Design Theory for Relational Databases

Functional Dependencies

Decompositions

Normal Forms

Relational Schema Design

- Goal of relational schema design is to avoid anomalies and redundancy.
  - *Update anomaly* : one occurrence of a fact is changed, but not all occurrences.
  - *Deletion anomaly* : valid fact is lost when a tuple is deleted.

Example of Bad Design

Drinkers(name, addr, beersLiked, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because each of the ???'s can be figured out by using the Functional Dependencies (see next) name -> addr favBeer and beersLiked -> manf.

This Bad Design Also Exhibits Anomalies

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to Addr2, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Functional Dependencies

- X -> Y is an assertion about a relation R that whenever two tuples of R agree on all the attributes of X, then they must also agree on all attributes in set Y.
  - Say "X -> Y holds in R."
  - Convention: …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes.
  - Convention: we denote sets of attributes, just ABC, rather than $\{A,B,C\}$.

Splitting Right Sides of FD's

- $X\text{->}A_1A_2\ldots A_n$ holds for R exactly when each of $X\text{->}A_1$, $X\text{->}A_2$,…, $X\text{->}A_n$ holds for R.
- Example: $A\text{->}BC$ is equivalent to $A\text{->}B$ and $A\text{->}C$.
- There is no splitting rule for left sides.
- We'll generally express FD's with singleton right sides.

Example: FD's

Drinkers(name, addr, beersLiked, manf, favBeer)

- Reasonable FD's to assert:
  1. name -> addr favBeer
     - Note this FD is the same as name -> addr and name -> favBeer.
  2. beersLiked -> manf

Example: Possible Data

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Because name -> addr

Because name -> favBeer

Because beersLiked -> manf

Keys of Relations

- *K* is a *superkey* for relation *R* if *K* functionally determines all of *R*.
- *K* is a *key* for *R* if *K* is a superkey, but no proper subset of *K* is a superkey.

- If two tuples agree on key attributes -> they must agree on all attributes, so they are the same tuple.
- We cannot have two tuples with the same key values.
- Key -> unique identifier

Example: Superkey

Drinkers(name, addr, beersLiked, manf, favBeer)

- {name, beersLiked} is a superkey because together these attributes determine all the other attributes.
  - name -> addr favBeer
  - beersLiked -> manf

Example: Key

- {name, beersLiked} is a key because neither {name} nor {beersLiked} is a superkey.
  - name doesn't -> manf;
  - beersLiked doesn't -> addr.
- There are no other keys, but lots of superkeys.
  - Any superset of {name, beersLiked}.

(See in Textbook: Exercise 3.1.3 [page 6])

Where Do Keys Come From?

1. Just assert a key *K*. (someone tells the key)
   - The only FD's are *K* -> *A* for all attributes *A*.
2. Assert FD's and deduce the keys by systematic exploration.

Someone tells constraints in FD form, and we find the key(s).

More FD's From Univ.

- Example: "no two courses can meet in the same room at the same time" tells us: hour, room -> course.

The constraint stated as a sentence can be expressed as a FD.

Sometimes we have some knowledge about the requirements and relationships between our data and we have to express it precisly in FD form.

Inferring FD's

- We are given FD's X_1 -> A_1, X_2 -> A_2,…, X_n -> A_n , and we want to know whether an FD Y -> B must hold in any relation that satisfies the given FD's.
  - Example: If A -> B and B -> C hold, surely A -> C holds, even if we don't say so.

  (See in Textbook: Example 3.4 [page 6])
- Important for design of good relation schemas to find all FDs which follow from the given FDs.

Inference Test

- To test if Y -> B, start by assuming two tuples agree in all attributes of Y.
  (can we have the second tuple?)

$\leftarrow Y \rightarrow$

0000000. . . 0

00000?? . . . ?

