INFS1200/7900
Introduction to Information Systems

Functional Dependencies and
Normal Forms

Hassan Khosravi

Learning Outcomes

Description	Tag
Provide examples of modification, insertion, and deletion anomalies.	Functional-dependencies
Explain the term functional dependency.	
Given a set F of functional dependencies and a functional dependency fd, determine whether fd can be inferred from F using Armstrong's Axioms.	
Given a set of functional dependencies that hold over a table, determine the keys.	
Given a set of functional dependencies that hold over a table, determine the superkeys.	
Given a set F of functional dependencies and X of attributes of a relation, compute the closure of X, which is X+	
Given a relation R, be able to correctly decompose it into two or more relations	Decomposition
Justify why lossless join decompositions are preferred decompositions.	
Explain the benefits of Normalization.	Normalization
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 1NF.	
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 2NF.	
Explain what dependency preservation means.	
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 3NF.	
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in BCNF.	
Describe the process and benefits of Denormalization.	
Reason with the logical foundation of the relational data model and understand the fundamental principles of correct relational database design.	

UQ Active Learn

- UQ Active Learn contains muliple applications includding **UQpoll** and **UQwordcloud**

URL: apps.elearning.uq.edu.au
ID: 60140

Disclosure: This application allows me to track authors of the responses.

Design Guidelines

Functional Dependencies

Normalization

Informal Design Guidelines

- Informal measures of relational database schema quality and design guidelines
  - Semantics of the attributes do we need that attributes
  - Reducing the redundant values in tuples
  - Reducing the null values in tuples
  - Disallowing spurious tuples

fake, false

Guideline 1

- Design each relation so that it is easy to explain its meaning
- Do not combine attributes from multiple entity types and relationship types into a single relation

Redundant Values in Tuples

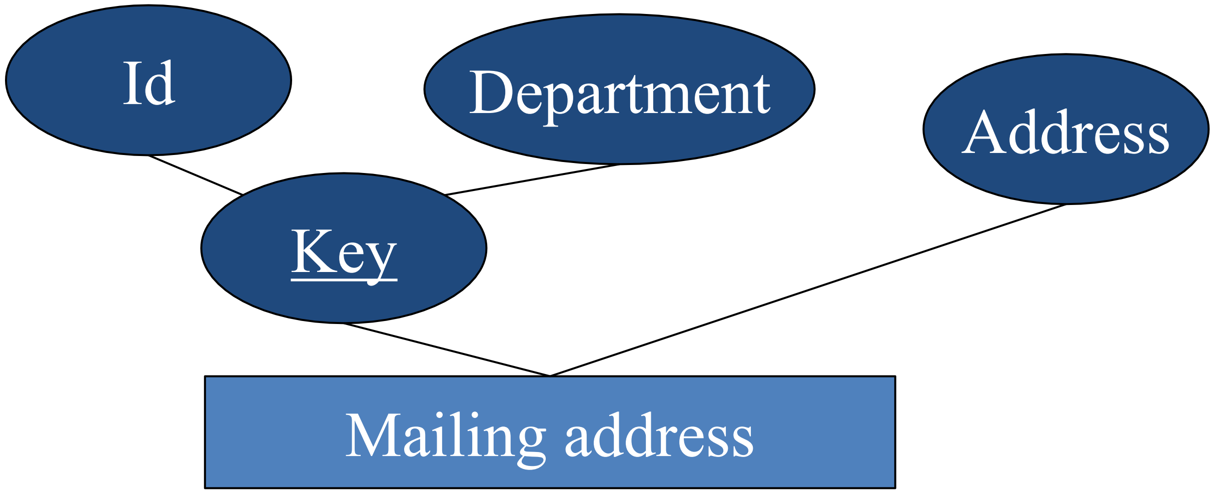
- One design goal is to minimize the storage space that base relations occupy price is not a very big issue

- The way the grouping of attributes into relation schemas is done, has a significant effect on storage space

- In addition, an incorrect grouping may cause *update anomalies* which may result in inconsistent data or even loss of data

A Motivating Example

- Imagine that we have created the following entity for mailing addresses at a University



Meets all the criteria that we have discussed for an entity so far

What Would an Instance Look Like?

<u>Id</u>	<u>Department</u>	Address
100	Computer Science	78-101 Main Mall
104	Computer Science	78-101 Main Mall
104	Math	69-201 Main Mall
105	Physics	50-205 Main Mall

- **Modification anomaly:** data inconsistency that results from data redundancy.
  - Example: Updating the mailing address of a department.
- **Deletion anomaly:** loss of certain attributes because of the deletion of other attributes.
  - Example: Deleting 105 would lead to deletion of the address of Physics.
- **Insertion anomaly:** Lack of ability to insert some attributes without the presence of other attributes. (key cannot be NULL - entity constraint)
  - Example: Storing the mailing address of a department that has no faculty members.

