Design Theory for Relational Databases

FUNCTIONAL DEPENDENCIES

DECOMPOSITIONS

NORMAL FORMS

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: no set formers in sets of attributes, just *ABC*, rather than {*A*,*B*,*C*}.

Splitting Right Sides of FD's

$X \rightarrow A_1A_2 \ldots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$,…, $X \rightarrow A_n$ hold for R.

Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.

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

DB Administrator Task

Assume we have a table/view (e.g., with ..., postal code, city, province, country, date, year, month, day)

- It is denormalized (Data Warehouse)
- e.g., a materialized view (joining: sales, location and date/time table)

Describe what are the reasonable FDs to assert in that table/view

Keys of Relations

Set of attributes K is a *superkey* for relation *R* if *K* functionally determines all attributes of *R*.

K is a *key* for *R* if *K* is a superkey, but no proper subset of *K* is a superkey.

Example: Superkey

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

Is {name, beersLiked} a superkey?

Example: Superkey

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

{name, beersLiked} is a superkey because together these attributes determine all the other attributes.

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

Example: Key

Is {name, beersLiked} a key?

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 that contains {name, beersLiked}.

Where Do Keys Come From?

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

More FD's From "Physics"

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

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.

Important for design of good relation schemas.

Inference Test

To test if *Y -> B*, start by assuming two tuples agree in all attributes of *Y*.

Y

00000 0 0 . . . 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.

Example/Task

- Assume *ABC -> D, D -> F, F -> GH, I -> J*.
- Is it true *ABC -> G?*
- Is it true *ABC -> GH?*
- Is it true *ABC -> GHI?*
- Use inference test to provide justification to your answer.

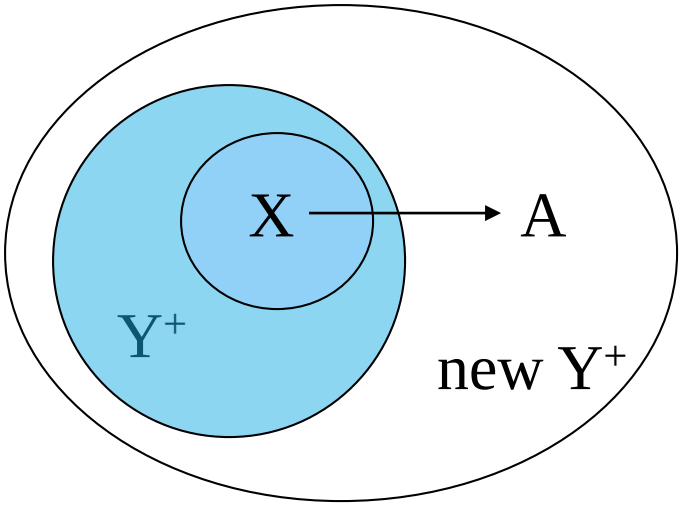
Closure Test

An easier way to test is to compute the *closure* of Y, denoted Y^+.

Closure Y^+ is the set of attributes that Y functionally determines.

Basis: $Y^+ = Y$.

Induction: Look for an FD's left side X that is a subset of the current Y^+. If the FD is $X \rightarrow A$, add A to Y^+.


X
A
Y+
new Y+

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC $^{+}$?*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC $^{+}$?*
  1. Basis *ABC* $^{+}$ = *ABC*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC $^+$?*
  1. Basis *ABC* $^+$ = *ABC*
  2. *ABC* $^+$ = *ABC* $^+$ union *D*, since ABC is a subset of *ABC* $^+$ and *ABC -> D*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC $^+$?*
  1. Basis *ABC* $^+$ = *ABC*
  2. *ABC* $^+$ = *ABC* $^+$ union *D*, since ABC is a subset of *ABC* $^+$ and *ABC -> D*
  3. *ABC* $^+$ = *ABC* $^+$ union *F*, since ABCD is a subset of *ABC* $^+$ and *ABCD -> F*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC $^+$?*
  1. Basis *ABC $^+$ = ABC*
  2. *ABC $^+$ = ABC $^+$* union *D*, since ABC is a subset of *ABC $^+$* and *ABC -> D*
  3. *ABC $^+$ = ABC $^+$* union *F*, since ABCD is a subset of *ABC $^+$* and *ABCD -> F*
  4. *ABC $^+$ = ABC $^+$* plus *GH*, since *F* is a subset of *ABC $^+$* and *F -> GH*

  Therefore ABC $^+$ = ABCDFGH.

