Normal forms and MVD

- **Normal Forms**
- Decomposition to 3NF
- Decomposition to BCNF

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

PPD

1. **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

1. *Update*: (ID, Address), (ID, Skill)
2. *Insert*: (ID, Name, Hire Date), (ID, Code)
3. *Delete*: (ID, Name, Hire Date), (ID, Code)

2. **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	?

3. **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

Desirable Properties of Decomposition

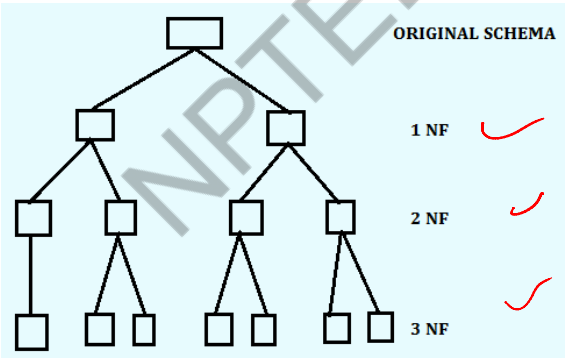
PPD

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)



BCNF
MVD

Normalization and Normal Forms

PPD

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 (1 NF)
- Second Normal Form (2 NF)
- Third Normal Form (3 NF)

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 (4 NF)**
- Essential Tuple Normal Form (ETNF)
- Join Dependencies And Fifth Normal Form (5 NF)
- Sixth Normal Form (6NF)
- Domain/Key Normal Form (DKNF)

not in syllabus

First Normal Form (1 NF)

PPD

A relation is in first Normal Form if and only if all underlying domains contain atomic values only

In other words, a relation doesn't have multivalued attributes (MVA)

Example:

STUDENT(Sid, Sname, Cname)

Students

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

SID : Primary Key

Students

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

SID : Primary Key

First Normal Form (1 NF): Possible Redundancy

PPD

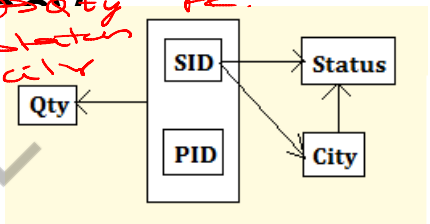
Example:

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 the tuple <S3,40,Rohtak,P1,245>, then we loose the information about S3 that S3 lives in Rohtak.
- **Insertion Anomaly –** We cannot insert a Supplier S5 located in Karnal, until S5 supplies at least one part.
- **Updation Anomaly –** If Supplier S1 moves from Delhi to Kanpur, then it is difficult to update all the tuples containing (S1, Delhi) as SID and City respectively.

Normal Forms are the methods of reducing redundancy. However, Sometimes 1 NF increases redundancy. It does not make any efforts in order to decrease redundancy.

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

First Normal Form (1 NF): Possible Redundancy

When LHS is not a Superkey :

Let X → Y is 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 →Y and X is not a Candidate Key
⇒ X can duplicate
⇒ 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 → 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 →Y and X is a Candidate Key
⇒ X cannot duplicate
⇒ corresponding Y value may or may not duplicate.

X	Y
1	4
2	6
3	4

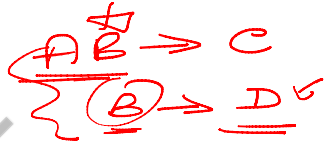
Source: http://www.edugrabs.com/normal-forms/#fnf

Second Normal Form (2 NF)

Relation ***R*** is in Second Normal Form (2NF) only iff :

R should be in 1NF and

R should not contain any *Partial Dependency*



Partial Dependency:

Let ***R*** be a relational Schema and ***X,Y,A*** be the attribute sets over ***R*** where
X: Any Candidate Key, ***Y:*** Proper Subset of Candidate Key, and ***A:*** Non Key Attribute

If Y → A exists in R, then R is not in 2 NF.

(Y → A) is a Partial dependency only if
- ***Y:*** Proper subset of Candidate Key
- ***A:*** Non Prime Attribute

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

Second Normal Form (2 NF)

PPD

Example:

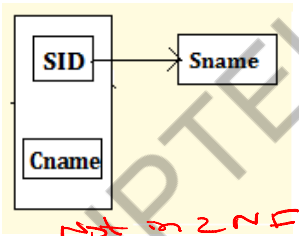
STUDENT(Sid, Sname, Cname) (already in 1NF)

Students:

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

(SID, Cname): Primary Key

- **Redundancy?**
  - *Sname*
- **Anomaly?**
  - *Yes*



Not in 2NF

Functional Dependencies:
{SID,Cname} → Sname SID → Sname

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

Post Normalization

R1:

SID	Sname
S1	A
S2	B
S3	A

{SID}: Primary Key

SID → Sname

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

Second Normal Form (2 NF): Possible Redundancy

Example:

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

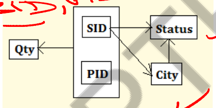
Supplier:

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)

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

Partial Dependencies:
SID → Status SID → City

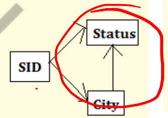


Post Normalization

Sup_City :

SID	Status	City

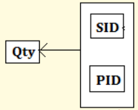
FDD of Sup_City :



Sup_Qty :

SID	PID	Qty

FDD of Sup_qty :



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.
- **Updation Anomaly –** If the status value for a city is changed, then we will face the problem of searching every tuple for that city.