Guideline 2

- Design the base relation schema so that no insertion, deletion, or modification anomalies occur in the relations
- If any do occur, ensure that all applications that access the database update the relations in such a way as to not compromise the integrity of the database

Guideline 3

- As far as possible, avoid placing attributes in a base relation whose values may be null
- If nulls are unavoidable, make sure that they apply in exceptional cases only and that they do not apply to a majority of tuples in the relation

Decomposing a Relation

- A decomposition of R replaces R by two or more relations such that:
  - Each new relation contains a subset of the attributes of R (and no attributes not appearing in R)
  - Every attribute of R appears in at least one new relation.
- Example:

Mailing Address (Id, Department, Address)

Academics(Id, Department)
Mailing Address(Department, Address)

How should we decompose tables to remove anomalies?

The join

- Definition: $R_1 \bowtie R_2$ is the (natural) join of the two relations
  - each tuple of R_1 is concatenated with every tuple in R_2 having the same values on the common attributes.

R_1

A	B
1	2
4	5
7	2

R_2

B	C
2	3
5	6
2	8

order doesn't matter

$R_1 \bowtie R_2 \longrightarrow$

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

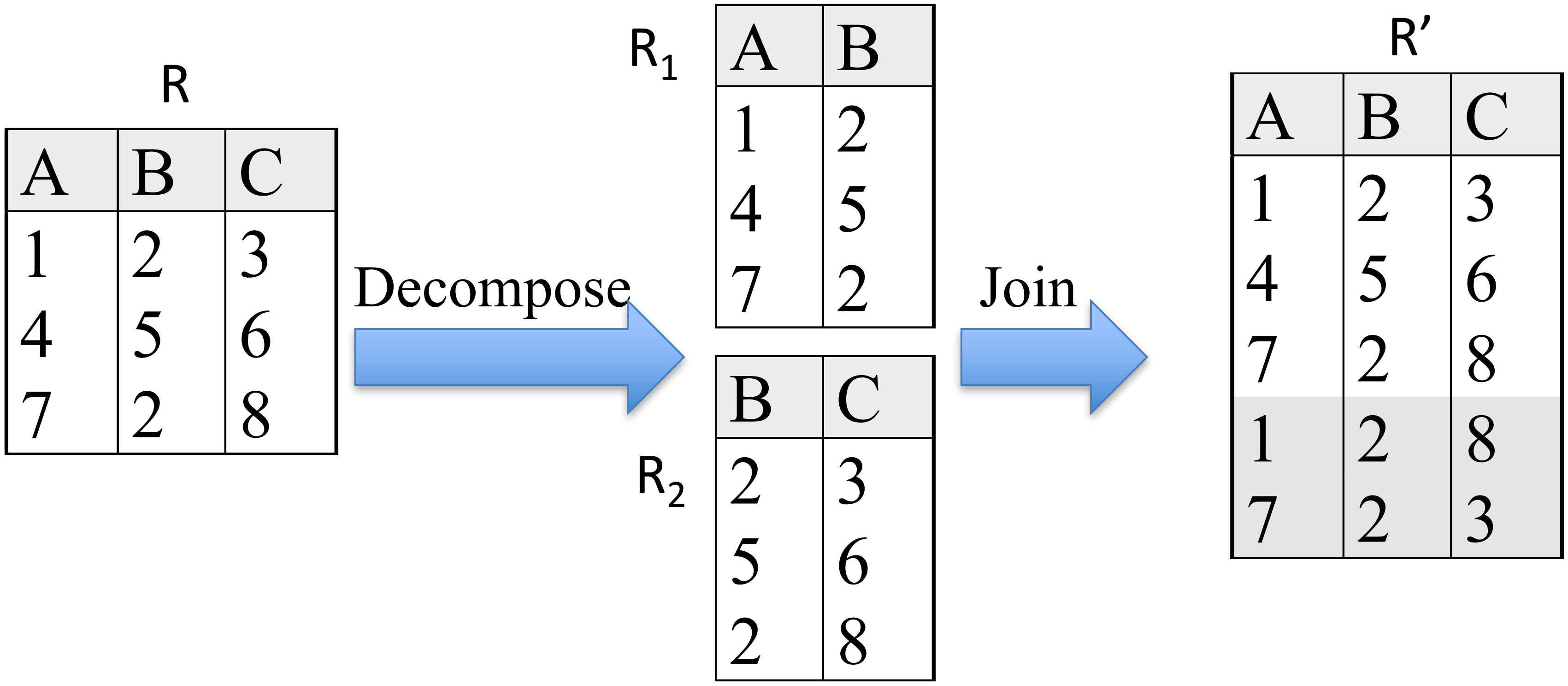
Lossless-Join Decompositions: Definition

Decomposition of R into R_1 and R_2 is a lossless-join w.r.t. a set of FDs F if, for every instance r that satisfies F:

$$R = R_1 \bowtie R_2$$

- Informally: If we break a relation, R, into bits, when we put the bits back together, we should get exactly R back again.

Example Lossy-Join Decomposition



R

A	B	C
1	2	3
4	5	6
7	2	8

R_1

A	B
1	2
4	5
7	2

R_2

B	C
2	3
5	6
2	8

R'

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

- The word loss in lossless refers to loss of information, not to loss of tuples. Maybe a better term here is "addition of spurious information".
- Last two rows are not in the original. Would this have happened if B determined C?