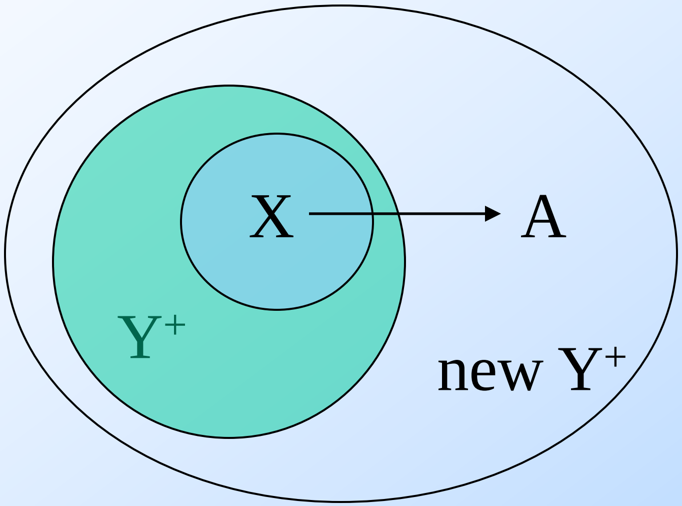
Inference Test – (2)

- Use the given FD’s to infer that these tuples must also agree in certain other attributes.
  - If B is one of these attributes, then *Y* -> *B* is true.
  - Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves *Y* -> *B* does not follow from the given FD’s. (See in Textbook, Exercise 3.2.4)

Closure Test

- An easier way to test is to compute the *closure* of Y, denoted Y^+.
- Basis: $Y^+ = Y$.
- Induction: Look for an FD's left side X that is a subset of the current Y^+. If the FD is $X \text{->} A$, add A to Y^+.

(See in Textbook, Algorithm 3.7: Closure of a Set of Attributes (page 10); Example 3.8 (page 11), Example 3.9 (page 11))



We apply X->A and add A to Y^+ in the next step.

Example 3.8 (Textbook)

Let us consider a relation with attributes A, B, C, D, E, and F.

Suppose that this relation has the FD's AB->C, BC->AD, D->E, and CF->B.

What is the closure of {A,B}?

Notice: BC->AD is equivalent to {BC->A, BC->D} (splitting rule)

Initialize X={A,B} then apply

AB->C … X={A,B,C}

BC->D … X={A,B,C,D}

D->E … X={A,B,C,D,E}

No more changes, so $\{A,B\}^+ = \{A,B,C,D,E\}$

Does AB->E follow from the original FD's ? Yes!

Does AB->F follow from the original FD's ? No!

Decomposition

The accepted way to eliminate anomalies is to decompose relations.

Given a relation R(A1, … An) we may decompose it into two relations S(B1, … Bm) and T(C1, … Ck) such that:

$\{A1, \ldots An\} = \{B1, \ldots Bm\} \cup \{C1, \ldots Ck\}$

$S = \pi_{B1, \ldots Bm} R$

$T = \pi_{C1, \ldots Ck} R$

Properties we want from a decomposition:

1. Recoverability from the decomposed relations by a join.

-> lossless join decomposition

2. Preservation of dependencies

When we reconstruct the original relation will it satisfy the original FD's?

Finding All Implied FD's

- Motivation: "normalization," the process where we break a relation schema into two or more schemas.
- Example: *ABCD* with FD's $AB \to C$, $C \to D$, and $D \to A$.
  - Decompose into *ABC*, *AD*. What FD's hold in *ABC* ?
  - Not only $AB \to C$, but also $C \to A$!

Why?

ABCD $a_1b_1cd_1$ $a_2b_2cd_2$

comes from ↑

ABC a_1b_1c a_2b_2c

$d_1=d_2$ because
C -> D

$a_1=a_2$ because
D -> A

Thus, tuples in the projection with equal C's have equal A's;
C -> A.

Basic Idea

1. Start with given FD's and find all *nontrivial* FD's that follow from the given FD's.
   - Nontrivial = right side not contained in the left. (Textbook: page 8)
2. Restrict to those FD's that involve only attributes of the projected schema.

