Database Management Systems - I, CS 157A

Physical RDBMS Model: Schema Design and Normalization

Ch. 3

Functional Dependencies

- Functional Dependencies (FD) are used in normalizing relations by decomposing a relation into two or more to remove certain dependencies → idea of a relation key

- FD on a relation R states that "if any two tuples in R agree (have same values) on the attributes $X_1, X_2, .., X_n$ then they also agree on the attributes $Y_1, Y_2, \ldots, Y_m$

- The above FD is expressed as $X_1, X_2, .., X_n \rightarrow Y_1, Y_2, \ldots, Y_m$

- Another representation is $X \rightarrow Y$

Splitting Right Sides of FD's

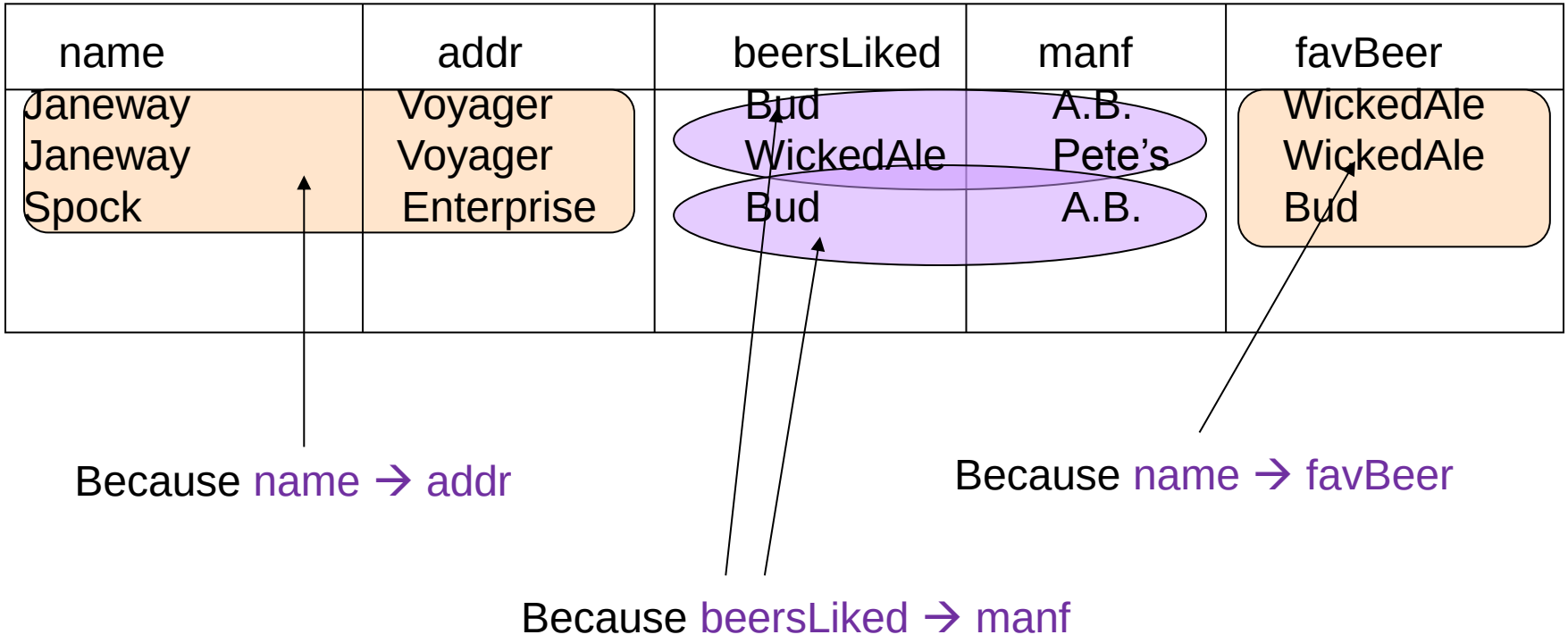
- The above FD ($X \rightarrow Y$) is equivalent to the following set of FD's:
  - $X_1, X_2, .., X_n \rightarrow Y_1$
  - $X_1, X_2, .., X_n \rightarrow Y_2$
  - …
  - $X_1, X_2, .., X_n \rightarrow Y_m$
- There is no splitting rule for left sides.
- We'll generally express FD's with singleton right 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



Keys of Relations

- Set of attributes are called key (K) iff:
  - This set of attributes functionally determines all other attributes. No two tuples will agree on all attributes including K
  - K must be minimal, and K determines all other attributes of R