Guideline 4

- Design the relation schemas so that they can be (relationally) *joined* with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated

Design Guidelines

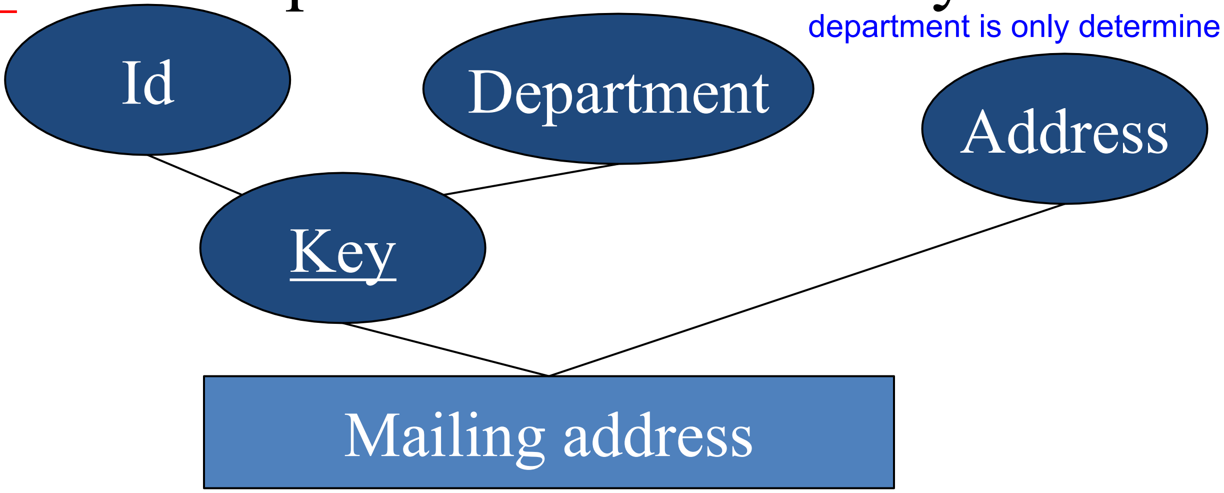
Functional Dependencies

Normalization

Functional Dependencies, Informally

- How do I know for sure if departments only have one address?
- Databases allow you to say that one attribute determines another through a functional dependency.
- So if Department determines Address but not Name, we say that there is a functional dependency from Department to Address. But Department is NOT a key.

department is only determine an address



Functional Dependencies, Formally

- A **functional dependency** **X→Y** holds if for every legal instance, for all tuples *t1*, *t2*:

$$\textit{if} \quad t1.X = t2.X \rightarrow t1.Y = t2.Y$$

- Which means given two tuples in *r*, if the X values agree, then the Y values must also agree.
- Example:

Department → Address
if t1.Department = t2.Department→ t1.Adress= t2.Adress

Identifying Functional Dependencies

- A FD is a statement about *all* allowable instances.
  - Must be identified by application semantics.
  - Given some instance of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!

Id	Department	Address
100	Computer Science	78-101 Main Mall
104	Computer Science	78-101 Main Mall
104	Math	69-201 Main Mall
105	Physics	50-205 Main Mall

- Based on this instance alone, we cannot conclude that Department → Address.

Question

- Consider the relation R with the following instance:

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

What FDs cannot be true given the instance above?

A. B→ C

[✓ B. B→ D]

C. D→B

D. All of the above can be true

E. None of the above can be true

anomalies are bad

Fixing Anomalies

Id	Department	Address
100	Computer Science	78-101 Main Mall
104	Computer Science	78-101 Main Mall
104	Math	69-201
105	Physics	50-205

⇓

Id	Department
100	Computer Science
104	Computer Science
104	Math
105	Physics

Department	Address
Computer Science	78-101 Main Mall
Math	69-201
Physics	50-205

Fixing Anomalies

Id	Department
100	Computer Science
104	Computer Science
104	Math
105	Physics

Department	Address
Computer Science	78-101 Main Mall
Math	69-201
Physics	50-205

Modification anomaly fixed: Updating the mailing address only in one place does not lead to inconsistencies in the database.

Deletion anomaly fixed: Deleting the row with id 105 does not delete the mailing address of Physics.

Insertion anomaly fixed: It is now possible to store the mailing address of a department that has no faculty members.

Fixing Anomalies Beyond our Example

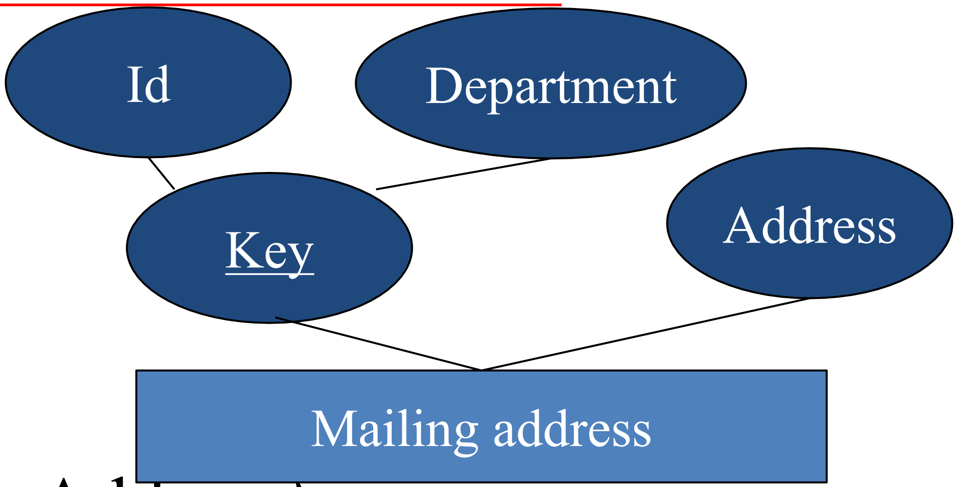
- We were able to fix the anomalies by splitting the Mailing address table.