Simple, Exponential Algorithm

1. For each set of attributes X, compute X^+.
2. Add $X \to A$ for all A in $X^+ - X$.
3. However, drop $XY \to A$ whenever we discover $X \to A$.
   - Because $XY \to A$ follows from $X \to A$ in any projection.
4. Finally, use only FD's involving projected attributes.

(See in Textbook, Algorithm 3.12: Projecting a set of FDs, page 16, Example 3.13)

A Few Tricks

- No need to compute the closure of the empty set or of the set of all attributes.
- If we find X^+ = all attributes, so is the closure of any superset of X.

Example: Projecting FD's

- ABC with FD's $A \to B$ and $B \to C$. Project onto AC.
  - $A^+ = ABC$; yields $A \to B$, $A \to C$.
    - We do not need to compute AB^+ or AC^+.
  - $B^+ = BC$; yields $B \to C$.
  - $C^+ = C$; yields nothing.
  - $BC^+ = BC$; yields nothing.

Example -- Continued

- Resulting FD's: $A \rightarrow B$, $A \rightarrow C$, and $B \rightarrow C$.
- Projection onto AC: $A \rightarrow C$.
  - Only FD that involves a subset of $\{A, C\}$.

A Geometric View of FD's

- Imagine the set of all *instances* of a particular relation.
- That is, all finite sets of tuples that have the proper number of components.
- Each instance is a point in this space.

Example: R(A,B)