  Hence, for instance, ABC -> H is true but ABC -> I is not

Task

- Assume *AB -> CH, C -> D, HD -> A, ACH -> EFG, I -> A*.
- *What is the closure of CH, CH $^{+}$?*

Is it true that CH -> G and CH -> B?.

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 ->C*, *C ->D*, and *D ->A*.

- Assume we decompose into *ABC*, *AD*. What FD's hold in *ABC* ?
- *AB ->C* holds but how about *C ->A*? Perform Inference Test OR Closure Test

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 ->C*, *C ->D*, and *D ->A*.

- Decompose into *ABC*, *AD*. What FD's hold in *ABC* ?
- Not only *AB ->C*, but also *C ->A*? C+ = CDA Yes!

Basic Idea for decomposing table

1. Start with given FD's and find all FD's that follow from the given FD's.
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. Finally, use only FD's involving projected attributes.

A Few Tricks

No need to compute the closure 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* ->*B* and *B* ->*C*. Project onto *AC*.

- $A^+ = ABC$; yields *A* ->*B*, *A* ->*C*.
  - We do not need to compute AB^+ or AC^+ or ABC^+.
- $B^+ = BC$; yields *B* ->*C*.
- $C^+ = C$; yields nothing.
- $BC^+ = BC$; yields nothing.

Example -- Continued

Resulting FD's: *A* ->*B*, *A* ->*C*, and *B* ->*C*.

Projection onto *AC* : *A* ->*C*.

- Only FD that involves a subset of {*A*,*C* }.