- *Set of attributes that contain* K *is called superkey* for relation R. Superkey satisfies the first condition but not necessarily the second (minimality)

- K is a superkey but no subset of K is a superkey

Example: Superkey

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

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

Example: Key

- {name, beersLiked} is a key because it satisfies minimality; neither {name} nor {beersLiked} is a superkey.
  - name doesn't → manf and beersLiked doesn't → addr
- There are no other keys, but lots of superkeys.
  - Any superset of {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 $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$,…, $X_n \rightarrow A_n$, and we want to know whether an FD $Y \rightarrow B$ must hold in any relation that satisfies the given FD's.
  - Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ 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 and check if they agree also on attributes of B

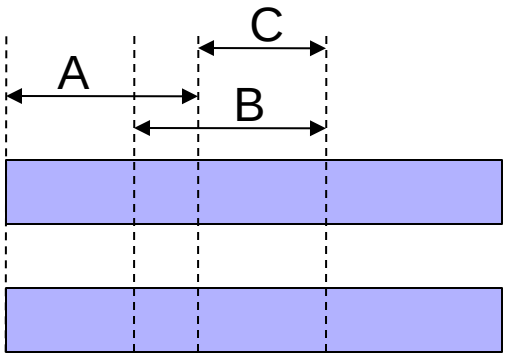
← Y →

0000000. . . 0

00000?? . . .?

Trivial FD

- A constraint on relation R is said to be trivial if it holds for every instance of R regardless of what other constraints are assumed
- For FD A → B is trivial iff B ⊆ A or the right side B is a subset of its left side
- Title year → title and title → title are trivial FD



- The above FD: A → B is equivalent to A → C

Inference Rules

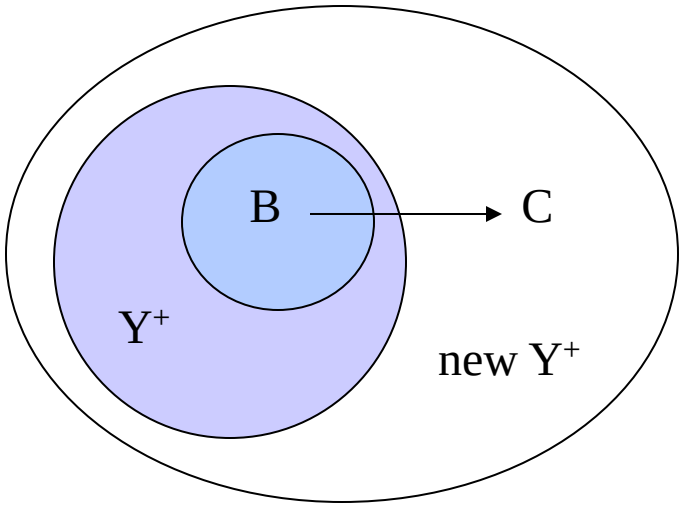
- **Reflexivity**: if B $\subseteq$ A, then A → B is trivial FD
- **Augmentation**: if A → B, then AC → BC
- **Transitivity**: if A → B and B → C then A → C

Computing the Closure of Attributes

- Assume set of attributes Y = $\{Y_1, Y_2, \ldots, Y_n\}$ and a set of FD's S
- Compute the *closure* of Y (set of attributes denoted by Y^+) under S, i.e., $Y \rightarrow Y^+$ follows FD of S and is always superset of Y.

  In other words, Y^+ **is a superset of Y and has all attributes that are determined knowing Y given the set S of FDs** (if Y is a candidate key then Y^+ is a superkey).
- Candidate key are all attributes that are not part of the RHS of our FDs.
- Start with: $Y^+ = Y$
- Induction: Search for a subset of current Y^+ that matches the LHS of one of the FDs ($B_1, B_2, \ldots, B_m \rightarrow C$) then add C to current Y^+
- Repeat till nothing more can be added $\rightarrow Y^+$
- Closure Y^+ is a superkey

Assume **R**(A,B,C,D,E,F)
Assume **FDs**:
A → C
C → D
D → B
E → F



Attributes not in the RHS
of the above FDs is: (AE)

