Schema Refinement and Normalization

Some slide content courtesy of Zack Ives & Raghu Ramakrishnan

Schema Refinement & Design Theory

- ER Diagrams give us a start in logical schema design
  - EER Modelling is a subjective process.
  - This may result in many different relation schema
- How can we validate whether the relations created are good?
  - There are minimal number of attributes necessary to support the data requirements of the enterprise;
  - attributes with a close logical relationship are found in the same relation;
  - minimal redundancy with each attribute represented only once, with exception of attributes that form all or part of foreign keys.

Not All Designs are Equally Good

Why is this a poor schema design?

Lecturer(lectID, name, salary, deptCode, dname, deptPhone, bldg)

Example

lecID	name	salary	deptCode	dname	deptPhone	bldg
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Focus on the Bad Design

- Redundancy?
  - Unnecessary repetition of data in relations
  - Certain items (e.g., deptCode, deptPhone, dname, bldg) get repeated

- Redundancy causes anomalies and consistency issues
- Types of anomalies:
  - Insertion Anomaly
  - Deletion Anomaly
  - Modification Anomaly

Focus on the Bad Design

- Insertion Anomaly
  - Example
  - Inserting lecturer -> re-entry dept

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Focus on the Bad Design

- Deletion Anomaly
  - Example:
  - Deleting lecturer -> lose dept info

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Focus on the Bad Design

- Modification Anomaly
  - Example:
  - phone changes ->dept info update

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Better Design - Normalization

- How can redundancies be avoid ?
  - Decompose relations to smaller relations without redundancies
  - Example:
  - LecturerInfo(lecID, name, salary, deptCode)
  - Dept(deptCode, dname, deptPhone, building)
  - No insertion, deletion or modification anomalies

- Normalization:
  - A formal process to minimize redundancies in relations

Considerations when decomposing relations

- Loss-less join property:
  - No loss of data from original relation
  - Can obtain original relation by joining decomposed relation
- Dependency-preserving property:
  - No loss of dependencies (i.e. functional dependencies) during decomposition

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Example of a lossy join

- Decomposing S to S_1 and S_2

S

S	P	D
S1	P1	D1
S2	P2	D2
S3	P1	D3

S_1

S	P
S1	P1
S2	P2
S3	P1

S_2

P	D
P1	D1
P2	D2
P1	D3

Joining S_1 and S_2 on P will result in spurious tuples: data is lost! Not preserving the **lossless-join** property

Lossless join decomposition

- Theorem

  Relation R_1 and R_2 *is* a lossless-join decomposition of relation S, *if* $R_1 \cap R_2$ (i.e. common attributes for R_1 and R_2) contains a key for either R_1 or R_2.

- Accordingly, if $X \rightarrow Y$ is causing an anomaly in relation R, we can decompose as follows:
  - Relation1: R-Y
  - Relation2: XY

  Relation1 $\cap$ Relation2 = {X} and X is key for Relation2

Functional Dependency

- Consider the lecturer relation again:

Lecturer

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Whenever the same deptCode appear, same dept info appear

Redundancy

Functional Dependency (Contd.)

- Formally, we can define a functional dependency as follows:

- A functional dependency, denoted by $X \rightarrow Y$, where X and Y are sets of attributes in relation R, specifies the following constraint:

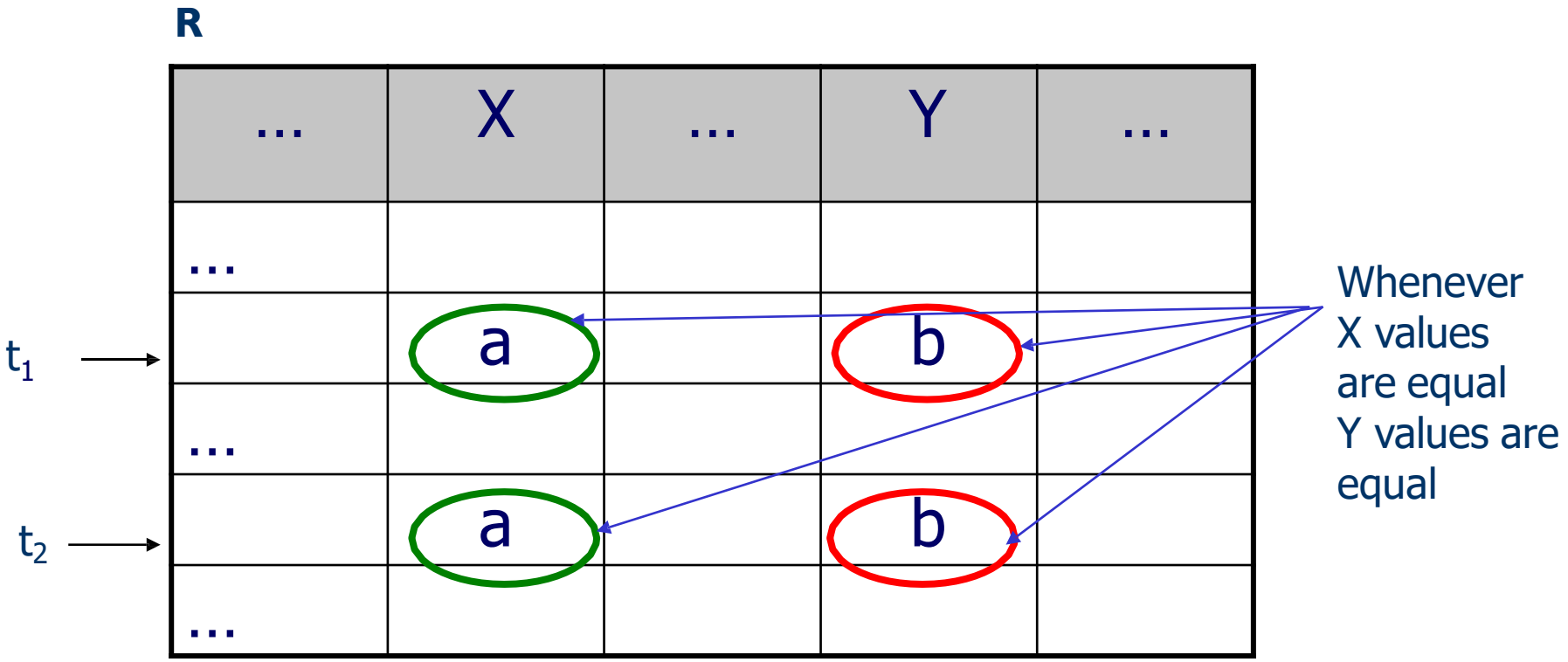
  Let t_1 and t_2 be tuples of relation R for **any** given instance

  whenever $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

  where $t_i[X]$ represents the values for X in tuple t_i

Functional Dependency (Contd.)

- Graphically, if X $\rightarrow$ Y in relation R, then



Functional Dependency (Contd.)

- Points to note:
  - Functional dependency X→Y does not necessarily mean Y→X
    - E.g. Lecturer relation

deptCode → building

However,

building ↛ deptCode

Many departments may share the same building!

Functional Dependency (Contd.)

- Functional dependency must hold for **any** instance (i.e. all instances)
- You cannot determine a functional dependency by considering one instance alone
- Semantics (meaning) of attributes in the domain (i.e. enterprise) needs to be understood
- Practically, considering instances provide only hints to the existence of functional dependencies

Functional Dependency (Contd.)

- Terminology:
  - Functional dependency $X \rightarrow Y$ in relation **R**, we say,
    - X functionally determines Y
    - Y is functionally dependent on X
  - X is also called the **determinant** (i.e. left-hand side - LHS of the functional dependency)

Functional Dependency (Contd.)

- Example
  - Lecturer(<u>lecID</u>, name, salary, deptCode, dname, deptPhone, bldg)
- Functional dependencies
  - lecID $\rightarrow$ name, salary, deptCode
  - deptCode $\rightarrow$ dname, deptPhone, building

Functional Dependency (Contd.)

- **Full functionally dependency**: A functional dependency $X \rightarrow Y$ is fully functional dependent if Y is functionally dependent on X, but not on any proper subset of X.
- Example
  - lecID $\rightarrow$ deptCode — is a full functional dependency
  - lecID, name $\rightarrow$ deptCode — is a valid functional dependency but not a full functional dependency
- **Partially dependency**: if the dependency holds for part of X

Functional Dependency (Contd.)

- **Transitive dependency**: If X, Y, and Z are sets of attributes in relation R, and X $\rightarrow$ Y, Y $\rightarrow$ Z, then X $\rightarrow$ Z.
  - We say that Z is transitively dependent on X.

- Example:
    - lecID $\rightarrow$ name, salary, deptCode
    - deptCode $\rightarrow$ dname, deptPhone, building
  - Then by transitive property
    - lecID $\rightarrow$ dname, deptPhone, building

Functional Dependency (Contd.)