{(1,2), (3,4)}

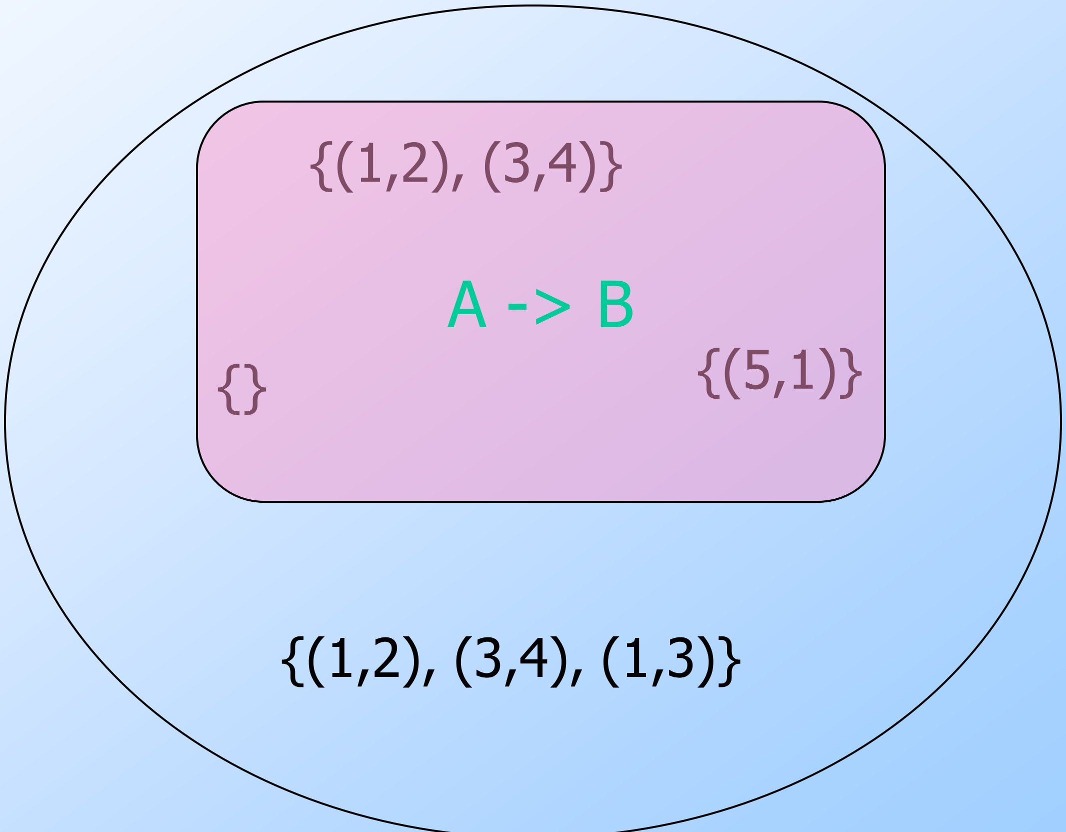
{} {(5,1)}

{(1,2), (3,4), (1,3)}

An FD is a Subset of Instances

- For each FD X -> A there is a subset of all instances that satisfy the FD.
- We can represent an FD by a region in the space.
- Trivial FD = an FD that is represented by the entire space.
  - Example: A -> A.

Example: A -> B for R(A,B)