- In the rest of this module, we will discuss how anomalies can be addressed formally.

Keys

if an attribute or a set of attributes can determine all other attributes, then it is key

- As a reminder, a key is a minimal set of attributes that uniquely identify a relation
  - i.e., a key is a minimal set of attributes that *functionally determines* all the attributes
- A superkey for a relation uniquely identifies the relation, but does not have to be minimal
  - i.e.,: key $\subseteq$ superkey
- Example
  - key {Id, Department}
  - superkey{Id, Department, Address}



Clicker question: Possible Keys

- Assume that the following FDs hold for a relation R(A,B,C,D):
  B→C
  C→B
  D→ABC

Which of the following is a key for the above relation?

A. B

B. C

C. BD

D. All of the above

E. None of the above answer

Clicker question: Possible Superkeys

- Assume the same relation R(A,B,C,D) and FDs
  $B \rightarrow C$
  $C \rightarrow B$
  $D \rightarrow ABC$

Which of the following are not a superkey for the above relation?

A. D

B. BD

C. BCD

[✓ D. All are superkeys]

E. None are superkeys

Explicit and Implicit FDs

- Given a set of (explicit) functional dependencies, we can determine implicit ones

studentid → *city*, *city* → *acode* implies *studentid* → *acode*

- A functional dependency fd is implied by a set F of functional dependencies if fd holds whenever all FDs in F hold.

fd= {studentid → acode}

F

fd1:	studentid → city
fd2:	city → acode

- **Closure of F**: the set of all FDs implied by F.

Armstrong's Axioms

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - ***Reflexivity***: If $Y \subseteq X$, then $X \rightarrow Y$
    e.g., city,major→city
  - ***Augmentation***: If $X \rightarrow Y$, then $X Z \rightarrow Y Z$ for any Z
    e.g., if sid→city, then sid,major → city,major
  - ***Transitivity***: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
    sid→ *city*, *city*→ *areacode* implies *sid*→ *areacode*
- These are *sound* and *complete* inference rules.

1. studentid→acode fd1, fd2, transitivity

F

fd1:	studentid→ city
fd2:	city→ acode

Additional Rules

- Couple of additional rules (that follow from axioms):
  - ***Union***: If X→Y and X→Z, then X→Y Z
    e.g., if sid→acode and sid→city, then sid→acode,city

1. X→X Y	fd1, augmentation
2. X Y→Z Y	fd2, augmentation
3. X→Z Y	1, 2, transitivity

F	
fd1:	X→Y
fd2:	X→Z

  - ***Decomposition***: If X→Y Z, then X→Y and X→Z
    e.g., if sid→acode,city then sid→acode, and sid→city

1 YZ →Y	Reflexivity
2 YZ → Z	Reflexivity
3. X→Y	fd1, 1, transitivity
5. X→Z	fd1, 2, transitivity

F	
fd1:	X→YZ

Example: Supplier-Part DB

- Suppliers supply parts to projects.
  - SupplierPart(sname,city,status,p#,pname,qty)
    - supplier attributes: sname, city, status
    - part attributes: p#, pname
    - supplier-part attributes: qty:

- Functional dependencies:
  - fd1: sname $\rightarrow$ city
  - fd2: city $\rightarrow$ status
  - fd3: p# $\rightarrow$ pname
  - fd4: sname, p# $\rightarrow$ qty

Exercise: Show that (sname, p#) is a superkey

Supplier-Part Key: Part 1:

Exercise: Show that (sname, p#) is a superkey of SupplierPart(sname,city,status,p#,pname,qty)

Proof has two parts:

a. Show: sname, p# is a (super)key

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

1. sname, p# → sname, p# reflex
2. sname → city fd1
3. sname → status 2, fd2, trans
4. sname, p# → city, p# 2, aug
5. sname, p# → status, p# 3, aug
6. sname, p# →

Example: Supplier-Part DB

- Suppliers supply parts to projects.
  - SupplierPart(sname,city,status,p#,pname,qty)
    - supplier attributes: sname, city, status
    - part attributes: p#, pname
    - supplier-part attributes: qty:

- Functional dependencies:
  - fd1: sname $\rightarrow$ city
  - fd2: city $\rightarrow$ status
  - fd3: p# $\rightarrow$ pname
  - fd4: sname, p# $\rightarrow$ qty

as these 2 together determine all the things

Exercise: Show that (sname, p#) is a key

Computing the Closure

- Closure for a set of attributes X is denoted by X^+
- X^+ includes all attributes of the relation IFF X is a (super) key

```
Algorithm for finding Closure of X:
Let Closure = X
    Until Closure doesn't change do
        if $a_1, \ldots, a_n \rightarrow C$ is a FD and $\{a_1, \ldots, a_n\} \in$ Closure
            Then add C to Closure
```