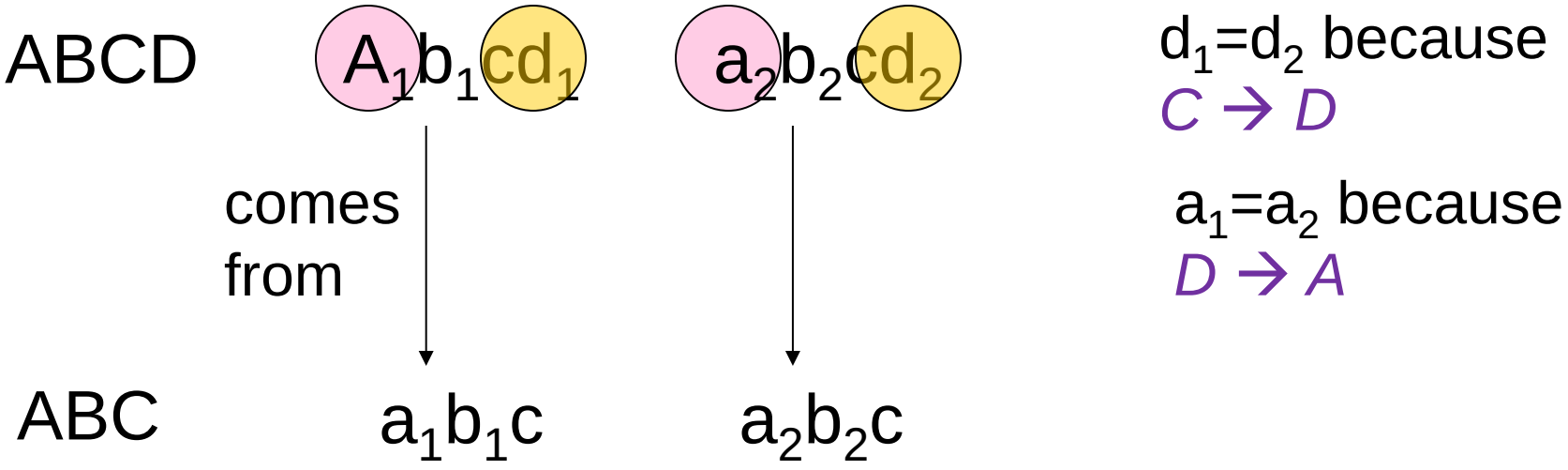
In other words, (AE) is a candidate key.
Also, $(AE)^+$ = (AECDBF) – all attributes for R
This also means that (AE) is the PK for R.

Finding All Implied FD's

- **Motivation**: "normalization," the process where we break a relation schema into two or more schemas.

- **Example**: *ABCD* is a R with FD's *AB* →*C*, *C* →*D*, and *D* →*A (infer C* →*A):*
  - Decompose R into *ABC*, *AD*.
  - What FD's hold in *ABC* ? Not only *AB* →*C*, but also *C* →*A*
  - *What FD's hold in AD? D* →*A*
  - *What happened to C* →*D (later)?*

Why?

