Database Management Systems

Module 26: Relational Database Design/6: Normal Forms

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Week Recap

- Identified the features of good relational design
- Familiarized with the First Normal Form
- Introduced the notion and the theory of functional dependencies
- Discussed issues in "good" design in the context of functional dependencies
- Studied Algorithms for Properties of Functional Dependencies
- Understood the Characterization for and Determination of Lossless Join and Determination of Dependency Preservation

Module Objectives

- To Understand the Normal Forms and their Importance in Relational Design

Module Outline

- Normal Forms

Normal Forms

Normalization or Schema Refinement

- Normalization or Schema Refinement is a technique of organizing the data in the database
- A systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics
  - Insertion Anomaly
  - Update Anomaly
  - Deletion Anomaly
- Most common technique for the Schema Refinement is decomposition.
  - Goal of Normalization: Eliminate Redundancy
- Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations
- Normalization is used for mainly two purpose:
  - Eliminating redundant (useless) data
  - Ensuring data dependencies make sense, that is, data is logically stored

Anomalies

a) **Update Anomaly:** Employee 519 is shown as having different addresses on different records

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

Resolution: Decompose the Schema

a) *Update*: (ID, Address), (ID, Skill)

b) *Insert*: (ID, Name, Hire Date), (ID, Code)

c) *Delete*: (ID, Name, Hire Date), (ID, Code)

b) **Insertion Anomaly**: Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?

c) **Deletion Anomaly**: All information about Dr. Giddens is lost if he temporarily ceases to be assigned to any courses.

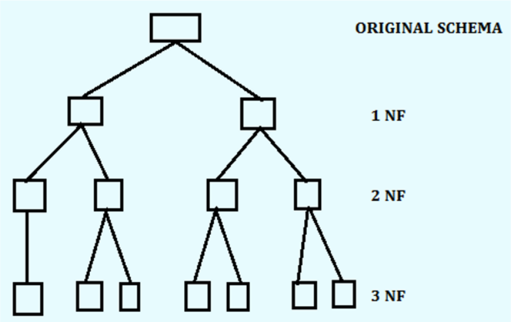
Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

Desirable Properties of Decomposition

- Lossless Join Decomposition Property
  - It should be possible to reconstruct the original table
- Dependency Preserving Property
  - No functional dependency (or other constraints should get violated)



Normalization and Normal Forms

- A normal form specifies a set of conditions that the relational schema must satisfy in terms of its constraints – they offer varied levels of guarantee for the design
- Normalization rules are divided into various normal forms. Most common normal forms are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
- Informally, a relational database relation is often described as "normalized" if it meets third normal form. Most 3NF relations are free of insertion, update, and deletion anomalies

Normalization and Normal Forms

- Additional Normal Forms
  - Elementary Key Normal Form (EKNF)
  - **Boyce-codd Normal Form (BCNF)**
  - **Multivalued Dependencies And Fourth Normal Form (4NF)**
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies and Fifth Normal Form (5NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)

1NF: First Normal Form

- A relation is in First Normal Form if and only if all underlying domains contain atomic values only (doesn't have multivalued attributes (MVA))
- **STUDENT(Sid, Sname, Cname)**

Students		
SID	**Sname**	**Cname**
S1	A	C,C++
S2	B	C++, DB
S3	A	DB
SID : Primary Key		

MVA exists ⇒ Not in 1NF

Students		
SID	**Sname**	**Cname**
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB
SID, Cname : Primary Key		

No MVA ⇒ In 1NF

Source: http://www.edugrabs.com/normal-forms/#fnf

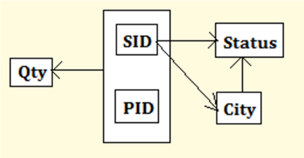
1NF (2): Possible Redundancy

- Example: **Supplier(SID, Status, City, PID, Qty)**

Supplier:

SID	Status	City	PID	Qty
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315

Key : (SID, PID)



Drawbacks:

- Deletion Anomaly: If we delete <S3,40,Rohtak,P1,245>, then we lose the information that S3 lives in Rohtak.
- Insertion Anomaly: We cannot insert a Supplier S5 located in Karnal, until S5 supplies at least one part.
- Update Anomaly: If Supplier S1 moves from Delhi to Kanpur, then it is difficult to update all the tuples having SID as S1 and City as Delhi.

Normalization is a method to reduce redundancy. However, sometimes 1NF increases redundancy.

1NF (3): Possible Redundancy

- **When LHS is not a Superkey** :
  - Let $X \rightarrow Y$ be a non trivial FD over R with X is not a superkey of R, then redundancy exist between X and Y attribute set.
  - Hence in order to identify the redundancy, we need not to look at the actual data, it can be identified by given functional dependency.
  - Example : $X \rightarrow Y$ and X is not a Candidate Key
    $\Rightarrow$ X can duplicate
    $\Rightarrow$ Corresponding Y value would duplicate also.

X	Y
1	3
1	3
2	3
2	3
4	6

- **When LHS is a Superkey** :
  - If $X \rightarrow Y$ is a non trivial FD over R with X is a superkey of R, then redundancy does not exist between X and Y attribute set.
  - Example : $X \rightarrow Y$ and X is a Candidate Key
    $\Rightarrow$ X cannot duplicate
    $\Rightarrow$ Corresponding Y value may or may not duplicate.

X	Y
1	4
2	6
3	4

2NF: Second Normal Form

- Relation $\boldsymbol{R}$ is in Second Normal Form (2NF) only iff :
  - $\boldsymbol{R}$ is in 1NF and
  - $\boldsymbol{R}$ contains no Partial Dependency

Partial Dependency:

Let $\boldsymbol{R}$ be a relational Schema and $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{A}$ be the attribute sets over $\boldsymbol{R}$ where $\boldsymbol{X}$: Any Candidate Key, $\boldsymbol{Y}$: Proper Subset of Candidate Key, and $\boldsymbol{A}$: Non Prime Attribute

If $Y \rightarrow A$ exists in R, then R is not in 2NF.

$(\boldsymbol{Y} \rightarrow \boldsymbol{A})$ is a Partial dependency only if

- $\boldsymbol{Y}$: Proper subset of Candidate Key
- $\boldsymbol{A}$: Non Prime Attribute

A **prime attribute** *of a relation is an attribute that is a part of a candidate key of the relation*

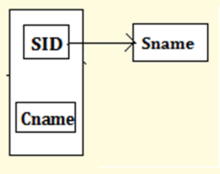
2NF (2)

- **STUDENT(Sid, Sname, Cname)** (already in 1NF)

Students:

SID	Sname	Cname
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB

(SID, Cname): Primary Key



Functional Dependencies:
$\{SID, Cname\} \rightarrow Sname$
$SID \rightarrow Sname$

Partial Dependencies:
$SID \rightarrow Sname$ (as SID is a Proper Subset of Candidate Key $\{SID, Cname\}$)

- Redundancy?
  - Sname
- Anomaly?
  - Yes

Key Normalization

R1:

SID	Sname
S1	A
S2	B
S3	A

{SID}: Primary Key

R2:

SID	Cname
S1	C
S1	C++
S2	C++
S2	DB
S3	DB

{SID,Cname}: Primary Key

The above two relations R1 and R2 are
1.Lossless Join
2.2NF
3.Dependency Preserving

Source: http://www.edugrabs.com/2nf-second-normal-form/

2NF (3): Possible Redundancy

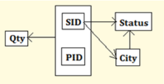
- **Supplier(SID, Status, City, PID, Qty)**

Supplier:

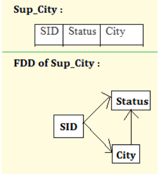
SID	Status	City	PID	Qty
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315

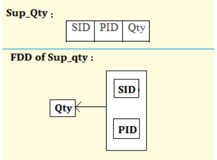
Key : (SID, PID)

Partial Dependencies:
$SID \rightarrow Status$
$SID \rightarrow City$



Post Normalization





Drawbacks:

- **Deletion Anomaly**: If we delete a tuple in *Sup_City*, then we not only loose the information about a supplier, but also loose the status value of a particular city.
- **Insertion Anomaly**: We cannot insert a City and its status until a supplier supplies at least one part.
- **Update Anomaly**: If the status value for a city is changed, then we will face the problem of searching every tuple for that city.