- Observation:
  - A functional dependency results in a redundancy unless the determinant of the functional dependency contains a key!
  - Explanation:
    - If $X \rightarrow Y$ exists in relation R and if X does not contain a key, then there can be multiple distinct tuples in R with the same values for X which also mean the same Y values repeat for these tuples.
    - However, if X contains a key, each tuple's X-value is unique. Therefore no redundancy based on the functional dependency.

Normal Forms

- Unnormalized Form: a table that contains one or more repeating groups
- There are many normal forms allowing varying degrees of redundancy

- We will study the following
  - 1st Normal Form (1NF)
  - 2nd Normal Form (2NF)
  - 3rd Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)

1st Normal Form

- A relation is in 1st normal form if each attribute value is a single, atomic value from its domain
- That is, an attribute value cannot be multiple or composite value
- By definition of relational model (i.e. domain constraint), every relation is in 1st normal form

1st Normal Form (Contd.)

- Example: unnormalized relation

Student

stdID	name	Course	cName	semester	grade
S001	Paul Lee	CS2040	Database Sys.	2	A
		CS2031	Networks	3	B+
S002	Mary Smith	CS1001	Found. of IT	1	A
		CS2040	Database Sys.	2	A-

- Example: Relation in 1st Normal Form

Student

stdID	name	course	cName	semester	grade
S001	Paul Lee	CS2040	Database Sys.	2	A
S001	Paul Lee	CS2031	Networks	3	B+
S002	Mary Smith	CS1001	Found. of IT	1	A
S002	Mary Smith	CS2040	Database Sys.	2	A-

2nd Normal Form

- A relation R is in 2nd normal form if
  - R is in 1st normal form, and
  - Every non-candidate key attribute is fully functionally dependent on a candidate key

2nd Normal Form (contd.)

- Example

Student (stdID, course, semester, name, cName, grade)

Functional dependencies:

FD1: stdID $\rightarrow$ name (partial dependency)

FD2: course $\rightarrow$ cName (partial dependecy)

FD3: stdID, course, semester $\rightarrow$ grade

stdID	name	course	cName	semester	grade
S001	Paul Lee	CS2040	Database Sys.	2	A
S001	Paul Lee	CS2031	Networks	3	B+
S002	Mary Smith	CS1001	Found. of IT	1	A
S002	Mary Smith	CS2040	Database Sys.	2	A-

2nd Normal Form (contd.)

- Example:

  Decomposing the Student relation

  Student (stdID, course, semester, name, cName, grade)

- Step1: based on FD1

  Student1 (stdID, name)

  Student2(stdID, course, semester, cName, grade)

- Step2: based on FD2

  Student1 (stdID, name)

  Student2(course, cName)

  Student3(stdID,course, semester, grade)

Not in 2nd Normal Form Because of FD2

Decomposed relations in 2nd Normal Form

Decomposing based on FD1 & FD2

- Student (<u>stdID, course, semester</u>, name, cName, grade)

FD1: stdID $\rightarrow$ name (partial dependency)

Step 0: find R, X, Y, in the definition:

R = <u>stdID, course, semester</u>, name, cName, grade

Now we want ***to fix a partial dependency FD1***, so

X = stdID, Y = name, XY = stdID, name

Step 1: get relation1: r1 (R-Y= <u>stdID, course, semester</u>, ~~name~~, cName, grade)

Step 2: get relation2: r2 (XY= <u>stdID</u>, name)

Now, based on r1, r2, and we want to eliminate FD2: course $\rightarrow$ cName:

Keep r2, convert r1 into 2 relations in a same way as on FD1:

X = course, Y = cName, XY = course, cName

Step 1: get relation1: r3 (R-Y= <u>stdID, course, semester</u>, ~~name~~, ~~cName~~, grade)

Step 2: get relation2: r4 (XY= <u>course,</u> cName)

Final result is r2, r3, r4: Student1 (stdID, name), Student2(course, cName), Student3(stdID, course, semester, grade)

3rd Normal Form

- A relation R is in 3rd Normal Form if
  - R is in 2nd Normal Form, and
  - No non-candidate-key attribute is ransitively dependent on a candidate key

3rd Normal Form (contd.)

- Example: Lecturer relation

LecturerInfo(<u>lecID</u>, name, salary, deptCode, dname, deptPhone, bldg.)

Functional Dependencies:

FD1: lecID $\rightarrow$ name, salary, deptCode (Primary Key)

FD2: deptCode $\rightarrow$ dname, deptPhone, building

(Transitive dependency)

- This relation is in 2nd Normal Form but not in 3rd Normal Form because of FD2.