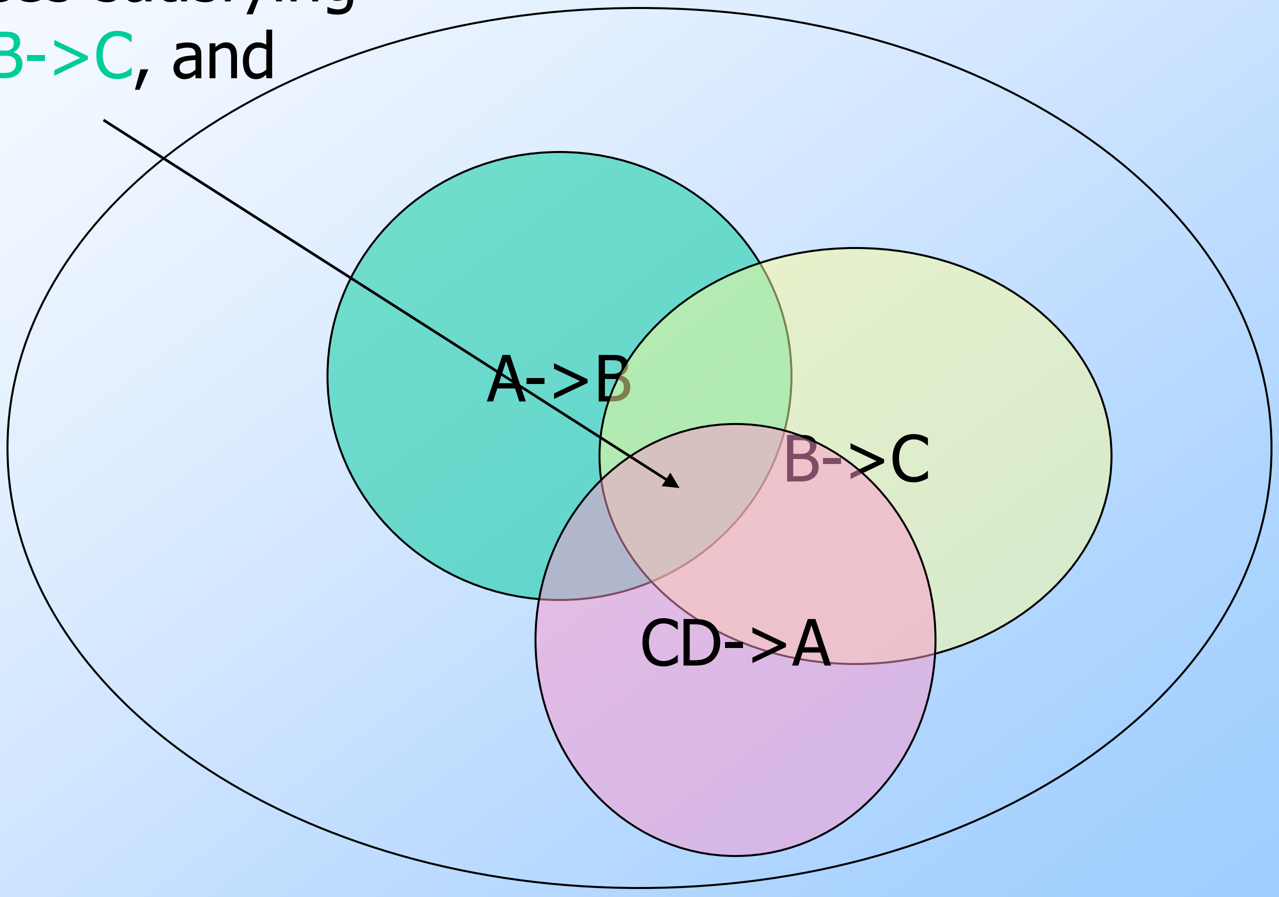


Representing Sets of FD's

- If each FD is a set of relation instances, then a collection of FD's corresponds to the intersection of those sets.
  - Intersection = all instances that satisfy all of the FD's.

Example

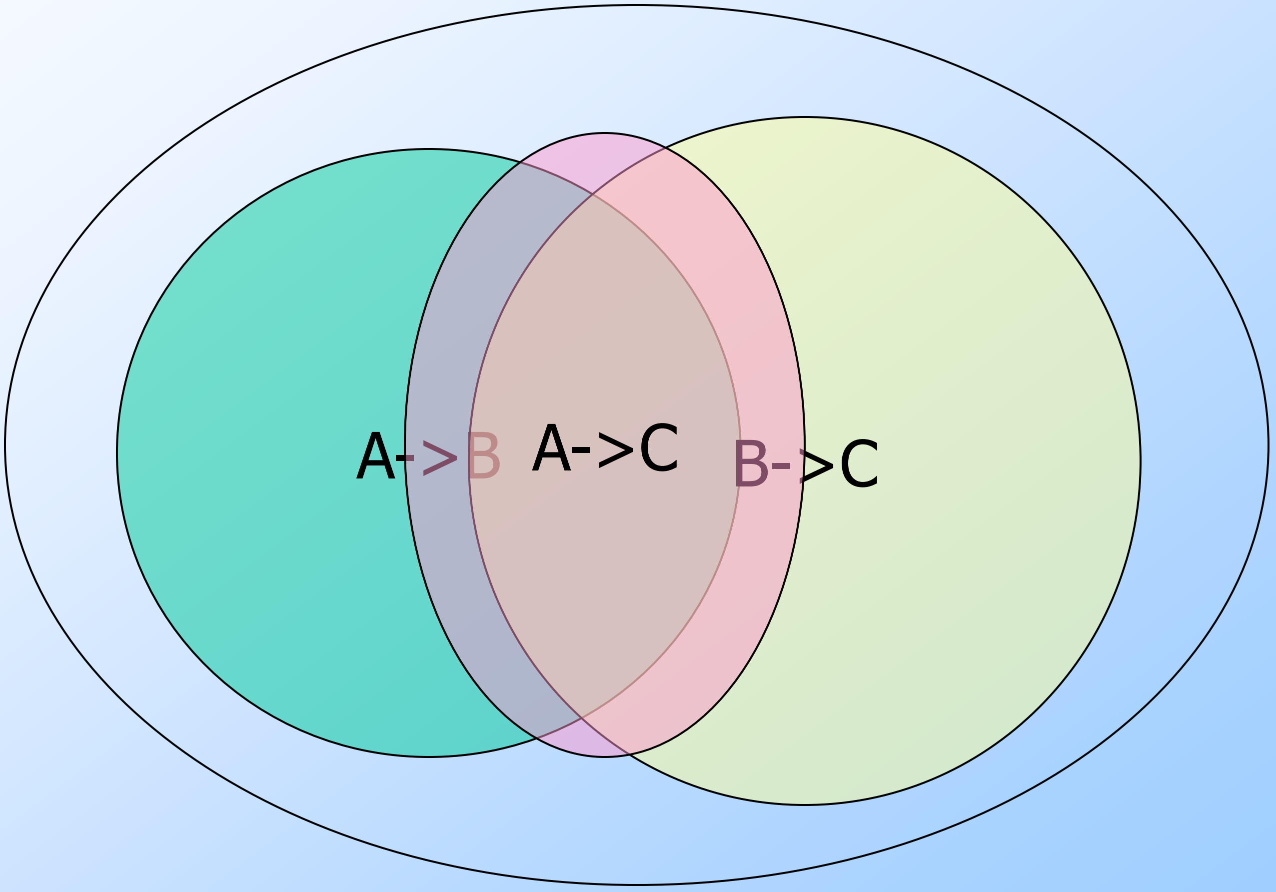
Instances satisfying
A->B, B->C, and
CD->A