- Example
  - $\text{Studentid}^+ = \{\text{Studentid}\}$
  - $\text{Studentid}^+ = \{\text{Studentid, city}\}$
  - $\text{Studentid}^+ = \{\text{Studentid, city, acode}\}$

	F
fd1:	studentid $\rightarrow$ city
fd2:	city $\rightarrow$ acode

Supplier-Part Key: Closure

```
Algorithm for finding Closure of X:
Let Closure = X
    Until Closure doesn't change do
        if $a_1, …, a_n \rightarrow C$ is a FD and $\{a_1, …, a_n\} \in$ Closure
            Then add C to Closure
```

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

Ex: SupplierPart(sname,city,status,p#,pname,qty)

$\{sname,p\#\}^+ =$ sname,p#, city, status, pname, qty

$\{sname\}^+ =$ sname, city, status

$\{p\#\}^+ =$ p#, pname

So seeing if a set of attributes is a superkey means checking to see if it's closure is all the attributes – pretty simple

Question: Finding Key

- Which of the following is a key of the Relation R(ABCDE) with FD's:

F	
fd1:	$D \rightarrow C$
fd2:	$CE \rightarrow A$,
fd3:	$D \rightarrow A$
fd4:	$AE \rightarrow D$.

D+={A,C}
CE+={A}
AE+={A,E,C,D}
B

AEB or BDE

A. ABDE super key

B. BCE correct

C. CDE

D. All of these are keys

E. None of these are keys

Approaching Normality

- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, A B C.
    - No FDs hold: There is no redundancy here.
    - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value!
- **Normalization:** the process of removing redundancy from data
- We were able to fix the anomalies by splitting (decomposing) the Mailing address table, but how should we do this more formally? and how is it related to functional dependencies?

Design Guidelines

Functional Dependencies

Normalization

Normalization

- Normalization is a process that aims at achieving better designed relational database schemas using
  - Functional Dependencies
  - Primary Keys

- The normalization process takes a relational schema through a series of tests to certify whether it satisfies certain conditions
  - The schemas that satisfy certain conditions are said to be in a given *'Normal Form'*.
  - Unsatisfactory schemas are decomposed by breaking up their attributes into smaller relations that possess desirable properties (e.g., no anomalies)

Review of "Key" Concepts

- **Superkey** - A set of attributes such that no two tuples have the same values for these attributes
- **Key** - A Minimal Superkey, called a Candidate Key if more than one
  - **Primary key** - A selected candidate key
  - **Secondary key** - Remaining candidate keys
- **Prime Attribute** - An attribute that is a member of any candidate key
- **Non-prime attribute** - An attribute that is not a member of any candidate key

First Normal Form (1NF)

- A relation schema is in 1NF if domains of attributes include only atomic (simple, indivisible) values and the value of an attribute is a single value from the domain of that attribute
- 1NF disallows
  - having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple
  - "relations within relations" and "relations as attributes of tuples

Customer	Order	Items
Jones	123	Basket, Football
Gupta	876	Hat, Glass, Pencil

A non-1nf relation

Relations in 1NF

Customer	Order	Item
Jones	123	Basket
Jones	123	Football
Gupta	876	Hat
Gupta	876	Glass
Gupta	876	Pencil

Problem with this design:

Redundancy (Jones, 123)

Customer	Order	Item1	Item2	Item3	Item4
Jones	123	Basket	Football	Null	Null
Gupta	876	Hat	Glass	Pencil	Null

Problem with this design:

Too many Null values

What if an order has >max (4) items

Second Normal Form (2NF)

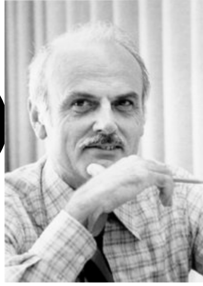
- A relation is in 2NF, if for every FD X→Y where X is a minimal key and Y is a non-prime attribute, then no proper subset of X determines Y.

not part of any candidate key

- e.g., the address relation is not in 2NF:
  - House#, street, postal_code is a minimal key
  - House#, street, postal_code → Province
  - Postal_code → province

X = House#, street, postal code
Y = province

Boyce-Codd Normal Form (BCNF)



use to avoid redundancy

using to decompose

Raymond Boyce

Ted Codd

a attribute will determine any other things

A relation R is in BCNF if for all non-trivial dependencies in R:
If X → b then X is a superkey for R

b is any attribute

- A dependency is trivial if the LHS contains the RHS, e.g., City, Province→ City is a trivial dependency
- Informally: Whenever a set of attributes of R determine another attribute, it should determine all the attributes of R.