Source: http://www.edugrabs.com/2nf-second-normal-form/

3NF: Third Normal Form

Let $\boldsymbol{R}$ be the relational schema.

- [E. F. Codd,1971] $\boldsymbol{R}$ is in 3NF only if:
  - $\boldsymbol{R}$ should be in 2NF
  - $\boldsymbol{R}$ should not contain transitive dependencies (OR, Every non-prime attribute of R is non-transitively dependent on every key of R)
- [Carlo Zaniolo, 1982] Alternately, $\boldsymbol{R}$ is in 3NF iff for each of its functional dependencies $X \rightarrow A$, at least one of the following conditions holds:
  - X contains A (that is, A is a subset of X, meaning $X \rightarrow A$ is trivial functional dependency), or
  - X is a superkey, or
  - Every element of $A - X$, the set difference between A and X, is a *prime attribute* (i.e., each attribute in $A - X$ is contained in some candidate key)
- [Simple Statement] A relational schema $\boldsymbol{R}$ is in 3NF if for every FD $\boldsymbol{X} \rightarrow \boldsymbol{A}$ associated with R either
  - $\boldsymbol{A} \subseteq \boldsymbol{X}$ (that is, the FD is trivial) or
  - $\boldsymbol{X}$ is a superkey of $\boldsymbol{R}$ or
  - $\boldsymbol{A}$ is part of some candidate key (not just superkey!)
- **A relation in 3NF is naturally in 2NF**

3NF (2): Transitive Dependency

- A **transitive dependency** is a functional dependency which holds by virtue of transitivity. A transitive dependency can occur only in a relation that has three or more attributes.
- Let A, B, and C designate three distinct attributes (or distinct collections of attributes) in the relation. Suppose all three of the following conditions hold:
  - $A \rightarrow B$
  - It is not the case that $B \rightarrow A$
  - $B \rightarrow C$
- Then the functional dependency $A \rightarrow C$ (which follows from 1 and 3 by the axiom of transitivity) is a transitive dependency

3NF (3): Transitive Dependency

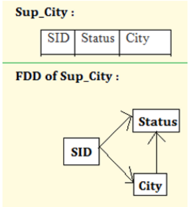
- Example of **transitive dependency**
- The functional dependency $\{Book\} \rightarrow \{Author\ Nationality\}$ applies; that is, if we know the book, we know the author's nationality. Furthermore:
  - $\{Book\} \rightarrow \{Author\}$
  - $\{Author\}$ does not $\rightarrow \{Book\}$
  - $\{Author\} \rightarrow \{Author\ Nationality\}$
- Therefore $\{Book\} \rightarrow \{Author\ Nationality\}$ is a transitive dependency.
- Transitive dependency occurred because a non-key attribute (Author) was determining another non-key attribute (Author Nationality).

Book	Genre	Author	Author Nationality
Twenty Thousand Leagues Under the Sea	Science Fiction	Jules Verne	French
Journey to the Center of the Earth	Science Fiction	Jules Verne	French
Leaves of Grass	Poetry	Walt Whitman	American
Anna Karenina	Literary Fiction	Leo Tolstoy	Russian
A Confession	Religious Autobiography	Leo Tolstoy	Russian

3NF (4): Example

- Example:
  Sup_City(SID, Status, City) (already in 2NF)

Sup_City:		
SID	Status	City
S1	30	Delhi
S2	10	Karnal
S3	40	Rohtak
S4	30	Delhi
SID: Primary Key		



- Redundancy?
  - Status
- Anomaly?
  - Yes

Functional Dependencies:
SID $\rightarrow$ Status,
SID $\rightarrow$ City,
City$\rightarrow$ Status
Transitive Dependency :
SID $\rightarrow$ Status
$\{$As SID $\rightarrow$ City and City $\rightarrow$ Status$\}$

Post Normalization

SC:	
SID	**City**
S1	Delhi
S2	Karnal
S3	Rohtak
S4	Delhi
SID: Primary Key	

CS:	
City	**Status**
Delhi	30
Karnal	10
Rohtak	40
City: Primary Key	

The above two relations SC and CS are
- Lossless Join
- 3NF
- Dependency Preserving

3NF (5): Example

- Relation $\boldsymbol{dept\_advisor}(\boldsymbol{s\_ID}, \boldsymbol{i\_ID}, \boldsymbol{dept\_name})$
- $\boldsymbol{F} = \{\boldsymbol{s\_ID}, \boldsymbol{dept\_name} \rightarrow \boldsymbol{i\_ID}, \boldsymbol{i\_ID} \rightarrow \boldsymbol{dept\_name}\}$
- Two candidate keys: $\boldsymbol{s\_ID}, \boldsymbol{dept\_name}$, and $\boldsymbol{i\_ID}, \boldsymbol{s\_ID}$
- R is in 3NF
    - $\boldsymbol{s\_ID}, \boldsymbol{dept\_name} \rightarrow \boldsymbol{i\_ID}$
        - $\boldsymbol{s\_ID}, \boldsymbol{dept\_name}$ is a superkey
    - $\boldsymbol{i\_ID} \rightarrow \boldsymbol{dept\_name}$
        - $\boldsymbol{dept\_name}$ is contained in a candidate key

A relational schema $\boldsymbol{R}$ is in 3NF if for every FD $\boldsymbol{X} \rightarrow \boldsymbol{A}$ associated with $\boldsymbol{R}$ either

- $\boldsymbol{A} \subseteq \boldsymbol{X}$ (i.e., the FD is trivial) or
- $\boldsymbol{X}$ is a superkey of $\boldsymbol{R}$ or
- $\boldsymbol{A}$ is part of some key (not just superkey!)

3NF (6): Redundancy

- **There is some redundancy in this schema**
- Example of problems due to redundancy in 3NF ($J : s\_ID, L : i\_ID, K : dept\_name$)
  - $R = (J, L, K)$. $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- Repetition of information (for example, the relationship l_1, k_1)
  - $(i\_ID, dept\_name)$
- Need to use null values (for example, to represent the relationship l_2, k_2 where there is no corresponding value for J).
  - $(i\_ID, dept\_name)$ if there is no separate relation mapping instructors to departments

Module Summary

- Studied the Normal Forms and their Importance in Relational Design – how progressive increase of constraints can minimize redundancy in a schema

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 27: Relational Database Design/7: Normal Forms

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Studied the Normal Forms and their Importance in Relational Design – how progressive increase of constraints can minimize redundancy in a schema

Module Objectives

- To Learn the Decomposition Algorithm for a Relation to 3NF
- To Learn the Decomposition Algorithm for a Relation to BCNF

Module Outline

- Decomposition to 3NF
- Decomposition to BCNF

Decomposition to 3NF

3NF Decomposition: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; as seen above)
  - But functional dependencies can be checked on individual relations without computing a join
  - **There is always a lossless-join, dependency-preserving decomposition into 3NF**