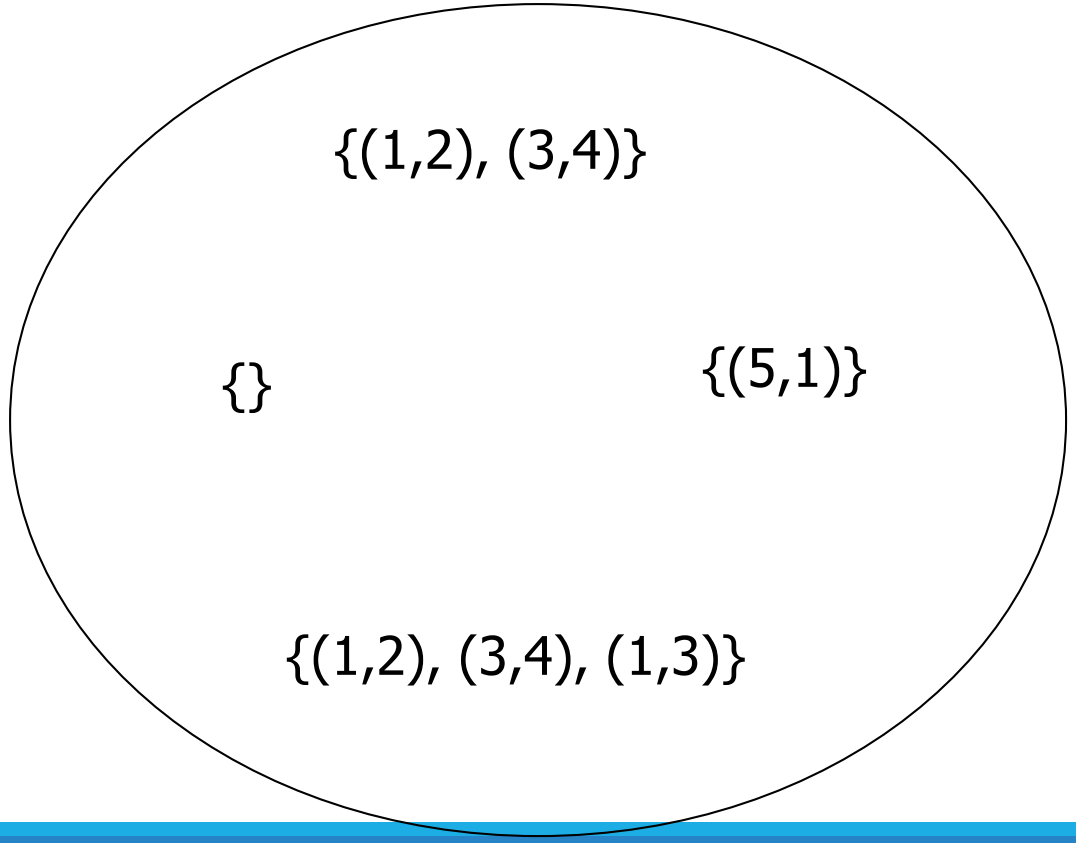
A Geometric View of FD's

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)



An FD is a Subset of Instances

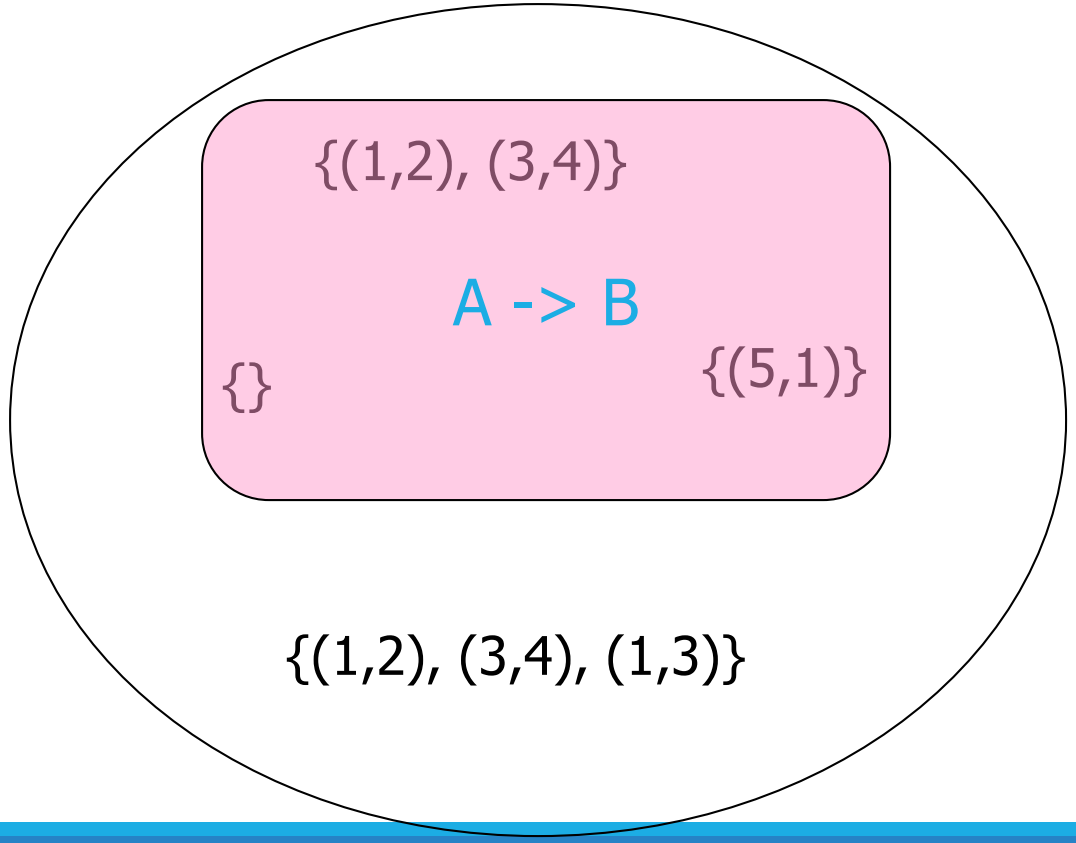
For each FD *X -> A* there is a subset of all instances that satisfy the FD.