BCNF Example

- Address(<u>House#, Street, </u>City, Province, <u>PostalCode</u>)

House#, Street, PostalCode → City

House#, Street, PostalCode → Province

PostalCode → City

PostalCode → Province

Is Address in BCNF?

because address cannot determine all the attributes

$\{PostalCode\}^+ = \{PostalCode, City, Province\}$

No. PostalCode is not a superkey

PostalCode → City is violating BCNF

Decomposing into BCNF Losslessly

1. Put FDs in standard form (only one attribute on RHS)

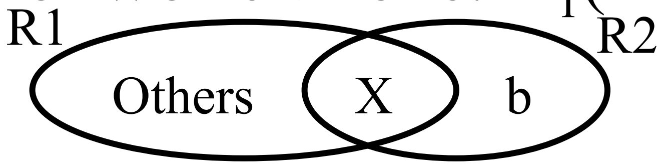
   Replace any FD X→AB with X→A and X→B

2. Add implicit FDs to your list of FDs

   If FDs contain A→B and B→C add A→C to list of FDs

3. Pick any f ∈ FD that violates BCNF of the form X→b

4. Decompose R into two relations: $R_1(A\text{-}b)$ & $R_2(X \cup b)$



5. Recurse on R_1 and R_2 using FD

Note: answer may vary depending on order you choose. That's okay

How do we decompose into BCNF losslessly?

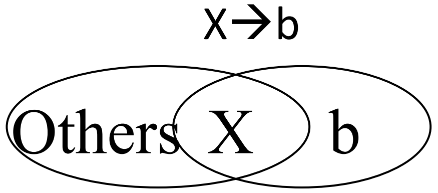
- The algorithm terminates since all two attribute relations are in BCNF.

R(X,Y)
No FD so no redundancy
$X \rightarrow Y$ so X is key, so in BCNF
$Y \rightarrow X$ so Y is key, so in BCNF
$Y \rightarrow X$ and $X \rightarrow Y$, both X and Y are keys, so in BCNF

BCNF Decomposition Example

- Relation: R(ABCD)

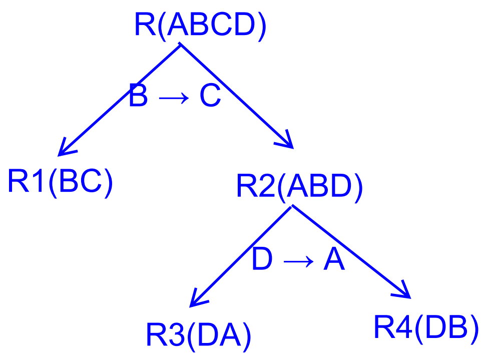
fd1:	B→C
fd2:	D→A



closure and keys:
A+ = A
B+ = BC
C+ = C
D+ = AD

BD+ = BDAC is the only key

considering B → C, is B a Superkey in R?
No. decompose
R1(B,C) R2(A,B,D)
considering D → A, is D a Superkey for R2?
No. decompose
R3(D,A), R4(D,B)


$$\begin{tikzcd}
 & R(ABCD) \arrow[dl] \arrow[dr, "B \to C"] & & \\
R1(BC) & & R2(ABD) \arrow[dl] \arrow[dr, "D \to A"] & \\
 & R3(DA) & & R4(DB)
\end{tikzcd}$$

Determining Which FDs Apply

For an FD $X \rightarrow b$, if the decomposed relation S contains $\{X \cup b\}$, and $b \in X^+$ then the FD holds for S.

- For example. Consider relation R(A,B,C,D,E) with functional dependencies $AB \rightarrow C$, $BC \rightarrow D$, $CD \rightarrow E$, $DE \rightarrow A$, and $AE \rightarrow B$.
- Project these FD's onto the relation S(A,B,C,D).
- Does $AB \rightarrow D$ hold?
  - First check if A, B and D are all in S? They are
  - Find $AB^+ = ABCDE$
  - Then yes $AB \rightarrow D$ does hold in S.
- Does $CD \rightarrow E$ hold?
  - No CDE are not all in S

Question

- Consider relation R(A,B,C,D,E) with functional dependencies AB → C, BC → D, CD → E, DE → A, and AE → B. Project these FD's onto the relation S(A,B,C,D).
- Which of the following hold in S?

A. A→B

A+ = A. so it does not hold

B. AB→E

E is not in S

C. AE→B

[✓ D. BCD→A]

Yes. BCD+ = ABCDE & all in S

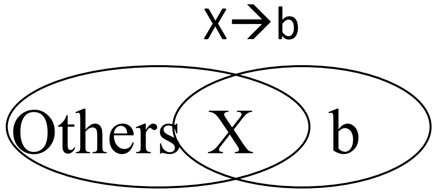
we use all FDs for finding closures, so for D we use DC → E even though E is not present in S

E. None of the above

Another BCNF Example

- R(ABCDE)

fd1: AB→C
fd2: D→E



Finding closure of the following:
- AB+ = ABC
- D+ = DE

AB → C violates BCNF in R
decompose

D → E violates BCNF in R2
decompose

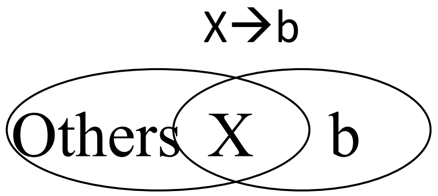
$$\begin{tikzcd}
 & R(ABCDE) \arrow[dl] \arrow[dr, "AB \rightarrow C"] & & \\
R1(ABC) & & R2(ABDE) \arrow[dl] \arrow[dr, "D \rightarrow E"] & \\
 & R3(DE) & & R4(ABD)
\end{tikzcd}$$