Second Normal Form (2 NF): Possible Redundancy

PPD

In the **Sup_City** relation :

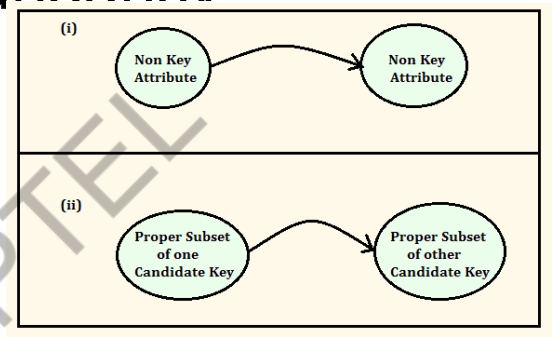
City → Status

- Non Key Attribute → Non Key Attribute

In the **STUDENT** relation:

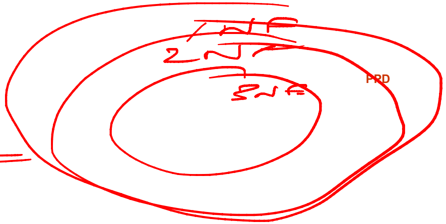
SID → Cname

- Proper Subset of 1 CK → Proper Subset of other CK



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

Third Normal Form (3 NF)



Let ***R*** be the relational schema

- [E. F. Codd ,1971] ***R*** is in 3NF only if:
  - ***R*** should be in 2NF
  - ***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, ***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 **R** is in 3NF if for every FD **X** → **A** associated with R either
  - **A** ⊆ **X** (i.e., the FD is trivial) or
  - **X** is a superkey of **R** or
  - **A** is part of some key (not just superkey!)

Source: http://www.edugrabs.com/3nf-third-normal-form/

Third Normal Form (3 NF)

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 → B
- It is not the case that B → A
- B → C

Then the functional dependency A → C (which follows from 1 and 3 by the axiom of transitivity) is a transitive dependency

Third Normal Form (3 NF)

Example of **transitive dependency**

The functional dependency {Book} → {Author Nationality} applies; that is, if we know the book, we know the author's nationality. Furthermore:

- {Book} → {Author}
- {Author} does not → {Book}
- {Author} → {Author Nationality}

Therefore {Book} → {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

Third Normal Form (3 NF)

PPD

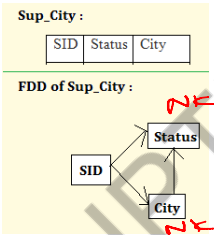
Example:

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



Functional Dependencies:
SID → Status, SID → City
City → Status

Transitive Dependency :
SID → Status {As SID → City and City → Status}

- **Redundancy?**
  - *Status*
- **Anomaly?**
  - *Yes*

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
1. Lossless Join
2. 3NF
3. Dependency Preserving

Third Normal Form (3 NF)

Example

Relation *dept_advisor*:

dept_advisor* (*s_ID, i_ID, dept_name)

***F* = {*s_ID, dept_name* → *i_ID*, *i_ID* → *dept_name*}**

Two candidate keys: ***s_ID, dept_name***, and ***i_ID, s_ID***

R is in 3NF

s_ID, dept_name* → *i_ID

s_ID , dept_name is a superkey

i_ID* → *dept_name

dept_name is contained in a candidate key

A relational schema **R** is in 3NF if for every FD **X** → **A** associated with R either

- **A** ⊆ **X** (i.e., the FD is trivial) or
- **X** is a superkey of **R** or
- **A** is part of some key (not just superkey!)

Redundancy in 3NF

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 (e.g., the relationship l_1, k_1)

(i_ID, dept_name)

need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for *J*).

(i_ID, dept_namel) if there is no separate relation mapping instructors to departments

Third Normal Form (3 NF): Possible Redundancy

A table is automatically in 3NF if one of the following hold :

(i) If relation consists of two attributes.