Implication of FD's

- If an FD Y -> B follows from FD's X_1 -> A_1,…,X_n -> A_n , then the region in the space of instances for Y -> B must include the intersection of the regions for the FD's X_i-> A_i .
  - That is, every instance satisfying all the FD's X_i-> A_i surely satisfies Y -> B.
  - But an instance could satisfy Y -> B, yet not be in this intersection.

Example



Relational Schema Design

- Goal of relational schema design is to avoid anomalies and redundancy.
  - *Update anomaly* : one occurrence of a fact is changed, but not all occurrences.
  - *Deletion anomaly* : valid fact is lost when a tuple is deleted.

Example of Bad Design

Drinkers(name, addr, beersLiked, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because each of the ???'s can be figured out by using the FD's name -> addr favBeer and beersLiked -> manf.

This Bad Design Also Exhibits Anomalies

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to Addr2, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Boyce-Codd Normal Form

- We say a relation *R* is in *BCNF* if whenever *X* -> *Y* is a nontrivial FD that holds in *R*, *X* is a superkey.
  - Remember: *nontrivial* means *Y* is not contained in *X*.
  - Remember, a *superkey* is any superset of a key (not necessarily a proper superset).

Example

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

FD's: name->addr favBeer, beersLiked->manf

- Only key is {name, beersLiked}.
- In each FD, the left side is *not* a superkey.
- Any one of these FD's shows *Drinkers* is not in BCNF

Another Example

Beers(<u>name</u>, manf, manfAddr)

FD's: name->manf, manf->manfAddr

- Only key is {name} .
- name->manf does not violate BCNF, but manf->manfAddr does.

Decomposition into BCNF

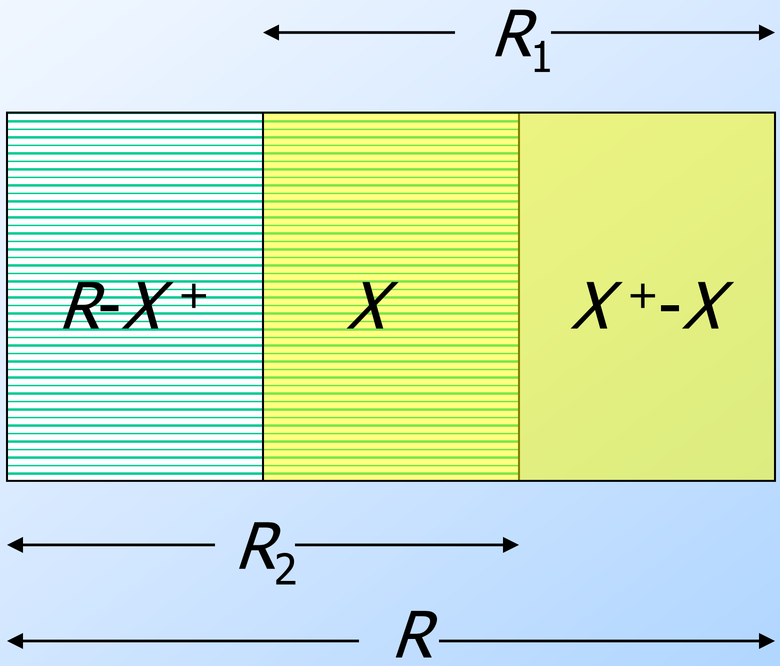
- Given: relation *R* with FD's *F*.
- Look among the given FD's for a BCNF violation $X \rightarrow Y$.
  - If any FD following from *F* violates BCNF, then there will surely be an FD in *F* itself that violates BCNF.
- Compute X^+.
  - Not all attributes, or else X is a superkey.