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)



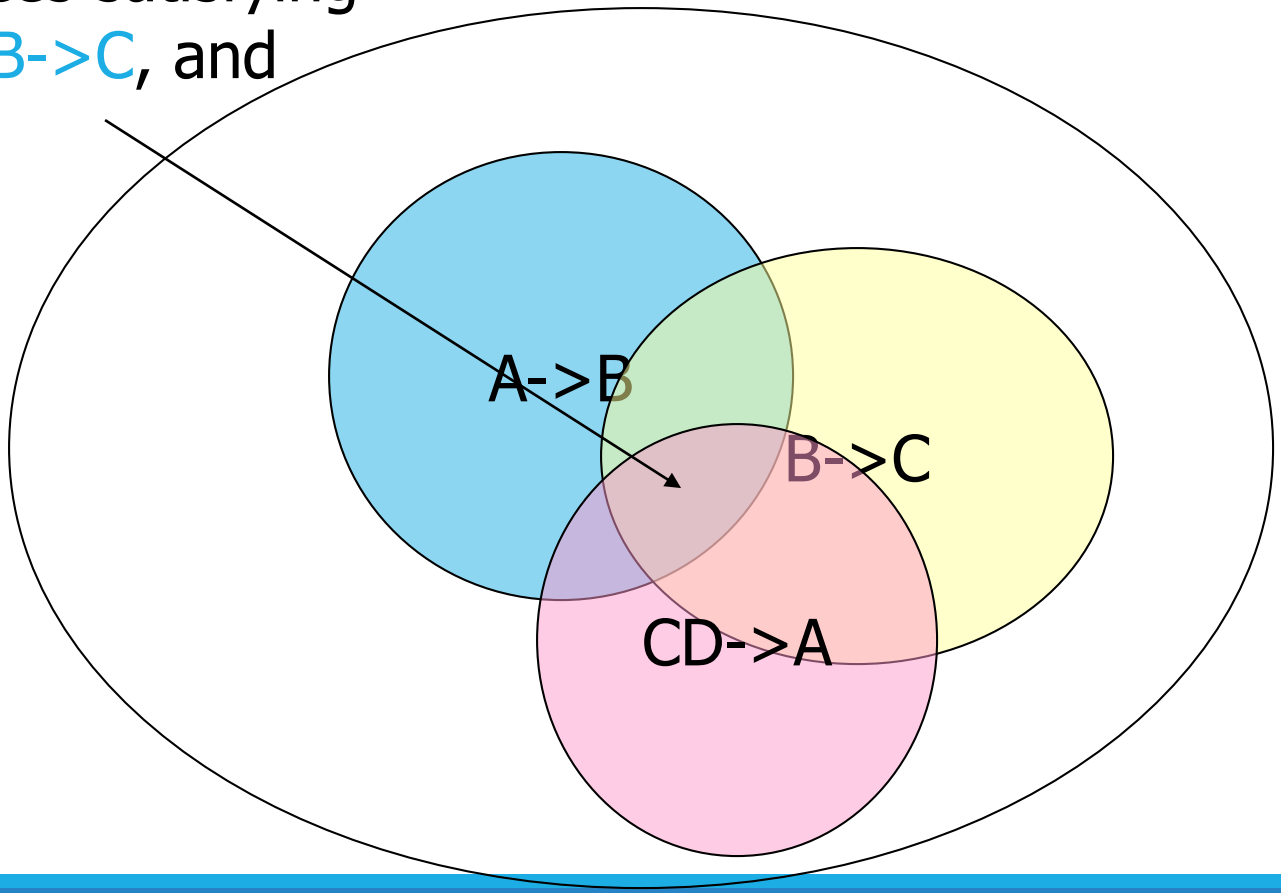
Representing Sets of FD's

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

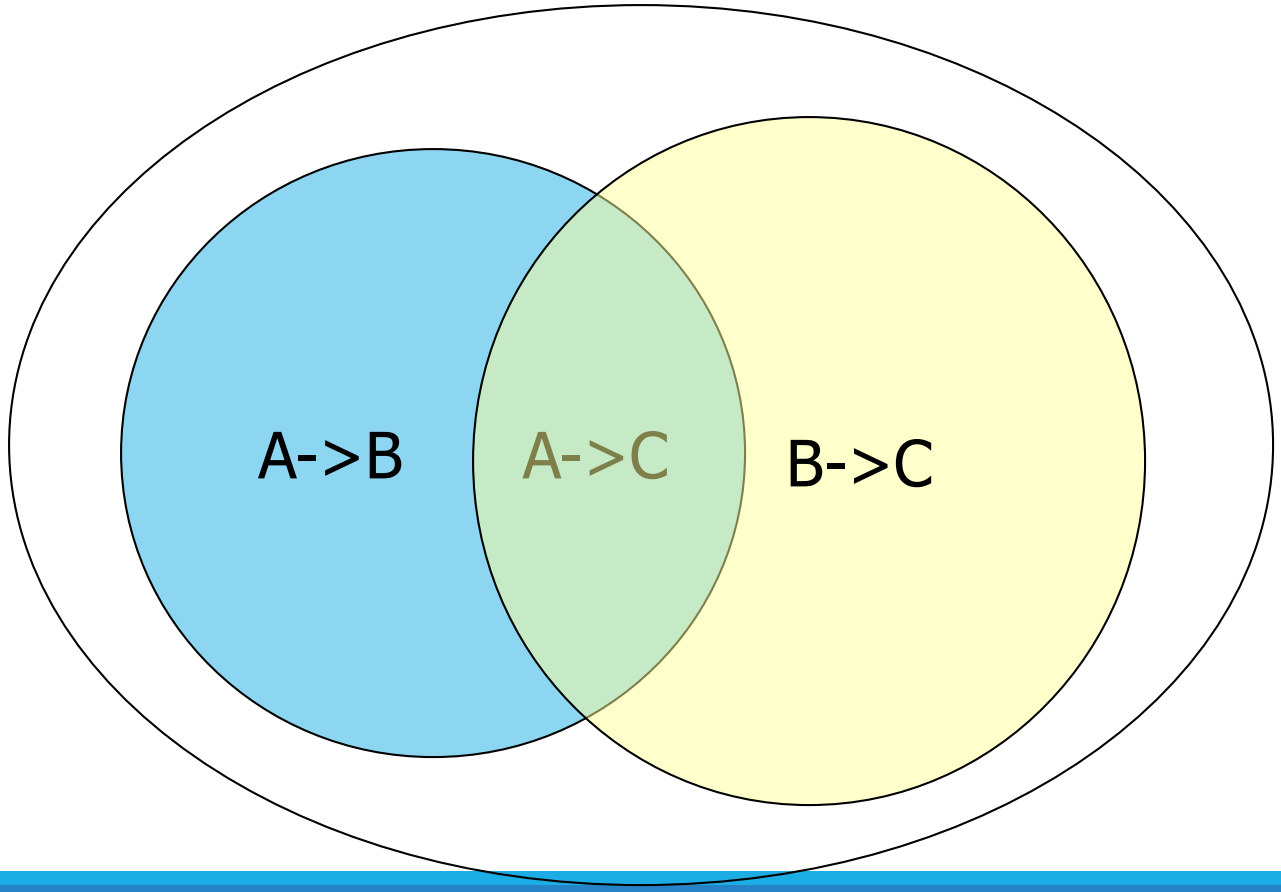


Implication of FD's

If an FD $Y \rightarrow B$ follows from FD's $X_1 \rightarrow A_1,\ldots,X_n \rightarrow A_n$, then the region in the space of instances for $Y \rightarrow B$ must include the intersection of the regions for the FD's $X_i \rightarrow A_i$.

- That is, every instance satisfying all the FD's $X_i \rightarrow A_i$ surely satisfies $Y \rightarrow B$.

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 *Intrepid*, 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*.

- *nontrivial* means *Y* is not contained in *X*.
- Remember, a *superkey* is any superset of a key

Example

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

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

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(name, manf, manfAddr)

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

Only key is {name}.

name->manf does not violate BCNF because name is a superkey, but manf->manfAddr does because manf is not a superkey.