(ii) If 2NF table consists of only one non key attributes

If X → A is a dependency, then the table is in the 3NF, if one of the following conditions exists:

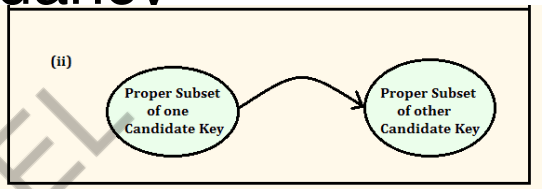
If X is a superkey

If X is a part of superkey

If X → A is a dependency, then the table is said to be NOT in 3NF if the following:

If X is a proper subset of some key (partial dependency)

If X is not a proper subset of key (non key)



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

PPD

- Normal Forms
- **Decomposition to 3NF**
- Decomposition to BCNF

DECOMPOSITION TO 3NF

Third Normal Form: 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**

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
- Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

3NF Decomposition Algorithm

Given: relation R, set F of functional dependencies

Find: decomposition of R into a set of 3NF relation Ri

Algorithm:

1. Eliminate redundant FDs, resulting in a canonical cover Fc of F
2. Create a relation Ri = XY for each FD X → Y in Fc
3. If the key K of R does not occur in any relation Ri, create one more relation Ri=K

3NF Decomposition Algorithm (Formal)

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, $1 \le j \le i$ contains $\alpha\ \beta$
then begin
$i := i + 1$;
$R_i := \alpha\ \beta$
end
if none of the schemas R_j, $1 \le j \le i$ 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 $(R_1, R_2, ..., R_i)$

3NF Decomposition Algorithm

Above algorithm ensures:

- Each relation schema R_i is in 3NF
- Decomposition is d
  - Dependency preserving and
  - Lossless-join

Example of 3NF Decomposition

Relation schema:

cust_banker_branch = (*customer_id, employee_id, branch_name, type*)

The functional dependencies for this relation schema are:

1. *customer_id, employee_id* → *branch_name, type*
2. *employee_id* → *branch_name*
3. *customer_id, branch_name* → *employee_id*

We first compute a canonical cover

branch_name is extraneous in the r.h.s. of the 1st dependency

No other attribute is extraneous, so we get F_C =

customer_id, employee_id → *type* *employee_id* → *branch_name*
customer_id, branch_name → *employee_id*

Example of 3NF Decomposition

The **for** loop generates following 3NF schema: (*customer_id, employee_id, type*) (*employee_id, branch_name*) (*customer_id, 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 (*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→B, ABCD→E, EF→GH, ACDF→EG}

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

Solution: Practice Problem for 3NF Decomposition: 1

R = ABCDEFGH

FD = F = {A→B, ABCD→E, EF→GH, ACDF→EG}

Compute Canonical Cover (Fc)

- Make RHS a single attribute: {A→B, ABCD→E, EF→G, EF→H, ACDF→E, ACDF→G}
- Minimize LHS: ACD→E instead of ABCD→E
- Eliminate redundant FDs
  - Can ACDF→G be removed?
  - Can ACDF→E be removed?
- Fc = {A→B, ACD→E, EF→G, EF→H}

Superkey = ACDF

3NF Decomposition = {AB, ACDE, EFG, EFH} U {ACDF}

This is a hidden slide giving solution. Check only after you have solved)

Practice Problem for 3NF Decomposition: 2

R = CSJDPQV

FDs = {C→CSJDPQV, SD→P, JP→C,J→S}

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

Solution: Practice Problem for 3NF Decomposition: 2

PPD

R = CSJDPQV

FDs = F = {C→CSJDPQV, SD→P, JP→C,J→S}

Compute Canonical Cover (Fc)

- Minimal cover: {C→J, C→D, C→Q, C→V, JP→C, J→S, SD→P} (Combine LHS)
- Fc = {C→JDQV, JP→C, J→S, SD→P}

Superkey = C

3NF Decomposition = {CJDQV, JPC, JS, SDP}

This is a hidden slide giving solution. Check only after you have solved)

- Normal Forms
- Decomposition to 3NF
- **Decomposition to BCNF**

DECOMPOSITION TO BCNF

Testing for BCNF

To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF

1. compute α^+ (the attribute closure of α), and
2. 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.