3NF Decomposition (2): 3NF Definition

- A relational schema $\boldsymbol{R}$ is in 3NF if for every FD $\boldsymbol{X} \rightarrow \boldsymbol{A}$ associated with R either
  - $\boldsymbol{A} \subseteq \boldsymbol{X}$ (that is, the FD is trivial) or
  - $\boldsymbol{X}$ is a superkey of $\boldsymbol{R}$ or
  - $\boldsymbol{A}$ is part of some candidate key (not just superkey!)
- **A relation in 3NF is naturally in 2NF**

3NF Decomposition (3): Testing for 3NF

- Optimization: Need to check only FDs in F, need not check all FDs in F^+.
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of R
  - This test is rather more expensive, since it involve finding candidate keys
  - **Testing for 3NF has been shown to be NP-hard**
  - **Decomposition into 3NF can be done in polynomial time**

3NF Decomposition (4): Algorithm

- Given: relation R, set F of functional dependencies
- Find: decomposition of R into a set of 3NF relation R_i
- Algorithm:
  a) Eliminate redundant FDs, resulting in a canonical cover F_c of F
  b) Create a relation $R_i = XY$ for each FD $X \rightarrow Y$ in F_c
  c) If the key K of R does not occur in any relation R_i, create one more relation $R_i = K$

3NF Decomposition (5): Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
    if none of the schemas $R_j, 1 \leq j \leq i$ contains $\alpha\beta$
        then begin
            $i := i + 1$;
            $R_i := \alpha\beta$
        end
if none of the schemas $R_j, 1 \leq j \leq i$ contains a candidate key for R
    then begin
            $i := i + 1$;
            $R_i :=$ any candidate key for R;
    end
/* Optionally, remove redundant relations */
repeat
if any schema R_j is contained in another schema R_k
    then /* delete R_j */
        $R_j = R$;
        $i = i - 1$;
return $(R_1, R_2, \cdots, R_i)$

3NF Decomposition (6): Algorithm

- Upon decomposition:
  - Each relation schema R_i is in 3NF
  - Decomposition is
    - Dependency Preserving
    - Lossless Join
- Prove these properties

3NF Decomposition (7): Example

- Relation schema:
  $cust\_banker\_branch = (\underline{customer\_id}, \underline{employee\_id}, branch\_name, type)$
- The functional dependencies for this relation schema are:
  a) $customer\_id, employee\_id \rightarrow branch\_name, type$
  b) $employee\_id \rightarrow branch\_name$
  c) $customer\_id, branch\_name \rightarrow employee\_id$
- We first compute a canonical cover
  - $branch\_name$ is extraneous in the RHS of the 1st dependency
  - No other attribute is extraneous, so we get $F_c =$
    $customer\_id, employee\_id \rightarrow type$
    $employee\_id \rightarrow branch\_name$
    $customer\_id, branch\_name \rightarrow employee\_id$

3NF Decomposition (8): Example

- The **for** loop generates following 3NF schema:

  $(\underline{customer\_id}, \underline{employee\_id}, type)$
  $(\underline{employee\_id}, branch\_name)$
  $(\underline{customer\_id}, \underline{branch\_name}, employee\_id)$

  - Observe that $(customer\_id, employee\_id, type)$ contains a candidate key of the original schema, so no further relation schema needs be added
- At end of for loop, detect and delete schemas, such as $(\underline{employee\_id}, branch\_name)$, which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:

  $(customer\_id, employee\_id, type)$
  $(customer\_id, branch\_name, employee\_id)$

Practice Problem for 3NF Decomposition (1)

- $R = ABCDEFGH$
- FDs $= \{A \rightarrow B, ABCD \rightarrow E, EF \rightarrow GH, ACDF \rightarrow EG\}$

Solution is given in the next slide (hidden from presentation – check after you have solved

Practice Problem for 3NF Decomposition (2)

- $R = CSJDPQV$
- FDs $= \{C \rightarrow CSJDPQV, SD \rightarrow P, JP \rightarrow C, J \rightarrow S\}$

Solution is given in the next slide (hidden from presentation – check after you have solved)

Practice Problem for 3NF Decomposition (3)

Decompose the following schema to 3NF in the following steps

- Compute all keys for R
- Compute a Canonical Cover F_c for F Put the FDs into alphabetical order.
- Using F_c, employ the 3NFdecom algorithm to obtain a lossless and dependency preserving decomposition of relation R into a collection of relations that are in 3NF
- Does your schema allow redundancy?

- $R(ABCDEFGH)$:
  $F = \{A \rightarrow CD, ACF \rightarrow G, AD \rightarrow BEF, BCG \rightarrow D, CF \rightarrow AH, CH \rightarrow G, D \rightarrow B, H \rightarrow DEG\}$
- $R(ABCDE)$:
  $F = \{A \rightarrow B, A \rightarrow C, C \rightarrow D, A \rightarrow E\}$
- $R(ABCDE)$:
  $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$
- $R(ABCD)$:
  $F = \{A \rightarrow D, AB \rightarrow C, AD \rightarrow C, B \rightarrow C, D \rightarrow AB\}$

Decomposition to BCNF

BCNF Decomposition: BCNF Definition

- A relation schema R is in BCNF with respect to a set F of FDs if for all FDs in F^+ of the form

  $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$)
  - α is a superkey for R

BCNF Decomposition (2): Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  a) Compute α^+ (the attribute closure of α), and
  b) Verify that it includes all attributes of R, that is, it is a superkey of R.
- **Simplified test:** To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.
- However, **simplified test using only F is incorrect when testing a relation in a decomposition of R**
  - Consider $R = (A, B, C, D, E)$, with $F = \{A \rightarrow B, BC \rightarrow D\}$
    - Decompose R into $R_1 = (A, B)$ and $R_2 = (A, C, D, E)$
    - Neither of the dependencies in F contain only attributes from (A, C, D, E) so we might be mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

BCNF Decomposition (3): Testing for BCNF Decomposition

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction** of F to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
  - Or use the original set of dependencies F that hold on R, but with the following test:
    - ▷ for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.
    - ▷ If the condition is violated by some $\alpha \to \beta$ in F, the dependency
      $\alpha \to (\alpha^+ - \alpha) \cap R_i$
      can be shown to hold on R_i, and R_i violates BCNF.
    - ▷ We use above dependency to decompose Ri

BCNF Decomposition (4): Testing Dependency Preservation: Using Closure Set of FD (Exp. Algo.): Module 25

Consider the example given below, we will apply both the algorithms to check dependency preservation and will discuss the results.

- **R** (A, B, C, D)
  F $= \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$
- Decomposition: **R1**(A, B) **R2**(B, C) **R3**(C, D)
  - $A \rightarrow B$ is preserved on table R1
  - $B \rightarrow C$ is preserved on table R2
  - $C \rightarrow D$ is preserved on table R3
  - We have to check whether the one remaining FD: **D→A** is preserved or not.

R1	R2	R3
$F_1=\{\mathbf{A} \rightarrow AB, \mathbf{B} \rightarrow BA\}$	$F_2=\{\mathbf{B} \rightarrow BC, \mathbf{C} \rightarrow CB\}$	$F_3=\{\mathbf{C} \rightarrow CD, \mathbf{D} \rightarrow DC\}$

- $F' = F_1 \cup F_2 \cup F_3$.
- Checking for: $\mathbf{D} \rightarrow A$ in F'^{+}
  - ▷ $D \rightarrow C$ (from R3), $C \rightarrow B$ (from R2), $B \rightarrow A$ (from R1) : **D→ A** (By Transitivity)
    Hence all dependencies are preserved.

