_2\rangle, \langle s_2, p_1, j_1\rangle, \langle s_1, p_2, j_1\rangle$ appear in Supply
  then $\langle s_1, p_1, j_1\rangle$ also appears in Supply

- Supply satisfies the join dependency JD({Supplier,Part},{Part,Proj},{Supplier,Proj}), i.e.

  Supply = Supply[Supplier,Part] ⋈ Supply[Part,Proj] ⋈ Supply[Supplier,Proj]

- A join dependency $JD(R_1, R_2, \ldots, R_n)$ on a relation R specifies that every instance of R has a nonadditive-join decomposition into $R_1, R_2, \ldots, R_n$
- A MVD is a special case of a JD where $n = 2$
- A $JD(R_1, R_2, \ldots, R_n)$ on R is a trivial JD if some $R_i = R$

Join Dependencies and Update Anomalies

- Join dependencies induce update anomalies

Supply1

Supplier	Part	Proj
Smith	Bolt	ProjX
Smith	Nut	ProjY

Supply2

Supplier	Part	Proj
Smith	Bolt	ProjX
Smith	Nut	ProjY
Walton	Bolt	ProjY
Smith	Bolt	ProjY

- In Supply1, inserting (Walton,Bolt,ProjY) implies insertion of (Smith,Bolt,ProjY) (yet converse is not true)
- Deleting (Walton,Bolt,ProjY) from Supply2 has no side effects
- If (Smith,Bolt,ProjY) is deleted from Supply2, then either the first or the third tuple must be deleted

Fifth Normal Form (5NF)

- A relation schema R is in 5NF w.r.t. a set F or FDs, MVDs, and JDs if for every nontrivial $JD(R_1, R_2, \ldots, R_n)$, each R_i is a superkey

Supplier	Part	Proj

⇓ 5NF normalization

Supplier	Part

Supplier	Proj

Part	Proj

- 5NF is also called PJNF (project-join normal form)
- Since a MVD is a special case of a JD, every relation in 5NF is also in 4NF
- Every relation can be non losslessly decomposed into 5NF relations
- If a relation is in 3NF and all its keys are simple, then it is in 5NF
- Discovering JDs in practice for large databases is difficult

- Many obvious join dependencies are based on keys
- Further decomposition ultimately leads to irreducible relations

Critique of Relational Normalization

(1) Normal forms are easier to understand and appreciate from a richer point of view on data modeling, namely, ER or OO

(2) Practical relevance of normal forms has been overemphasized

1) Normal forms are easier to understand and appreciate from a richer point of view on data modeling, namely, ER or OO

- Particularly striking for the "higher" normal forms (4NF, 5NF)
- MVDs are not stable when relation schemas are modified (leading to embedded dependencies) ⇒ complexity of MVDs is largely a relational problem
- A "complex" normal form like 4NF is better analyzed in ER terms than just with the multi-valued dependency ⇒ there are various interpretations for an MVD
  - ◇ multi-valued attribute
  - ◇ grouping of independent facts within one relation
  - ◇ integrity constraint in a genuine relationship
- ER-based design methodologies start with entities and relationships observed in the real world, and their systematic translation into relations produces 3NF (or higher) most of the time

2) Practical relevance of normal forms has been overemphasized

- Goal of normalization: produce simple relations representing in a natural way a portion of the "real world"
- But why start with a complex description in the first place to have to normalize it afterwards?
- Reason should be traced back to pre-relational technology: space was at a premium and attributes were grouped in large physical records to save space

- Another reason for favoring large relation schemas was to minimize the number of joins in access programs
- Multi-level architectures now permit different schemas at the logical and physical levels
- Normalization was a nice relatively easy piece of relational theory (hard to resist for researchers!)
- Normal forms are properties of relations in isolation; if and when (inter-relation) constraints are seriously taken care of by DBMSs, normalization as a criterion for the quality of a database schema will lose some of its emphasis

- Definition of the relational model was a "revolution" against "bad" practices that were prevailing in database management before the relational model
- Revolution went too far: 1NF is too restrictive for modeling complex data, 1NF was a simple radical idea
- Decomposition algorithms suppose that all functional dependencies have been specified
- Overlooking FDs may produce undesirable designs
- Not easy to solve in practice: it is better to allow grouping attributes on less formal grounds during conceptual modeling
- Algorithms that require a minimal cover depend on which minimal cover is picked (nondeterminism)

Denormalization for Efficiency

- Normalization considers only consistency, performance also matters
- When the normalized database entails too many costly joins, then decide, **at the physical level** to store the joins rather than the projections
- Still, it is more appropriate to start performance tuning with a well-designed database
- Rationale for design in stages: first analysis, then design, then implementation and performance tuning

Relational Database Design

- A complex process
- Made simpler by starting with a more expressive schema in a suitable model (ER, OO)
- Principles of translating an ER schema into a relational schema are simple ...
- But faithfully translating an ER schema into a relational schema is an immense task, if done without loss of information
- In practice, even with sophisticated CASE tools, some information will be lost in the translation for the relational schema to remain manageable
- Traditional relational design neglects essential dependencies (inclusion and join dependencies)
- Normalization theory concerns both ER and relational schemas

Thesis

- Relational DB design has over-emphasized importance of normalization theory