Final answer: R1(ABC), R3(ABD), R4(DE)

Example: Implicit FDs matter

- R(A,B,C,D,E,F)

fd1	A→B
fd2	DE→ F
fd3	B→C



A+ = ABC
B+ = BC
DE+ = DEF
A → B is violating BCNF in R
  not a Superkey → decomposite
DE → F is violating BCNF in R2
  decomposite
A → C is violating BCNF in R4
  decomposite

$$\begin{tikzcd}
 & R(ABCDEF) \arrow[dl] \arrow[dr, "A \to B"] & & & \\
R1(AB) & & R2(ACDEF) \arrow[dl] \arrow[dr, "DE \to F"] & & \\
 & R3(DEF) & & R4(ACDE) \arrow[dl] \arrow[dr, "A \to C"] & \\
 & & R5(AC) & & R6(ADE)
\end{tikzcd}$$

Clicker Exercise: More BCNF

- Let R(ABCD) be a relation with functional dependencies $A \rightarrow B$, $C \rightarrow D$, $AD \rightarrow C$, $BC \rightarrow A$
- Decompose into BCNF. Which of the following is a lossless-join decomposition of R into Boyce-Codd Normal Form (BCNF)?

sequence?

A. {AB, AC, BD}

if randomly decompose → cause lossy joint

B. {AB, AC, CD}

AB, ACD
CD AC

C. {AB, AC, BCD}

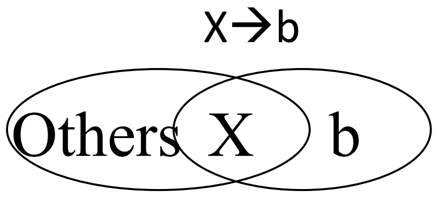
D. All of the above

E. None of the above

Clicker Exercise: More BCNF

- Let R(ABCD)

fd1	A→B
fd2	C→ D
fd3	AD→C
fd4	BC →A



AB+ = AB
C+ = CD
AD+ = ADBC
BC+ = ABC

BCNF is great, but…

- Guaranteed that there will be no redundancy of data
- Easy to understand (just look for superkeys)
- Easy to do.
- So what is the main problem with BCNF?
  - For one thing, BCNF may not preserve all dependencies

Dependency Preservation

A functional dependency X→Y is preserved in a relation R, if R contains all of the attributes of X and Y.

Example:

- R(A,B,C)   fd1 A→B   ,   fd2 A→D

fd1 is preserved in R and fd2 is not preserved in R

An Illustrative BCNF example

Unit	Company	Product

Unit → Company
Company, Product → Unit

Is unit a key?

Unit	Company

Unit	Product

Unit → Company

The second functional dependency is no longer preserved.

So What's the Problem?

Unit	Company
SKYWill	ABC
Team Meat	ABC

Unit	Product
SKYWill	Databases
Team Meat	Databases

Unit $\rightarrow$ Company

No problem so far. All *local* FD's are satisfied.
Let's join the relations back into a single table again:

Unit	Company	Product
SKYWill	ABC	Databases
Team Meat	ABC	Databases

Violates the FD: Company, Product $\rightarrow$ Unit

Third Normal Form (3NF)

A relation R is in 3NF if for all non-trivial dependencies in R:
    If X $\rightarrow$ b then
        X is a superkey for R OR
        b is a prime attribute of R

- A dependency is trivial if the LHS contains the RHS, e.g., City, Province$\rightarrow$ City is a trivial dependency

Example: R(Unit,Company, Product)

- Unit $\rightarrow$ Company
- Company, Product $\rightarrow$ Unit
  Keys: {Company, Product}, {Unit,Product}

- R is not in BCNF but is in 3NF.

To decompose into 3NF we rely on the *minimal cover*

Minimal Cover for a Set of FDs

- Goal: Transform FDs to be as small as possible.

Minimal cover G for a set of FDs F:

1. Closure of F = closure of G (i.e., imply the same FDs)
2. Right hand side of each FD in G is a single attribute
3. If we delete an FD in G or delete attributes from an FD in G, the closure changes

- Informally: Every FD in G is needed, and is "*as small as possible*" in order to get the same closure as F

Example:

- A→B, ABCD→E, EF→GH, ACDF→EG has the following minimal cover:
  - A→B, ACD→E, EF→G and EF→H

Finding Minimal Covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:

$A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow G$, $EF \rightarrow H$, $ACDF \rightarrow EG$

only one attribute on RHS

- Replace last rule with
  - $ACDF \rightarrow E$
  - $ACDF \rightarrow G$

Finding Minimal Covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:

$A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow G$, $EF \rightarrow H$, $ACDF \rightarrow E$, $ACDF \rightarrow G$

- Can we take anything away from the LHS?
  - $ABCD^+ = ABCDE$
  - $ACD^+ = ABCDE$, so remove B from the FD

Finding Minimal Covers of FDs

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:

$A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$, $EF \rightarrow H$, $ACDF \rightarrow E$, $ACDF \rightarrow G$

- Let's find $ACDF^+$ without considering the highlighted FDs
  - $ACDF^+ = ACDFEBGH$, so I can remove the highlighted rules

- Final answer: $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$, $EF \rightarrow H$