Decompose R Using X -> Y

- Replace R by relations with schemas:
  1. $R_1 = X^+$.
  2. $R_2 = R - (X^+ - X)$.
- *Project* given FD's F onto the two new relations.

(See in Textbook, Algorithm 3.20 BCNF Decomposition [page 26]; Exercise 3.3.1)

Decomposition Picture



Example: BCNF Decomposition

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

F = name->addr, name -> favBeer, beersLiked->manf

- Pick BCNF violation name->addr.
- Close the left side: $\{name\}^+ = \{name, addr, favBeer\}$.
- Decomposed relations:
  1. Drinkers1(<u>name</u>, addr, favBeer)
  2. Drinkers2(<u>name</u>, <u>beersLiked</u>, manf)

Example -- Continued

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.
- Projecting FD's is easy here.
- For Drinkers1(name, addr, favBeer), relevant FD's are name->addr and name->favBeer.
  - Thus, {name} is the only key and Drinkers1 is in BCNF.

Example -- Continued

- For Drinkers2(<u>name</u>, <u>beersLiked</u>, manf), the only FD is beersLiked->manf, and the only key is {name, beersLiked}.
  - Violation of BCNF.
- beersLiked^+ = {beersLiked, manf}, so we decompose *Drinkers2* into:
  1. Drinkers3(<u>beersLiked</u>, manf)
  2. Drinkers4(<u>name</u>, <u>beersLiked</u>)

Example -- Concluded

- The resulting decomposition of *Drinkers* :
  1. Drinkers1(<u>name</u>, addr, favBeer)
  2. Drinkers3(<u>beersLiked</u>, manf)
  3. Drinkers4(<u>name</u>, <u>beersLiked</u>)
- Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.

Third Normal Form -- Motivation

- There is one structure of FD's that causes trouble when we decompose.
- $AB \rightarrow C$ and $C \rightarrow B$.
  - Example: A = street address, B = city, C = zip code. (not A->C)
- There are two keys, $\{A,B\}$ and $\{A,C\}$.
- $C \rightarrow B$ is a BCNF violation, so we must decompose into AC, BC.

We Cannot Enforce FD's

- The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD *AB* -> *C* by checking FD's in these decomposed relations.
  (The decomposition is not dependency preserving)
- Example with *A* = street, *B* = city, and *C* = zip on the next slide.

An Unenforceable FD

street (A)	zip (C)
545 Tech Sq.	02138
545 Tech Sq.	02139

city (B)	zip (C)
Cambridge	02138
Cambridge	02139

Join tuples with equal zip codes.

street (A)	city (B)	zip (C)
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

Although no FD's were violated in the decomposed relations, FD street city -> zip is violated by the database as a whole.

3NF Let's Us Avoid This Problem

- 3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.
- An attribute is *prime* if it is a member of any key.
- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime.

Third Normal Form

- We say a relation R is in 3rd Normal Form (3NF) if

  whenever X -> Y is a nontrivial FD that holds in R, X is a superkey

  or the attributes in Y (and not in X) are prime.

Example: 3NF

- In our problem situation with FD's *AB->C* and *C->B*, we have keys *AB* and *AC*.
- Thus *A*, *B*, and *C* are each prime.
- Although *C->B* violates BCNF, it does not violate 3NF.

What 3NF and BCNF Give You

- There are two important properties of a decomposition:
  1. *Lossless Join* : it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original.
     (See in Textbook Exercise 3.3.4 [page 27])
  2. *Dependency Preservation* : it should be possible to check in the projected relations whether all the given FD's are satisfied.
     (See in Textbook Example 3.25 [page 34])

3NF and BCNF -- Continued

- We can get (1) with a BCNF decomposition.
- We can get both (1) and (2) with a 3NF decomposition.
- But we can't always get (1) and (2) with a BCNF decomposition.
  - street-city-zip is an example.