BCNF Decomposition (4): Testing Dependency Preservation: Using Closure of Attributes (Poly. Algo.): Module 25

- $R(ABCD)$:. $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$
- $Decomp = \{AB, BC, CD\}$
- On projections:

R1	R2	R3
F1 A → B	F2 B → C	F3 C → D

In this algo F1, F2, F3 are not the closure sets, rather the set of dependencies directly applicable on R1, R2, R3 respectively.

- Need to check for: $A \rightarrow B, B \rightarrow C, C \rightarrow D, \boldsymbol{D \rightarrow A}$
- $(D)+/F1 = D$. $(D)+/F2 = D$. $(D)+/F3 = D$. So, $\boldsymbol{D \rightarrow A}$ could not be preserved.
- In the previous method we saw the dependency was preserved. In reality also it is preserved. Therefore the polynomial time algorithm may not work in case of all examples. To prove preservation Algo 2 is sufficient but not necessary whereas Algo 1 is both sufficient as well as necessary.

Note: This difference in result can occur in any example where a functional dependency of one decomposed table uses another functional dependency in its closure which is not applicable on any of the decomposed table because of absence of all attributes in the table.

BCNF Decomposition (4): Algorithm

a) For all dependencies $A \rightarrow B$ in F^+, check if A is a superkey
   - By using attribute closure

b) If not, then
   - Choose a dependency in F^+ that breaks the BCNF rules, say $A \rightarrow B$
   - Create $R1 = AB$
   - Create $R2 = (R - (B - A))$
   - Note that: $R1 \cap R2 = A$ and $A \rightarrow AB$ $(= R1)$, so this is lossless decomposition

c) Repeat for $R1$, and $R2$
   - By defining $F1^+$ to be all dependencies in F that contain only attributes in $R1$
   - Similarly $F2^+$

BCNF Decomposition (5): Algorithm

result := $\{R\}$;
done := *false*;
compute F^+;
while (not done) **do**
    if (there is a schema R_i in *result* that is not in BCNF)
        then begin
            let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
            holds on R_i such that $\alpha \rightarrow \beta$ is not in F^+,
            and $\alpha \cap \beta = \phi$;
            result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
        end
    else *done* := *true*;

Note: each R_i is in BCNF, and decomposition is lossless-join.

BCNF Decomposition (6): Example

- $R = (A, B, C)$
  $$F = \{A \rightarrow B \\ B \rightarrow C\}$$
  $Key = \{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

BCNF Decomposition (7): Example

- *class (course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id)*
- Functional dependencies:
  - *course_id* $\rightarrow$ *title, dept_name, credits*
  - *building, room_number* $\rightarrow$ *capacity*
  - *course_id, sec_id, semester, year* $\rightarrow$ *building, room_number, time_slot_id*
- A candidate key *course_id, sec_id, semester, year*.
- BCNF Decomposition:
  - *course_id* $\rightarrow$ *title, dept_name, credits* holds
    - ▷ but *course_id* is not a superkey
  - We replace *class* by:
    - ▷ *course(course_id, title, dept_name, credits)*
    - ▷ *class-1 (course_id, sec_id, semester, year, building, room_number, capacity, time_slot_id)*

BCNF Decomposition (8): Example

- *course* is in BCNF
  - How do we know this?
- *building, room_number* $\rightarrow$ *capacity* holds on
  class-1(course_id, sec_id, semester, year, building, room_number, capacity, time_slot_id)
  - But {*building, room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom (building, room_number, capacity)*
    - *section (course_id, sec_id, semester, year, building, room_number, time_slot_id)*
- *classroom* and *section* are in BCNF.

BCNF Decomposition (8): Dependency Preservation

- It is not always possible to get a BCNF decomposition that is dependency preserving
- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\quad L \rightarrow K\}$
  Two candidate keys $=$ JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve
  $$JK \rightarrow L$$
  This implies that testing for $JK \rightarrow L$ requires a join

Practice Problem for BCNF Decomposition

Decompose the following schema to BCNF

- $R = ABCDE$. $F = \{A \rightarrow B, BC \rightarrow D\}$
- $R = ABCDEH$. $F = \{A \rightarrow BC, E \rightarrow HA\}$
- $R = CSJDPQV$. $F = \{C \rightarrow CSJDPQV, SD \rightarrow P, JP \rightarrow C, J \rightarrow S\}$
- $R = ABCD$. $F = \{C \rightarrow D, C \rightarrow A, B \rightarrow C\}$

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

S#	**3NF**	**BCNF**
1.	It concentrates on Primary Key	It concentrates on Candidate Key
2.	Redundancy is high as compared to BCNF	0% redundancy
3.	It preserves all the dependencies	It may not preserve the dependencies
4.	A dependency $X \rightarrow Y$ is allowed in 3NF if X is a super key or Y is a part of some key	A dependency $X \rightarrow Y$ is allowed if X is a super key

Module Summary

- Learnt how to decompose a schema into 3NF while preserving dependency and lossless join
- Learnt how to decompose a schema into BCNF with lossless join

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 28: Relational Database Design/8: Case Study

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Learnt how to decompose a schema into 3NF while preserving dependency and lossless join
- Learnt how to decompose a schema into BCNF with lossless join

Module Objectives

- To design the schema for a Library Information System

Module Outline

- Library Information System

Library Information System (LIS)

We are given to design a relational database schema for a Library Information System (LIS) of an Institute

- The specification document of the LIS has already been shared with you
- In this presentation, we include the key points from the Specs; but the actual document must be referred to
- We carry out the following tasks in the module:
  - Identify the Entity Sets with attributes
  - Identify the Relationships
  - Build the initial set of relational schema
  - Refine the set of schema with FDs that hold on them
  - Finalize the design of the schema
- The coding of various queries in SQL, based on these schema are left as exercises

LIS Specs Excerpts

- An institute library has 200000+ books and 10000+ members
- Books are regularly issued by members on loan and returned after a period.
- The library needs an LIS to manage the books, the members and the issue-return process
- Every **book** has
  - title
  - author (in case of multiple authors, only the first author is maintained)
  - publisher
  - year of publication
  - ISBN number (which is unique for the publication), and
  - accession number (which is the unique number of the copy of the book in the library)
  - *There may be multiple copies of the same book in the library*

LIS Specs Excerpts (2)

- There are four categories of **members** of the library:
  - undergraduate students
  - post graduate students
  - research scholars, and
  - faculty members
- Every **student** has
  -
    - name
    - roll number
    - department
    - gender
    - mobile number
    - date of birth, and
    - degree
      - undergrad
      - grad
      - doctoral

LIS Specs Excerpts (3)

- Every **faculty** has
  - name
  - employee id
  - department
  - gender
  - mobile number, and
  - date of joining
- Library also issues a unique **membership number to every member**. Every member has a **maximum quota** for the number of books she / he can issue for the maximum duration allowed to her / him. Currently these are set as:
  - Each undergraduate student can issue up to 2 books for 1 month duration
  - Each postgraduate student can issue up to 4 books for 1 month duration
  - Each research scholar can issue up to 6 books for 3 months duration
  - Each faculty member can issue up to 10 books for six months duration

LIS Specs Excerpts (4)

- The library has the following **rules** for issue:
  - A book may be issued to a member if it is not already issued to someone else (trivial)
  - A book may not be issued to a member if another copy of the same book is already issued to the same member
  - No issue will be done to a member if at the time of issue one or more of the books issued by the member has already exceeded its duration of issue
  - No issue will be allowed also if the quota is exceeded for the member
  - It is assumed that the name of every author or member has two parts
    - first name
    - last name

LIS Specs Excerpts (5): Queries

LIS should support the following operations / query:

- Add / Remove members, categories of members, books.
- Add / Remove / Edit quota for a category of member, duration for a category of member.
- Check if the library has a book given its title (part of title should match). If yes: title, author, publisher, year and ISBN should be listed.
- Check if the library has a book given its author. If yes: title, author, publisher, year and ISBN should be listed.
- Check if a copy of a book (given its ISBN) is available with the library for issue. All accession numbers should be listed with issued or available information.
- Check the available (free) quota of a member.
- Issue a book to a member. This should check for the rules of the library.
- Return a book from a member.
- and so on

LIS Entity Sets: books

- *Every book has title, author (in case of multiple authors, only the first author is maintained), publisher, year of publication, ISBN number (which is unique for the publication), and accession number (which is the unique number of the copy of the book in the library). There may be multiple copies of the same book in the library*
- Entity Set:
  - **books**
- Attributes:
  - title
  - author_name (composite)
  - publisher
  - year
  - ISBN_no
  - accession_no

LIS Entity Sets (2): students

- *Every student has name, roll number, department, gender, mobile number, date of birth, and degree (undergrad, grad, doctoral)*
- Entity Set:
  - **students**
- Attributes:
  - member_no – is unique
  - name (composite)
  - roll_no – is unique
  - department
  - gender
  - mobile_no – may be null
  - dob
  - degree

LIS Entity Sets (3): faculty

- *Every faculty has name, employee id, department, gender, mobile number, and date of joining*
- Entity Set:
  - **faculty**
- Attributes:
  - member_no – is unique
  - name (composite)
  - id – is unique
  - department
  - gender
  - mobile_no – may be null
  - doj

LIS Entity Sets (4): members

- *Library also issues a unique membership number to every member. There are four categories of members of the library: undergraduate students, post graduate students, research scholars, and faculty members*
- Entity Set:
  - **members**
- Attributes:
  - member_no
  - member_type (takes a value in ug, pg, rs or fc)

LIS Entity Sets (5): quota

- *Every member has a maximum quota for the number of books she / he can issue for the maximum duration allowed to her / him. Currently these are set as:*
    - *Each undergraduate student can issue up to 2 books for 1 month duration*
    - *Each postgraduate student can issue up to 4 books for 1 month duration*
    - *Each research scholar can issue up to 6 books for 3 months duration*
    - *Each faculty member can issue up to 10 books for six months duration*
- Entity Set:
    - **quota**
- Attributes:
    - member_type
    - max_books
    - max_duration

LIS Entity Sets (6): staff

- *Though not explicitly stated, library would have staffs to manage the LIS*
- Entity Set:
  - **staff**
- Attributes: (speculated – to ratify from customer)
  - name (composite)
  - id – is unique
  - gender
  - mobile_no
  - doj

LIS Relationships

- *Books are regularly issued by members on loan and returned after a period. The library needs an LIS to manage the books, the members and the issue-return process*
- Relationship
  - **book_issue**
- Involved Entity Sets
  - **students / faculty / members**
    - ▷ member_no
  - **books**
    - ▷ accession_no
- Relationship Attribute
  - doi – date of issue
- Type of relationship
  - Many-to-one from **books**

LIS Relational Schema

- **books**(title, author_fname, author_lname, publisher, year, ISBN_no, accession_no)
- **book_issue**(members, accession_no, doi)
- **members**(member_no, member_type)
- **quota**(member_type, max_books, max_duration)
- **students**(member_no, student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
- **faculty**(member_no, faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
- **staff**(staff_fname, staff_lname, id, gender, mobile_no, doj)

LIS Schema Refinement: books

- **books**(title, author_fname, author_lname, publisher, year, ISBN_no, accession_no)
  - ISBN_no $\rightarrow$ title, author_fname, author_lname, publisher, year
  - accession_no $\rightarrow$ ISBN_no
  - Key: accession_no
- Redundancy of book information across copies
- Good to normalize:
  - **book_catalogue**(title, author_fname, author_lname, publisher, year, ISBN_no)
    - ISBN_no $\rightarrow$ title, author_fname, author_lname, publisher, year
    - Key: ISBN_no
  - **book_copies**(ISBN_no, accession_no)
    - accession_no $\rightarrow$ ISBN_no
    - Key: accession_no
- Both in BCNF. Decomposition is lossless join and dependency preserving

LIS Schema Refinement (2): book_issue

- **book_issue**(member_no, accession_no, doi)
  - member_no, accession_no $\rightarrow$ doi
  - Key: members, accession_no
- In BCNF

LIS Schema Refinement (3): quota

- **quota**(member_type, max_books, max_duration)
  - member_type →max_books, max_duration
  - Key: member_type
- In BCNF

LIS Schema Refinement (4): members

- **members**(member_no, member_type)
  - member_no → member_type
  - Key: member_no
  - Value constraint on member_type
    - ug, pg, or rs: if the member is a student
    - fc: if the member is a faculty
  - In BCNF
  - How to determine the member_type?

LIS Schema Refinement (5): students

- **students**(member_no, student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
  - roll_no → student_fname, student_lname, department, gender, mobile_no, dob, degree
  - member_no → roll_no
  - roll_no → member_no
  - 2 Keys: roll_no | member_no
- In BCNF
- Issues:
  - member_no is needed for issue / return queries. It is unnecessary to have student's details with that.
  - member_no may also come from faculty relation.
  - member_type is needed for issue / return queries. This is implicit in degree – not explicitly given.

LIS Schema Refinement (6): faculty

- **faculty**(member_no, faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
  - id $\rightarrow$ faculty_fname, faculty_lname, department, gender, mobile_no, doj
  - id $\rightarrow$ member_no
  - member_no $\rightarrow$id
  - 2 Keys: id | member_no
- In BCNF
- Issues:
  - member_no is needed for issue / return queries. It is unnecessary to have faculty details with that.
  - member_no may also come from **students** relation.
  - member_type is needed for issue / return queries. This is implicit by the fact that we are in faculty relation.

LIS Schema Refinement (7): Query

- Consider a query:
  - Get the name of the member who has issued the book having accession number = 162715
    - If the member is a student,
      - SELECT student_fname as First_Name, student_lname as Last_Name
      - FROM **students**, **book_issue**
      - WHERE accession_no = 162715 AND book_issue.member_no = students.member_no;
    - If the member is a faculty,
      - SELECT faculty_fname as First_Name, faculty_lname as Last_Name
      - FROM **faculty**, **book_issue**
      - WHERE accession_no = 162715 AND book_issue.member_no = faculty.member_no;
  - **Which query to fire!**

LIS Schema Refinement (8): members

There are 4 categories of members: ug students, grad students, research scholars, and faculty members. This leads to the following specialization relationships:

- Consider the entity set **members** of a library and refine:
  - Attributes:
    - ▷ member_no
    - ▷ member_class – 'student' or 'faculty', used to choose table
    - ▷ member_type – ug,pg, rs, fc, ...
    - ▷ roll_no (if member_class – 'student'. Else null)
    - ▷ id (if member_class – 'faculty'. Else null)
- We can then exploit some hidden relationship:
  - students IS_A members
  - faculty IS_A members
- Type of relationship
  - One-to-one

LIS Schema Refinement (9): Query

- Consider the access query again:
  - Get the name of the member who has issued the book having accession number = 162715

```
SELECT
((SELECT faculty_fname as First_Name, faculty_lname as Last_Name
FROM faculty
WHERE member_class = 'faculty' AND members.id = faculty.id)
UNION
(SELECT student_fname as First_Name, student_lname as Last_Name
FROM students
WHERE member_class = 'student' AND members.roll_no = students.roll_no))
FROM members, book_issue
WHERE accession_no = 162715 AND book_issue.member_no = members.member_no;
```

LIS Schema Refinement (10): members

- **members**(member_no, member_class, member_type, roll_no, id)
  - member_no $\rightarrow$ member_type, member_class, roll_no, id
  - member_type $\rightarrow$ member_class
  - Key: member_no

LIS Schema Refinement (11): students

- **students**(student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
  - roll_no → student_fname, student_lname, department, gender, mobile_no, dob, degree
  - Keys: roll_no
  - *Note:*
    - member_no is no longer used
    - member_type and member_class are set in **members** from degree at the time of creation of a new record.