Testing Decomposition for BCNF

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 $\alpha \rightarrow \beta$ in F, the dependency

    $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$

    can be shown to hold on R_i, and R_i violates BCNF.
  - We use above dependency to decompose R_i

BCNF Decomposition Algorithm

1. For all dependencies A → B in F+, check if A is a superkey
   By using attribute closure
2. If not, then
   Choose a dependency in F+ that breaks the BCNF rules, say A → B
   Create R1 = A B
   Create R2 = (R – B)
   Note that: R1 ∩ R2 = A and A → AB (= R1), so this is lossless decomposition
3. Repeat for R1, and R2
   By defining F1+ to be all dependencies in F that contain only attributes in R1
   Similarly F2+

BCNF Decomposition Algorithm

result := {*R*}; *done* := false; /* initially */
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 $\alpha \rightarrow R_i$ is not in F^+,
        and $\alpha \cap \beta = \varnothing$;
      result := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β);
    end
  else *done* := **true**;

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

Example of BCNF Decomposition

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

R is not in BCNF ($B \rightarrow C$ but B is not superkey)

Decomposition

$R_1 = (B, C)$

$R_2 = (A, B)$

Example of BCNF Decomposition

class (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

Functional dependencies:

course_id→ *title*, *dept_name*, *credits*

building, *room_number*→*capacity*

course_id, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*

A candidate key {*course_id*, *sec_id*, *semester*, *year*}.

BCNF Decomposition:

course_id→ *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 (Cont.)

course is in BCNF

How do we know this?

building, *room_number*→*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 and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

$R = (J, K, L)$
$F = \{JK \rightarrow L$
$L \rightarrow K\}$

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

R = ABCDE. F = {A → B, BC → D}

R = ABCDE. F = {A → B, BC → D}

R = ABCDEH. F = {A → BC, E → HA}

R = CSJDPQV. F = {C →CSJDPQV, SD →P, JP →C,J → S}

R = ABCD. F = {C→D, C→A, B→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 may preserve all the dependencies	It may not preserve the dependencies.
4.	A dependency X → Y is allowed in 3NF if X is a super key or Y is a part of some key.	A dependency X → Y is allowed if X is a super key

Example problems worked out and uploaded in teams.

- **Multivalued Dependencies**
- Decomposition to 4NF
- Database-Design Process
- Modeling Temporal Data

MULTIVALUED DEPENDENCY

Multivalued Dependency

PPD

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 i.e. MDP. In the above relation, two multivalued dependencies exists –

- **Man →→ Phones**
- **Man →→ Dogs_Like**

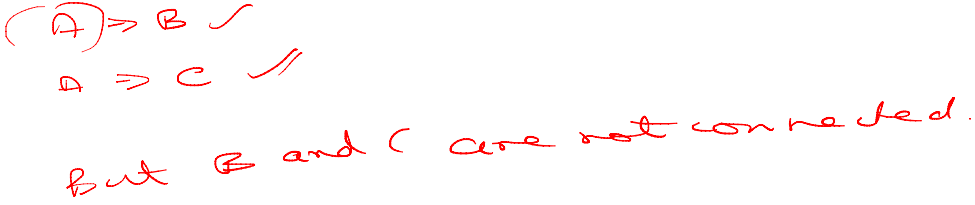
A man's phone are independent of the dogs they like. But after converting the above relation in Single Valued Attribute, each of a man's phones appears with each of the dogs they like in all combinations.

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/

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.



Multivalued Dependency

PPD

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 → Sname)***

Course(CID, Cname) *where* ***(CID → 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 →→ CID
2. SID →→ Cname

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

Multivalued Dependencies

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 schemas 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?

Multivalued Dependencies (MVDs)

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:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta] \quad t_4[R - \beta] = t_1[R - \beta]$$

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

- **course →→ book**
- **course →→ lecturer**

Test: **course →→ 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	

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 $Y \rightarrow\rightarrow Z$ (Y **multidetermines** Z) if and only if for all possible relations r (R)

$$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$

then

$$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

Note that since the behavior of Z and W are identical it follows that

$Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

Example (Cont.)

In our example:

$ID \rightarrow\rightarrow child\_name$
$ID \rightarrow\rightarrow phone\_number$

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 $Y \rightarrow\rightarrow Z$

Indeed we have (in above notation) $Z_1 = Z_2$
The claim follows.

Use of Multivalued Dependencies

We use multivalued dependencies in two ways:

1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
2. 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*.

Theory of MVDs

PPD

	Name		Rule
C-	**Complementation**	:	If X →→ Y, then X →→ {R − (X∪Y)} .
A-	**Augmentation**	:	If X →→ Y and W ⊇ Z, then WX →→ YZ.
T-	**Transitivity**	:	If X →→ Y and Y →→ Z, then X →→ (Z − Y).
	Replication	:	If X → Y, then X →→ Y but the reverse is not true.
	Coalescence	:	If X →→ Y and there is a W such that W ∩ Y is empty, W → Z, and Y ⊇ Z, then X → Z.

A MVD ***X*** →→ ***Y*** in ***R*** is called a trivial MVD is

Y is a subset of ***X*** (***X*** ⊇ ***Y***) or

X ∪ ***Y*** = ***R***. Otherwise, it is a non trivial MVD and we have to repeat values redundantly in the tuples.

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

Theory of MVDs

From the definition of multivalued dependency, we can derive the following rule:

- If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$

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

That's all