Decomposition into BCNF

Given: relation *R* with FD's *F*.

Look among the given FD's for a BCNF violation: *X* ->*Y*.

- Compute X^+

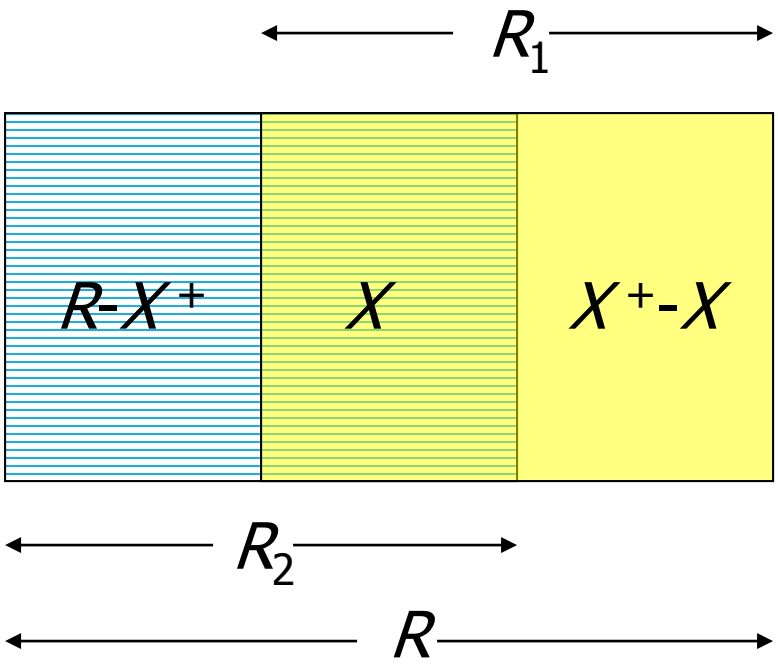
Decompose R Using X -> Y

Replace R by relations with schemas:

1. $R_1 = X^+$.
2. $R_2 = R - (X^+ - X) = R - X^+ + X$.

Project given FD's F onto the two new relations.

Decomposition Picture



Example: BCNF Decomposition

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

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

Pick BCNF violation name->addr.

Closure of the left side: $\{name\}^+$ = {name, addr, favBeer}.

Decomposed relations:

1. Drinkers1(name, addr, favBeer)
2. Drinkers2(name, beersLiked, 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, Drinkers1 is in BCNF.

Example -- Continued

For Drinkers2(name, beersLiked, manf), one of the FDs is beersLiked->manf but beersLiked is not a superkey.

- Violation of BCNF.

beersLiked$^+$ = {beersLiked, manf}, so we decompose *Drinkers2* into:

1. Drinkers3(beersLiked, manf)
2. Drinkers4(name, beersLiked)

Example -- Concluded

The resulting decomposition of *Drinkers* :

1. Drinkers1(name, addr, favBeer)
2. Drinkers3(beersLiked, manf)
3. Drinkers4(name, beersLiked)

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 ->C and *C ->B*.

- Example: *A* = street address, *B* = city, *C* = zip code.

There are two keys, {*A*,*B*} and {*A*,*C*}.

C ->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.

Example with *A* = street, *B* = city, and *C* = zip on the next slide.

An Unenforceable FD

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

city	zip
Cambridge	02138
Cambridge	02139

Join tuples with equal zip codes.

street	city	zip
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 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 ->*A* violates 3NF if and only if *X* is not a superkey, and also *A* is not 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.
2. *Dependency Preservation* : it should be possible to check in the projected relations whether all the given FD's are satisfied.

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.

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.

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.

Example: 3NF Synthesis

Relation R = ABCD.

FD's *A->BC* is equivalent to:

FD's *A->B* and *A->C*.

Decomposition: AB and AC from the FD's, plus AD for a key.

Why It Works

Preserves dependencies: each FD from a minimal basis is contained in a relation, thus preserved.

Lossless Join: yes

3NF: yes.

Actions

Review slides!

Read Chapters 3.1. – 3.5 (Design Theory for Relational Databases).