LIS Schema Refinement (12): faculty

- **faculty**(faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
  - id $\rightarrow$ faculty_fname, faculty_lname, department, gender, mobile_no, doj
  - Keys: id
  - *Note:*
    - member_no is no longer used
    - member_type and member_class are set in **members** at the time of creation of a new record

LIS Schema Refinement (13): Final

- **book_catalogue**(title, author_fname, author_lname, publisher, year, ISBN_no)
- **book_copies**(ISBN_no, accession_no)
- **book_issue**(member_no, accession_no, doi)
- **quota**(member_type, max_books, max_duration)
- **members**(member_no, member_class, member_type, roll_no, id)
- **students**(student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
- **faculty**(faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
- **staff**(staff_fname, staff_lname, id, gender, mobile_no, doj)

Module Summary

- Using the specification for a Library Information System, we have illustrated how a schema can be designed and then refined for finalization
- Coding of various queries based on these schema are left as exercises

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 29: Relational Database Design/9: MVD and 4NF

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Using the specification for a Library Information System, we have illustrated how a schema can be designed and then refined for finalization

Module Objectives

- To understand multi-valued dependencies arising out of attributes that can have multiple values
- To define Fourth Normal Form and learn the decomposition algorithm to 4NF

Module Outline

- Multivalued Dependencies
- Decomposition to 4NF

MVD

Multivalued Dependency

MVD: Multivalued Dependency

- Persons(Man, Phones, Dog_Like)

Person :			Meaning of the tuples
Man(M)	**Phones(P)**	**Dogs_Like(D)**	Man M have phones P, and likes the dogs D.
M1	P1/P2	D1/D2	M1 have phones P1 and P2, and likes the dogs D1 and D2.
M2	P3	D2	M2 have phones P3, and likes the dog D2.
Key : MPD			

There are no non trivial FDs because all attributes are combined forming Candidate Key, that is, MDP. In the above relation, two multivalued dependencies exists:

- Man $\twoheadrightarrow$ Phones
- Man $\twoheadrightarrow$ Dogs_Like

A man's phone are independent of the dogs they like. But after converting the above relation in Single Valued Attribute, each of a man's phones appears with each of the dogs they like in all combinations.

Post 1NF Normalization

Man(M)	Phones(P)	Dogs_Likes(D)
M1	P1	D1
M1	P2	D2
M2	P3	D2
M1	P1	D2
M1	P2	D1

Source: http://www.edugrabs.com/multivalued-dependency-mvd/

- If two or more independent relations are kept in a single relation, then Multivalued Dependency is possible. For example, Let there are two relations :
  - **Student(SID, Sname)** *where* **(SID $\rightarrow$ Sname)**
  - **Course(CID, Cname)** *where* **(CID $\rightarrow$ Cname)**
- There is no relation defined between Student and Course. If we kept them in a single relation named **Student_Course**, then MVD will exists because of m:n Cardinality
- If two or more MVDs exist in a relation, then while converting into SVAs, MVD exists.

Student:

SID	Sname
S1	A
S2	B

Course:

CID	Cname
C1	C
C2	B

SID	Sname	CID	Cname
S1	A	C1	C
S1	A	C2	B
S2	B	C1	C
S2	B	C2	B

2 MVDs exist:
1. SID $\rightarrow\rightarrow$ CID
2. SID $\rightarrow\rightarrow$ Cname

Source: http://www.edugrabs.com/multivalued-dependency-mvd/

MVD (3)

- Suppose we record names of children, and phone numbers for instructors:
  - *inst_child(ID, child_name)*
  - *inst_phone(ID, phone_number)*
- If we were to combine these schema to get
  - *inst_info(ID, child_name, phone_number)*
  - Example data:
    (99999, David, 512-555-1234)
    (99999, David, 512-555-4321)
    (99999, William, 512-555-1234)
    (99999, William, 512-555-4321)
- This relation is in BCNF
  - Why?

MVD: Definition

- Let R be a relation schema and let $\alpha \subseteq$ R and $\beta \subseteq$ R. The **multivalued dependency**

  $\alpha \twoheadrightarrow \beta$

  holds on R if in any legal relation *r(R)*, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2\ [\alpha]$, there exist tuples t_3 and t_4 in r such that:

$t_1[\alpha] = t_2\ [\alpha] = t_3\ [\alpha] = t_4\ [\alpha]$

$t_3[\beta] = t_1\ [\beta]$

$t_3[R - \beta] = t_2[R - \beta]$

$t_4\ [\beta] = t_2[\beta]$

$t_4[R - \beta] = t_1[R - \beta]$

Example: A relation of university courses, the books recommended for the course, and the lecturers who will be teaching the course:

- **course $\twoheadrightarrow$ book**
- **course $\twoheadrightarrow$ lecturer**

Test: **course $\twoheadrightarrow$ book**

Course	Book	Lecturer	Tuples
AHA	Silberschatz	John D	t1
AHA	Nederpelt	William M	t2
AHA	Silberschatz	William M	t3
AHA	Nederpelt	John D	t4
AHA	Silberschatz	Christian G	
AHA	Nederpelt	Christian G	
OSO	Silberschatz	John D	
OSO	Silberschatz	William M	

MVD: Example

- Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.
  Y, Z, W
- We say that Y $\twoheadrightarrow$ Z (Y **multidetermines** Z) if and only if for all possible relations r $(R\ )$
  $< y_1, z_1, w_1 >\in r$ and $< y_1, z_2, w_2 >\in r$
  then
  $< y_1, z_1, w_2 >\in$ r and $< y_1, z_2, w_1 >\in r$
- Note that since the behavior of Z and W are identical it follows that
  $Y \twoheadrightarrow Z$ if $Y \twoheadrightarrow W$

MVD: Example (2)

- In our example:
  $ID \twoheadrightarrow child\_name$
  $ID \twoheadrightarrow phone\_number$
- The above formal definition is supposed to formalize the notion that given a particular value of $Y(ID)$ it has associated with it a set of values of Z ($child\_name$) and a set of values of W ($phone\_number$), and these two sets are in some sense independent of each other.
- Note:
  - If $Y \rightarrow Z$ then $Y \twoheadrightarrow Z$
  - Indeed we have (in above notation) $Z_1 = Z_2$
    The claim follows.

MVD: Use

- We use multivalued dependencies in two ways:
  a) To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
  b) To specify **constraints** on the set of legal relations. We shall thus concern ourselves only with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation r fails to satisfy a given multivalued dependency, we can construct a relations r' that does satisfy the multivalued dependency by adding tuples to r.

MVD: Theory

	Name	Rule
C-	Complementation	If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow (R - (X \cup Y))$.
A-	Augmentation	If $X \twoheadrightarrow Y$ and $W \supseteq Z$, then $WX \twoheadrightarrow YZ$.
T-	Transitivity	If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow (Z - Y)$.
	Replication	If $X \rightarrow Y$, then $X \twoheadrightarrow Y$ but the reverse is not true.
	Coalescence	If $X \twoheadrightarrow Y$ and there is a W such that $W \cap Y$ is empty, $W \rightarrow Z$ and $Y \supseteq Z$, then $X \rightarrow Z$.