3rd Normal Form (contd.)

- Decomposing Lecturer relation based on FD2:

  LecturerInfo(lecID, name, salary, deptCode, dname, deptPhone, bldg)

  - Method:
    - Form a new relation out of the transitive dependency
    - Keep the PK relation
  - Result:
    - Lecturer1 (deptCode, dname, deptPhone, Building)
    - Lecturer2(lecID, name, salary, deptCode)
  - Decomposed relations are in 3rd Normal Form

Boyce-Codd Normal Form

- Example:

ClientInterview(clientNo, interviewDate, interviewTime, staffNo, roomNo)

The ClientInterview relation has three candidate keys:

- (clientNo, interviewDate)
- (staffNo,interviewDate, interviewTime),
- (roomNo, interviewDate, interviewTime).

ClientInterview has three composite candidate keys, which overlap by sharing the common attribute interviewDate. (clientNo, interviewDate) is selected to act as the primary key for this relation.

FD1: clientNo, interviewDate → interviewTime, staffNo, roomNo

FD2: staffNo, interviewDate, interviewTime → clientNo

FD3: roomNo, interviewDate, interviewTime → staffNo, clientNo

FD4: staffNo, interviewDate → roomNo

Boyce-Codd Normal Form (contd.)

- A relation R is in BCNF if and only if
  - Every functional dependency, X → Y in R, X contains a candidate key

- Because of FD1 – FD4, the relation is in 3rd Normal Form
- Because of FD4, the relation is not in BCNF

Boyce-Codd Normal Form (contd.)

- Decomposing based on FD4:
  - ClientInterview1 (staffNo, interviewDate, roomNo)
  - ClientInterview2(clientNo, interviewDate, staffNo, interviewTime)

  The above relations are in BCNF!
  However,
  FD3 (roomNo, interviewDate, interviewTime → staffNo, clientNo)
  is lost (not preserving the dependency)

- A dependency preserving decomposition to BCNF is not always possible (Therefore may stop at 3NF)

- We can always get a lossless join and dependency preserving decomposition to 3rd Normal Form

Normal Forms Compared

- BCNF is preferable, but sometimes in conflict with the goal of dependency preservation
  - It's strictly stronger than 3NF
- Let's see algorithms to obtain:
  - A BCNF lossless join decomposition
  - A 3NF lossless join, dependency preserving decomposition

Armstrong's axioms

- Armstrong's axioms are sound and complete inference rules for FDs
  - Sound: all the derived FDs (by using the axioms) are those logically implied by the given set
  - Complete: all the logically implied (by the given set) FDs can be derived by using the axioms.

Armstrong's Axioms: Inferring FDs

Some FDs exist due to others; can compute using Armstrong's axioms:

- Reflexivity: If $Y \subseteq X$ then $X \rightarrow Y$ (*trivial dependencies*)

  name, sid $\rightarrow$ name

- Augmentation: If $X \rightarrow Y$ then $XW \rightarrow YW$

  serno $\rightarrow$ subj so serno, exp-grade $\rightarrow$ subj, exp-grade

- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

  serno $\rightarrow$ cid and cid $\rightarrow$ subj

  so serno $\rightarrow$ subj

Armstrong's Axioms Lead to…

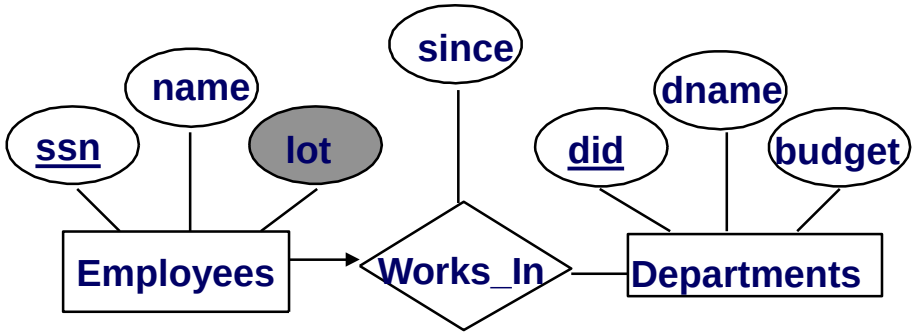
- Union: If $X \rightarrow Y$ and $X \rightarrow Z$
  then $X \rightarrow YZ$
- Pseudotransitivity: If $X \rightarrow Y$ and $WY \rightarrow Z$
  then $XW \rightarrow Z$
- Decomposition: If $X \rightarrow Y$ and $Z \subseteq Y$
  then $X \rightarrow Z$

Let's prove these from Armstrong's Axioms

Relationship between FDs and Keys

- Given R(A, B, C).
  - A $\rightarrow$ ABC means that A is a key.
- In general,
  - X $\rightarrow$ R means X is a (super)key.