Minimal Example Cover

- Consider the relation R(CSJDPQV) with FDs $C \rightarrow SJDPQV$, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$

Find a minimal cover

$C \rightarrow S$, $C \rightarrow J$, $C \rightarrow D$, $C \rightarrow P$, $C \rightarrow Q$, $C \rightarrow V$, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$

Minimize LHS before deleting redundant FDs

- If step 3 is done prior to step 2, the final set of FDs could still contain redundant FDs

1. One attribute on RHS
2. Delete Redundant FDs
3. Minimize LHS of each FD

Does not work

- ABCD→E, E→D, A→B, AC→D
  - Let's delete redundant FDs.
  - None of the FDs are redundant.
- ABCD→E, E→D, A→B, AC→D
  - Now let's shorten FDs
  - ABCD→E can be replaced by AC→E
- However the current set of FDs are not minimal
  - AC→E, E→D, A→B, AC→D
  - The highlighted FD can be deleted

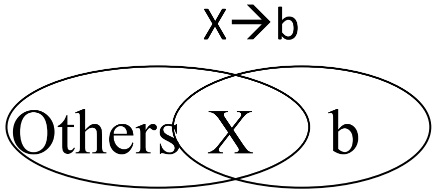
Decomposition into 3NF Using Minimal Cover

- Decomposition into 3NF:
  1. Given the FDs F, compute F': the *minimal cover for F*
  2. Decompose using F' if violating 3NF similar to how it was done for BCNF
  3. After each decomposition identify the set of dependencies N in F' that are not preserved by the decomposition.
  4. For each $X \rightarrow a$ in N create a relation $R_n(X \cup a)$ and add it to the decomposition

3NF Example

- Example: R(ABCDE)

fd1 AB→C
fd2 C→ D



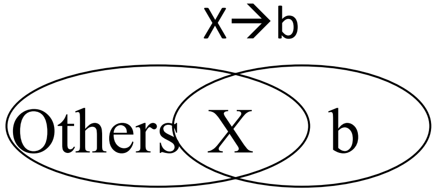
cover already minimal
ABE+ = ABCDE only key
AB → C is violating 3NF
decompose

$$\begin{tikzcd}
 & R(ABCDE) \arrow[dl] \arrow[d, "AB \to C"] \arrow[dr, "C \to D"] & \\
R1(ABC) & R2(ABDE) \arrow[dl] \arrow[dr, "AB \to D"] & R3(CD) \\
R4(ABD) & & R5(ABE)
\end{tikzcd}$$

Another 3NF Example

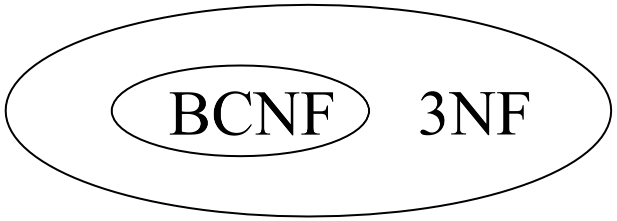
- Let R(CSJDPQV) be a relation with the following FDs. Decompose R into 3NF

fd1	SD→P
fd2	JP→C
fd3	J→S



BCNF & 3NF

- BCNF guarantees removal of all anomalies
- 3NF has some anomalies, but preserves all dependencies
- If a relation R is in BCNF it is in 3NF.
- A 3NF relation R may not be in BCNF.



- Most organizations go to BCNF or 3NF.

? Clicker Question: BCNF and 3NF

- Consider the following relation and functional dependencies:
  R(ABCD) FD's: ACD → B ; AC → D ; D → C ; AC → B
  Which of the following is true:

A. R is in neither BCNF nor 3NF

B. R is in BCNF but not 3NF

C. R is in 3NF but not in BCNF

D. R is in both BCNF and 3NF

ACD+ = ABCD
AC+ = ACDB
D+ = DC
AD+ = ADCB
keys: AC, AD

D → C (D not key so NOT BCNF)
C is part of a minimal key, so R is in 3NF

Denormalization

- Process of intentionally violating a normal form to gain performance improvements
- Performance improvements:
  1. Fewer joins
  2. Reduces number of foreign keys

- Useful in data analysis or if certain queries often require (joined) results, and the queries are frequent enough.

Learning Outcomes

Description	Tag
Provide examples of modification, insertion, and deletion anomalies.	Functional-dependencies
Explain the term functional dependency.	
Given a set F of functional dependencies and a functional dependency fd, determine whether fd can be inferred from F using Armstrong's Axioms.	
Given a set of functional dependencies that hold over a table, determine the keys.	
Given a set of functional dependencies that hold over a table, determine the superkeys.	
Given a set F of functional dependencies and X of attributes of a relation, compute the closure of X, which is X+	
Given a relation R, be able to correctly decompose it into two or more relations	Decomposition
Justify why lossless join decompositions are preferred decompositions.	
Explain the benefits of Normalization.	Normalization
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 1NF.	
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 2NF.	
Explain what dependency preservation means.	
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in 3NF.	
Given a relation schema R and a set of functional dependencies on it, show that R is/isn't in BCNF.	
Describe the process and benefits of Denormalization.	
Reason with the logical foundation of the relational data model and understand the fundamental principles of correct relational database design.	