- A MVD $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ in **R** is called a trivial MVD is
  - **Y** is a subset of **X** $(\mathbf{X} \supseteq \mathbf{Y})$ or
  - $\mathbf{X} \cup \mathbf{Y} = \mathbf{R}$. Otherwise, it is a non trivial MVD and we have to repeat values redundantly in the tuples.

MVD: Theory (2)

- From the definition of multivalued dependency, we can derive the following rule:
  - If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$

  That is, every functional dependency is also a multivalued dependency
- The **closure** D^+ of D is the set of all functional and multivalued dependencies logically implied by D.
  - We can compute D^+ from D, using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules

Decomposition to 4NF

Fourth Normal Form

- A relation schema R is in **4NF** with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq$ R and $\beta \subseteq$ R, at least one of the following hold:
  - $\alpha \twoheadrightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$ or $\alpha \cup \beta =$ R)
  - α is a superkey for schema R
- If a relation is in 4NF it is in BCNF

Restriction of Multivalued Dependencies

- The restriction of D to R_i is the set D_i consisting of
  - All functional dependencies in D^+ that include only attributes of R_i
  - All multivalued dependencies of the form
    $\alpha \twoheadrightarrow (\beta \cap R_i)$
    where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in D^+

4NF Decomposition Algorithm

a) For all dependencies A $\twoheadrightarrow$ B in D^+, check if A is a superkey
   - By using attribute closure

b) If not, then
   - Choose a dependency in F+ that breaks the 4NF rules, say A $\twoheadrightarrow$ B
   - Create R1 = A B
   - Create R2 = (R – (B – A))
   - Note that: R1 $\cap$ R2 = A and A $\twoheadrightarrow$ AB (= R1), so this is lossless decomposition

c) Repeat for R1, and R2
   - By defining $D1^+$ to be all dependencies in F that contain only attributes in R1
   - Similarly $D2^+$

4NF Decomposition Algorithm

result: $= \{\text{R}\}$;
done $:=$ false;
compute D^+;
Let D_i denote the restriction of D^+ to R_i
while (**not** *done*)
  if (there is a schema R_i in *result* that is not in 4NF) **then**
    begin
      let $\alpha \twoheadrightarrow \beta$ be a nontrivial multivalued dependency that holds
        on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \phi$;
      result $:= (\mathit{result} - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    end
  else *done*$:=$ true;

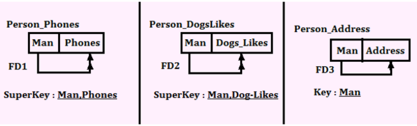
Note: each R_i is in 4NF, and decomposition is lossless-join

Example of 4NF Decomposition

- Example:
  - **Person_Modify(Man(M), Phones(P), Dog_Likes(D), Address(A))**
  - FDs:
    - FD1 : Man $\twoheadrightarrow$ Phones
    - FD2 : Man $\twoheadrightarrow$ Dogs_Like
    - FD3 : Man $\rightarrow$ Address
  - Key = MPD
  - All dependencies violate 4NF

Man(M)	Phones(P)	Dogs_Likes(D)	Address(A)
M1	P1	D1	49-ABC,Bhiwani(HR.)
M1	P2	D2	49-ABC,Bhiwani(HR.)
M2	P3	D2	36-XYZ,Rohtak(HR.)
M1	P1	D2	49-ABC,Bhiwani(HR.)
M1	P2	D1	49-ABC,Bhiwani(HR.)

Post Normalization



In the above relations for both the MVD's – ***'X' is Man***, which is again not the super key, but as ***X* ∪ *Y* = *R*** i.e. (Man & Phones) together make the relation.
So, the above MVD's are trivial and in FD 3, Address is functionally dependent on Man, where **Man** is the key in **Person_Address**, hence all the three relations are in 4NF.

Example of 4NF Decomposition

- R =(A, B, C, G, H, I)
  F = A $\twoheadrightarrow$ B
  B $\twoheadrightarrow$ HI
  CG $\twoheadrightarrow$ H
- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R
- Decomposition
  a) R_1 = (A, B) (R_1 is in 4NF)
  b) R_2 = (A, C, G, H, I) (R_2 is not in 4NF, decompose into R_3 and R_4)
  c) R_3 = (C, G, H) (R_3 is in 4NF)
  d) R_4 = (A, C, G, I) (R_4 is not in 4NF, decompose into R_5 and R_6)
    - $A \twoheadrightarrow B$ and $B \twoheadrightarrow$ HI $\rightarrow$ A $\twoheadrightarrow$ HI, (MVD transitivity), and
    - and hence A $\twoheadrightarrow$ I (MVD restriction to R_4)

  e) R_5= (A, I) (R_5 is in 4NF)
  f) R_6 = (A, C, G) (R_6 is in 4NF)

Module Summary

- Understood multi-valued dependencies to handle attributes that can have multiple values
- Learnt Fourth Normal Form and decomposition to 4NF

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 30: Relational Database Design/10: Design Summary and Temporal Data

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Understood multi-valued dependencies to handle attributes that can have multiple values
- Learnt Fourth Normal Form and decomposition to 4NF

Module Objectives

- To summarize the database design process
- To explore the issues with temporal data

Module Outline

- Database-Design Process
- Modeling Temporal Data

Database Design Process

Design Goals

- Goal for a relational database design is:
  - BCNF / 4NF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.
- Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key

Further Normal Forms

- Further NFs
  - Elementary Key Normal Form (EKNF)
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies And Fifth Normal Form (5 NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)
- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

Overall Database Design Process

- We have assumed schema R is given
  - R could have been generated when converting E-R diagram to a set of tables
  - R could have been a single relation containing all attributes that are of interest (**universal relation**)
  - Normalization breaks R into smaller relations
  - R could have been the result of some ad hoc design of relations, which we then test/convert to normal form

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an employee entity with attributes

    $department\_name$ and $building$,

    and a functional dependency

    $department\_name \rightarrow building$
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare — most relationships are binary

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying prereqs along with course_id, and title requires join of course with prereq
  - **Course(course_id, title,. . .)**
  - **Prerequisite(course_id, prereq)**
- Alternative 1: Use denormalized relation containing attributes of course as well as prereq with all above attributes: **Course(course_id, title, prereq,. . .)**
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: Use a materialized view defined as **Course** $\bowtie$ **Prerequisite**
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:
  Instead of earnings (*company_id, year, amount*), use
  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - ▷ Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company_year (company_id, earnings_2004, earnings_2005, earnings_2006*)
    - ▷ Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - ▷ Is an example of a **crosstab**, where values for one attribute become column names
    - ▷ Used in spreadsheets, and in data analysis tools

LIS Example for 4NF

- Consider a different version of relation **book_catalogue** having the following attributes:
  - *book_title*
  - *book_catalogue*, *author_lname*: A *book_title* may be associated with more than one author.
- **book_title** {*book_title*, *author_fname*, *author_lname*, *edition*}

book_title	author_fname	author_lname	edition
DBMS CONCEPTS	BRINDA	RAY	1
DBMS CONCEPTS	AJAY	SHARMA	1
DBMS CONCEPTS	BRINDA	RAY	2
DBMS CONCEPTS	AJAY	SHARMA	2
JAVA PROGRAMMING	ANITHA	RAJ	5
JAVA PROGRAMMING	RIYA	MISRA	5
JAVA PROGRAMMING	ADITI	PANDEY	5
JAVA PROGRAMMING	ANITHA	RAJ	6
JAVA PROGRAMMING	RIYA	MISRA	6
JAVA PROGRAMMING	ADITI	PANDEY	6