– ssn $\rightarrow$ did



Reasoning About FDs

- Given some FDs, we can usually infer additional FDs
  - *ssn* $\rightarrow$ *did*, *did* $\rightarrow$ *lot* implies *ssn* $\rightarrow$ *lot*
  - *A* $\rightarrow$ *BC* implies *A* $\rightarrow$ *B*
- An FD f is *logically implied* by a set of FDs F iff holds whenever all FDs in F hold.
  - F+ = closure of F is the set of all FDs that are implied by F.
- Closure of set of FDs

Defn. Let *F* be a set of FD's.

Its *closure*, F^+, is the set of all FD's:

{X $\rightarrow$ Y | X $\rightarrow$ Y is derivable from *F* by Armstrong's Axioms}

Attribute Closures - X^+

Defn. The closure of an attribute set X, X^+, is:

$$X^+ = \bigcup \{Y \mid X \rightarrow Y \in F^+\}$$

- This answers the question "is Y determined (transitively) by X?"; compute X^+ by:

closure := X;
repeat until no change {
if there is an FD U → V in *F*
such that U is in *closure*
then add V to *closure*}

Computing X^+

- Input F (a set of FDs), and X (a set of attributes)
- Output: Result=X^+ (under F)
- Method:
  - Step 1: Result :=X;
  - Step 2: Take $Y \rightarrow Z$ in F, **and** Y is in Result, do:

    Result := Result $\cup$ Z
  - Repeat step 2 until Result cannot be changed and then output Result.

Example of Attribute Closure X^+

- Does $F = \{A \rightarrow B,\quad B \rightarrow C,\quad CD \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Step 1: Result = A
Step 2: Consider $A \rightarrow B$, Result = AB
Consider $B \rightarrow C$, Result = ABC
Consider $CD \rightarrow E$, CD is not in ABC, so stop
Step 3: $A^+ = \{ABC\}$
E is NOT in A^+, so $A \rightarrow E$ is NOT in F^+

Example of computing X^+

$F = \{A \rightarrow B, AC \rightarrow D, AB \rightarrow C\}$?

What is X^+ for $X = A$? (i.e. what is the attribute closure for A?)

Answer: $A^+ = ABCD$

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if X is a superkey, we compute X^+, and check if X^+ contains all attributes of *R*.
- Testing functional dependencies
  - To check if a functional dependency $X \rightarrow Y$ holds (or, in other words, is in F^+), just check if $Y \subseteq X^+$.
  - That is, we compute X^+ by using attribute closure, and then check if it contains Y.
  - Is a simple and cheap test, and very useful
- Computing closure of F

Equivalence of FD sets

Defn. Two sets of FD's, F and G, are *equivalent* if their closures are equivalent, $F^+ = G^+$

e.g., these two sets are equivalent:

$\{XY \rightarrow Z, X \rightarrow Y\}$ and

$\{X \rightarrow Z, X \rightarrow Y\}$

- F^+ contains a huge number of FD's (exponential in the size of the schema)
- Would like to have smallest "representative" FD set

Equivalence of FD sets (Example)

- R(A,B,C)
- Consider F={A→B, B→C, C→A} and G={C→B, B→A, A→C}
- Are F and G equivalent?
- If F covers G
  - All FDs in G can be derived from F

  G: $C \rightarrow B$ $B \rightarrow A$ $A \rightarrow C$

  $C_F^+ = CA\underline{B}$ $B_F^+ = BC\underline{A}$ $A_F^+ = AB\underline{C}$
- If G covers F
  - All FDs in F can be derived from G

  F: $A \rightarrow B$ $B \rightarrow C$ $C \rightarrow A$

  $A_G^+ = AC\underline{B}$ $B_G^+ = BA\underline{C}$ $C_G^+ = CB\underline{A}$
- Therefore, F ≡ G

Minimal Cover

Defn. A FD set F is *minimal* if:

1. Every FD in F is of the form $X \rightarrow A$, where A is a single attribute

2. For no $X \rightarrow A$ in F is:
   $F - \{X \rightarrow A\}$ equivalent to F

3. For no $X \rightarrow A$ in F and $Z \subset X$ is:
   $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ equivalent to F

we express each FD in simplest form

in a sense, each FD is "essential" to the cover

Defn. F is a *minimum cover* for G if F is minimal and is equivalent to G.

e.g.,

$\{X \rightarrow Z, X \rightarrow Y\}$ is a minimal cover for
$\{XY \rightarrow Z, X \rightarrow Y\}$

Minimal Cover Example

- R(A,B,C,D,E)
- Consider F={A→D, BC→AD, C→B, E→A, E→D}
- Find Minimal cover of F?
- All FDs have singleton RHS
  - BC→AD: Decompose to BC→ A, BC→D
  - F={A→D, BC→A, BC→D, C→B, E→A, E→D}
- Find extraneous attribute in LHS
  - Examine LHS of each FD that has 2 or more attributes.
  - BC→A
    - $B^+ = B$ $C^+ = CBAD$ (eliminate B)
  - BC→D
    - $B^+ = B$ $C^+ = CBAD$ (eliminate B)
  - F ={A→D, C→A, C →D, C→B, E→A, E→D}

Minimal Cover Example (Contd)

- F ={A→D, C→A, C→D, C→B, E→A, E→D}
- Eliminate redundant FDs
  - Examine each FD one at a time, see if FD is needed, if not eliminate
  - A→D
    - Pretend this FD does not exist and see if we can determine D from A
    - A^+ = A (Keep)
  - C→A
    - C^+ = CDB (Keep)
  - C→D
    - C^+ = CADB (includes D, eliminate)
  - C→B
    - C^+ = CAD (Keep)
  - E→A
    - E^+ = ED (Keep)
  - E→A
    - E^+ = EAD (eliminate)

F = {A→D, C→A, C→B, E→A}

More on Closures

If F is a set of FD's and $X \rightarrow Y \notin F^+$
then for some attribute $A \in Y$, $X \rightarrow A \notin F^+$

Proof by counterexample.

Assume otherwise and let $Y = \{A_1, \ldots, A_n\}$
Since we assume $X \rightarrow A_1$, ..., $X \rightarrow A_n$ are in F^+

then $X \rightarrow A_1 \ldots A_n$ is in F^+ by union rule,

hence, $X \rightarrow Y$ is in F^+ which is a contradiction

Why Armstrong's Axioms?

Why are Armstrong's axioms (or an equivalent rule set) appropriate for FD's? They are:

- *Consistent*: any relation satisfying FD's in F will satisfy those in F^+
- *Complete*: if an FD $X \rightarrow Y$ cannot be derived by Armstrong's axioms from F, then there exists some relational instance satisfying F but not $X \rightarrow Y$

➢ In other words, Armstrong's axioms derive ***all*** the FD's that should hold

Dependency Preservation

Ensures we can "easily" check whether a FD $X \rightarrow Y$ is violated during an update to a database:

- The *projection* of an FD set F onto a set of attributes Z, F_Z is

$$\{X \rightarrow Y \mid X \rightarrow Y \in F^+, X \cup Y \subseteq Z\}$$

  i.e., it is those FDs local to Z's attributes

- A decomposition $R_1, \ldots, R_k$ is *dependency preserving* if $F^+ = (F_{R1} \cup \ldots \cup F_{Rk})^+$

The decomposition hasn't "lost" any essential FD's, so we can check without doing a join

Example of Lossless and Dependency-Preserving Decompositions

Given relation scheme

R(name, street, city, st, zip, item, price)

And FD set
name → street, city
street, city → st
street, city → zip
name, item → price

Consider the decomposition

R_1(name, street, city, st, zip) and R_2(name, item, price)

- *Is it lossless?*
- *Is it dependency preserving?*

What if we replaced the first FD by name, street → city?

Another Example

Given scheme: R(sid, fid, subj)

and FD set: fid $\rightarrow$ subj

sid, subj $\rightarrow$ fid

Consider the decomposition

R_1(sid, fid) and R_2(fid, subj)

- *Is it lossless?*
- *Is it dependency preserving?*

FD's and Keys

- Ideally, we want a design s.t. for each nontrivial dependency $X \rightarrow Y$, X is a superkey for some relation schema in R
  - We just saw that this isn't always possible
- Hence we have two kinds of *normal forms*

Summary

- We can always decompose into 3NF and get:
  - Lossless join
  - Dependency preservation
- But with BCNF we are only guaranteed lossless joins
- BCNF is stronger than 3NF: every BCNF schema is also in 3NF