Testing for a Lossless Join

Exercise 3.3.4

Suppose we have a relation schema R(A,B,C) with FD A->B.

Suppose also that we decide to decompose this schema into S(A,B) and T(B,C).

Give an example of an instance of relation R whose projection onto S and T and subsequent rejoining does not yield the same relation instance.

(So the decomposition is not lossless.)

Solution: R={t1, t2}

where t1=(a,b,c) and t2=(d,b,e)

What if the FD is B->C ?

Testing for a Lossless Join

- If we project R onto $R_1, R_2, \ldots, R_k$, can we recover R by rejoining?
- Any tuple in R can be recovered from its projected fragments.
- So the only question is: when we rejoin, do we ever get back something we didn't have originally?

The Chase Test

- Suppose tuple *t* comes back in the join.
- Then *t* is the join of projections of some tuples of *R*, one for each R_i of the decomposition.
- Can we use the given FD's to show that one of these tuples must be *t* ?

(See in Textbook, Example 3.22, Example 3.23, Example 3.24 [pages 31-33])

The Chase – (2)

- Start by assuming $t = abc\ldots$.
- For each i, there is a tuple s_i of R that has a, b, c,… in the attributes of R_i.
- s_i can have any values in other attributes.
- We'll use the same letter as in t, but with a subscript, for these components.

Example: The Chase

- Let $R = ABCD$, and the decomposition be AB, BC, and CD.
- Let the given FD's be $C \to D$ and $B \to A$.
- Suppose the tuple t = abcd is the join of tuples projected onto AB, BC, CD.

The *Tableau*

The tuples of R projected onto AB, BC, CD.

A	B	C	D
a	b	c_1	d_1
$\cancel{a_2}$ a	b	c	$\cancel{d_2}$ d
a_3	b_3	c	d

Use $B \to A$

Use $C \to D$

We've proved the second tuple must be *t*.

Summary of the Chase

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless).
4. Otherwise, the final tableau is a counterexample.

Example: Lossy Join

- Same relation $R = ABCD$ and same decomposition.
- But with only the FD $C\text{->}D$.

The *Tableau*

These projections rejoin to form *abcd*.

A	B	C	D
$\boxed{a \quad b}$		c_1	d_1
a_2	$\boxed{b \quad c}$		$\cancel{d_2}$ d
a_3	b_3	$\boxed{c \quad d}$	

Use $C\text{->}D$

These three tuples are an example *R* that shows the join lossy. *abcd* is not in *R*, but we can project and rejoin to get *abcd*.

3NF Synthesis Algorithm

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- Need *minimal basis* for the FD's:
  1. Right sides are single attributes.
  2. No FD can be removed.
  3. No attribute can be removed from a left side.

(See in Textbook chapter 3.2.7 and Example 3.11 [page 14])

Constructing a Minimal Basis

1. Split right sides.
2. Repeatedly try to remove an FD and see if the remaining FD’s are equivalent to the original.
3. Repeatedly try to remove an attribute from a left side and see if the resulting FD’s are equivalent to the original.

3NF Synthesis – (2)

- One relation for each FD in the minimal basis.
  - Schema is the union of the left and right sides.
- If no key is contained in an FD, then add one relation whose schema is some key. (See in Textbook, Algorithm 3.26: 3NF decomposition)

Example: 3NF Synthesis

- Relation R = ABCD.
- FD's *A->B* and *A->C*.
- Decomposition: AB and AC from the FD's, plus AD for a key.

(See in Textbook: Example 3.27)

Why It Works

- Preserves dependencies: each FD from a minimal basis is contained in a relation, thus preserved.
- Lossless Join: use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables.
- 3NF: hard part – a property of minimal bases.