Figure: **book_catalogue**

LIS Example 4NF (2)

book_title	author_fname	author_lname	edition
DBMS CONCEPTS	BRINDA	RAY	1
DBMS CONCEPTS	AJAY	SHARMA	1
DBMS CONCEPTS	BRINDA	RAY	2
DBMS CONCEPTS	AJAY	SHARMA	2
JAVA PROGRAMMING	ANITHA	RAJ	5
JAVA PROGRAMMING	RIYA	MISRA	5
JAVA PROGRAMMING	ADITI	PANDEY	5
JAVA PROGRAMMING	ANITHA	RAJ	6
JAVA PROGRAMMING	RIYA	MISRA	6
JAVA PROGRAMMING	ADITI	PANDEY	6

Figure: **book_catalogue**

- Since the relation has no FDs, it is already in BCNF.
- However, the relation has two nontrivial MVDs $book\_title \twoheadrightarrow \{author\_fname, author\_lname\}$ and $book\_title \twoheadrightarrow edition$. Thus, it is not in 4NF.
- Nontrivial MVDs must be decomposed to convert it into a set of relations in 4NF.

LIS Example 4NF (3)

book_title	author_fname	author_lname
DBMS CONCEPTS	BRINDA	RAY
DBMS CONCEPTS	AJAY	SHARMA
JAVA PROGRAMMING	ANITHA	RAJ
JAVA PROGRAMMING	RIYA	MISRA
JAVA PROGRAMMING	ADITI	PANDEY

Figure: **book_author**

book_title	edition
DBMS CONCEPTS	1
DBMS CONCEPTS	2
JAVA PROGRAMMING	5
JAVA PROGRAMMING	6

Figure: **book_edition**

- We decompose **book_catalogue** into **book_author** and **book_edition** because:
  - **book_author** has trivial MVD $book\_title \twoheadrightarrow \{author\_fname, author\_lname\}$
  - **book_edition** has trivial MVD $book\_title \twoheadrightarrow edition$.

Temporal Databases

Temporal Databases

- Some data may be inherently historical because they include time-dependent / time-varying data, such as:
  - Medical Records
  - Judicial records
  - Share prices
  - Exchange rates
  - Interest rates
  - Company profits
  - etc.
- The desire to model such data means that we need to store not only the respective value but also an associated date or a time period for which the value is valid. Typical queries expressed informally might include:
  - Give me last month's history of the Dollar-Pound Sterling exchange rate.
  - Give me the share prices of the NYSE on October 17, 1996.
- Temporal databases provide a uniform and systematic way of dealing with historical data

Source: https://www.cs.uct.ac.za/mit_notes/database/htmls/chp18.html

Temporal Data

- **Temporal data** have an association time interval during which the data are valid.
- A **snapshot** is the value of the data at a particular point in time
- In practice, database designers may add start and end time attributes to relations
- For example, *course(course_id, course_title)* is replaced by
  course(course_id, course_title, start, end)
  - Constraint: no two tuples can have overlapping valid times and are Hard to enforce efficiently
  - Foreign key references may be to current version of data, or to data at a point in time
    - For example, student transcript should refer to course information at the time the course was taken

Temporal Database Theory

- **Model of Temporal Domain**: Single-dimensional linearly ordered which may be
  - Discrete or dense
  - Bounded or unbounded
  - Single dimensional or multi-dimensional
  - Linear or non-linear
- **Timestamp Model**
- **Temporal ER model** by adding valid time to
  - Attributes: address of an instructor at different points in time
  - Entities: time duration when a student entity exists
  - Relationships: time during which a student attended a course
  - But no accepted standard
- **Temporal Functional Dependency Theory**
- **Temporal Logic**
- **Temporal Query Languge**: TQuel [1987], TSQL2 [1995], SQL/Temporal [1996], SQL/TP [1997]

Modeling Temporal Data: Uni / Bi Temporal

- There are **two different aspects** of time in temporal databases.
    - **Valid Time**: Time period during which a fact is true in real world, provided to the system.
    - **Transaction Time**: Time period during which a fact is stored in the database, based on transaction serialization order and is the timestamp generated automatically by the system.
- Temporal Relation is one where each tuple has associated time; either valid time or transaction time or both associated with it.
    - **Uni-Temporal Relations**: Has one axis of time, either *Valid Time* or *Transaction Time*.
    - **Bi-Temporal Relations**: Has both axis of time – *Valid time* and *Transaction time*. It includes Valid Start Time, Valid End Time, Transaction Start Time, Transaction End Time.

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Example (1)

- **Example.**
  - Let's see an example of a person, John:
    - John was born on April 3, 1992 in Chennai.
    - His father registered his birth after three days on April 6, 1992.
    - John did his entire schooling and college in Chennai.
    - He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
    - He registered his change of address only on Jan 10, 2016.

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Example (2)

- John's Data In Non-Temporal Database

Date	Real world event	Address
April 3, 1992	John is born	
April 6, 1992	John's father registered his birth	Chennai
June 21, 2015	John gets a job	Chennai
Jan 10, 2016	John registers his new address	Mumbai

In a non-temporal database, John's address is entered as Chennai from 1992. When he registers his new address in 2016, the database gets updated and the address field now shows his Mumbai address. The previous Chennai address details will not be available. So, it will be difficult to find out exactly when he was living in Chennai and when he moved to Mumbai.

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

Modeling Temporal Data: Example (3)

- Uni-Temporal Relation (Adding Valid Time To John's Data)

Name	City	Valid From	Valid Till
John	Chennai	April 3, 1992	June 20, 2015
John	Mumbai	June 21, 2015	∞

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

- The valid time temporal database contents look like this:
  `Name, City, Valid From, Valid Till`
- Johns father registers his birth on 6th April 1992, a new database entry is made:
  `Person(John, Chennai, 3-Apr-1992,` ∞`).`
- On January 10, 2016 John reports his new address in Mumbai:
  `Person(John, Mumbai, 21-June-2015,` ∞`).`
  - The original entry is updated:
    `Person(John, Chennai, 3-Apr-1992, 20-June-2015).`

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Example (4)

- Bi-Temporal Relation (John's Data Using Both Valid And Transaction Time)

Name	City	Valid From	Valid Till	Entered	Superseded
John	Chennai	April 3, 1992	June 20, 2015	April 6, 1992	Jan 10, 2016
John	Mumbai	June 21, 2015	∞	Jan 10, 2016	∞

- The database contents look like this:
  `Name, City, Valid From, Valid Till, Entered, Superseded`
- Johns father registers his birth on 6th April 1992:
  `Person(John, Chennai, 3-Apr-1992,` ∞`, 6-Apr-1992,` ∞`).`
- On January 10, 2016 John reports his new address in Mumbai:
  `Person(John, Mumbai, 21-June-2015,` ∞`, 10-Jan-2016,` ∞`).`
  - The original entry is updated as:
    `Person(John, Chennai, 3-Apr-1992, 20-June-2015, 6-Apr-1992 , 10-Jan-2016).`

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Summary

- **Advantages**
  - The main advantages of this bi-temporal relations is that it provides historical and roll back information.
    - Historical Information – Valid Time.
    - Rollback Information – Transaction Time.
  - *For example*, you can get the result for a query on John's history, like: Where did John live in the year 2001?. The result for this query can be got with the valid time entry. The transaction time entry is important to get the rollback information.
- **Disadvantages**
  - More storage
  - Complex query processing
  - Complex maintenance including backup and recovery

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Module Summary

- Discussed aspects of the database design process
- Studied the issues with temporal data