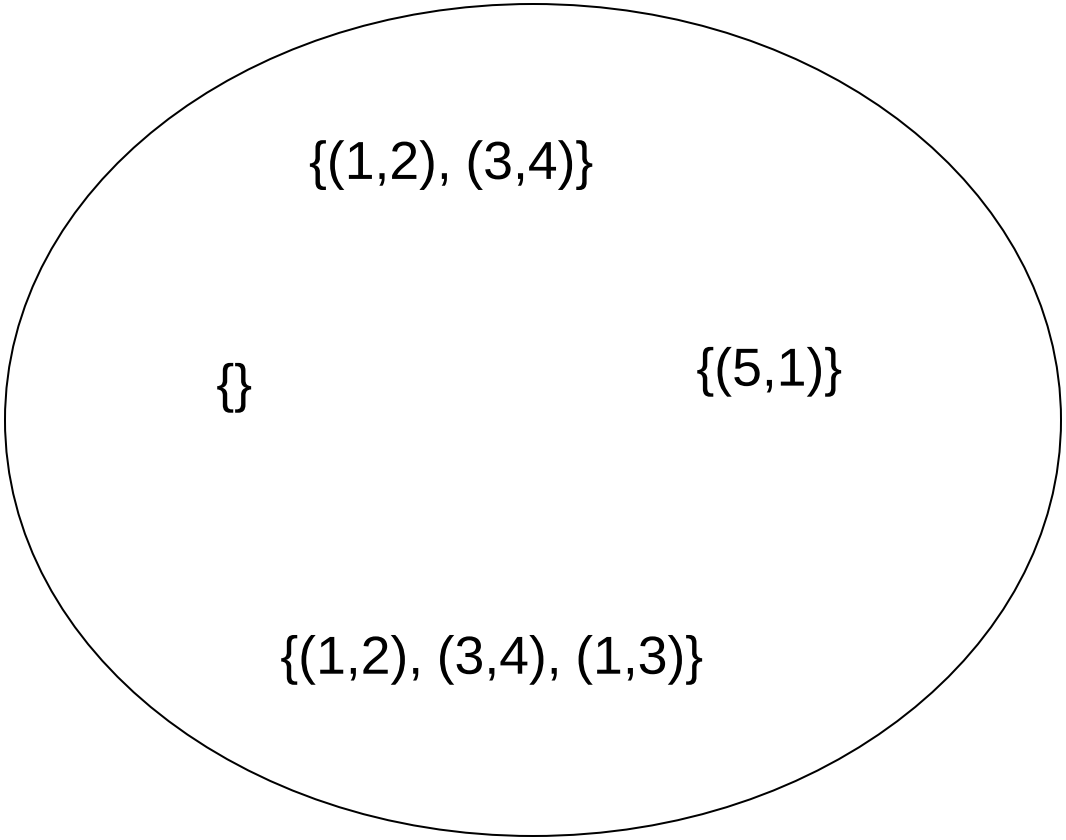


Thus, tuples in the projection with equal C's have equal A's;
$C \rightarrow A$ *(proves inference)*

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

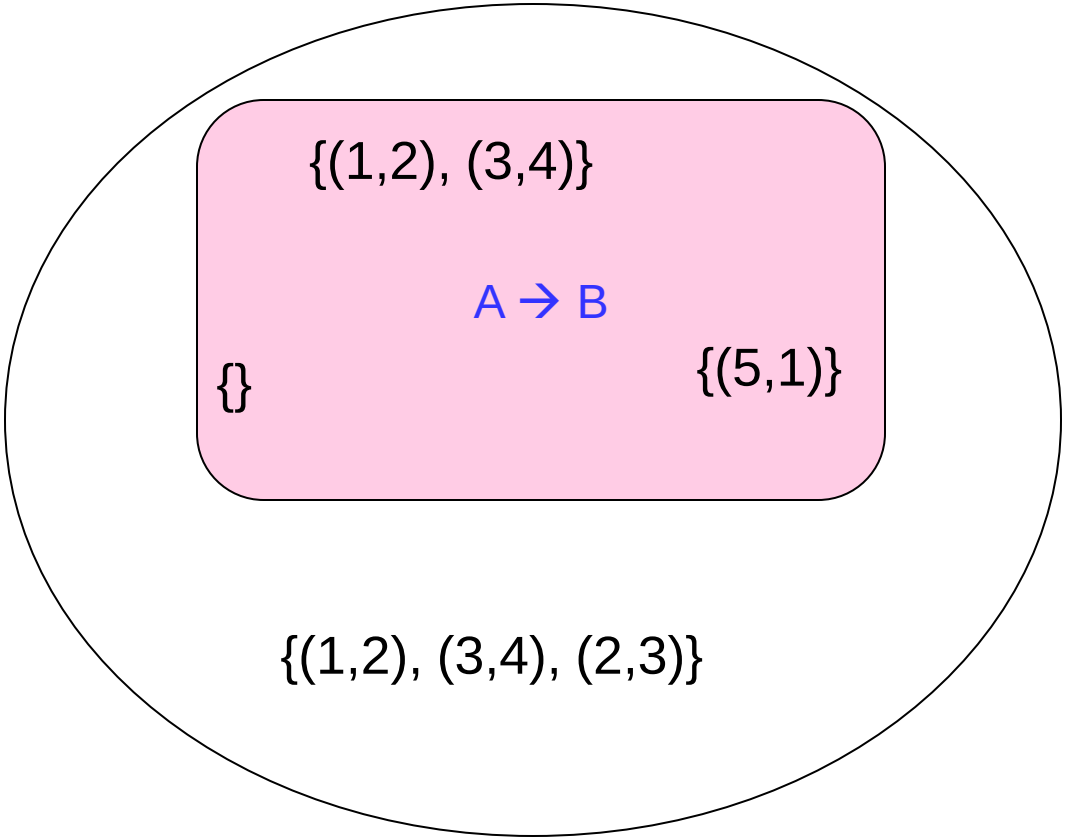
Example: R(A,B)



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 a 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)

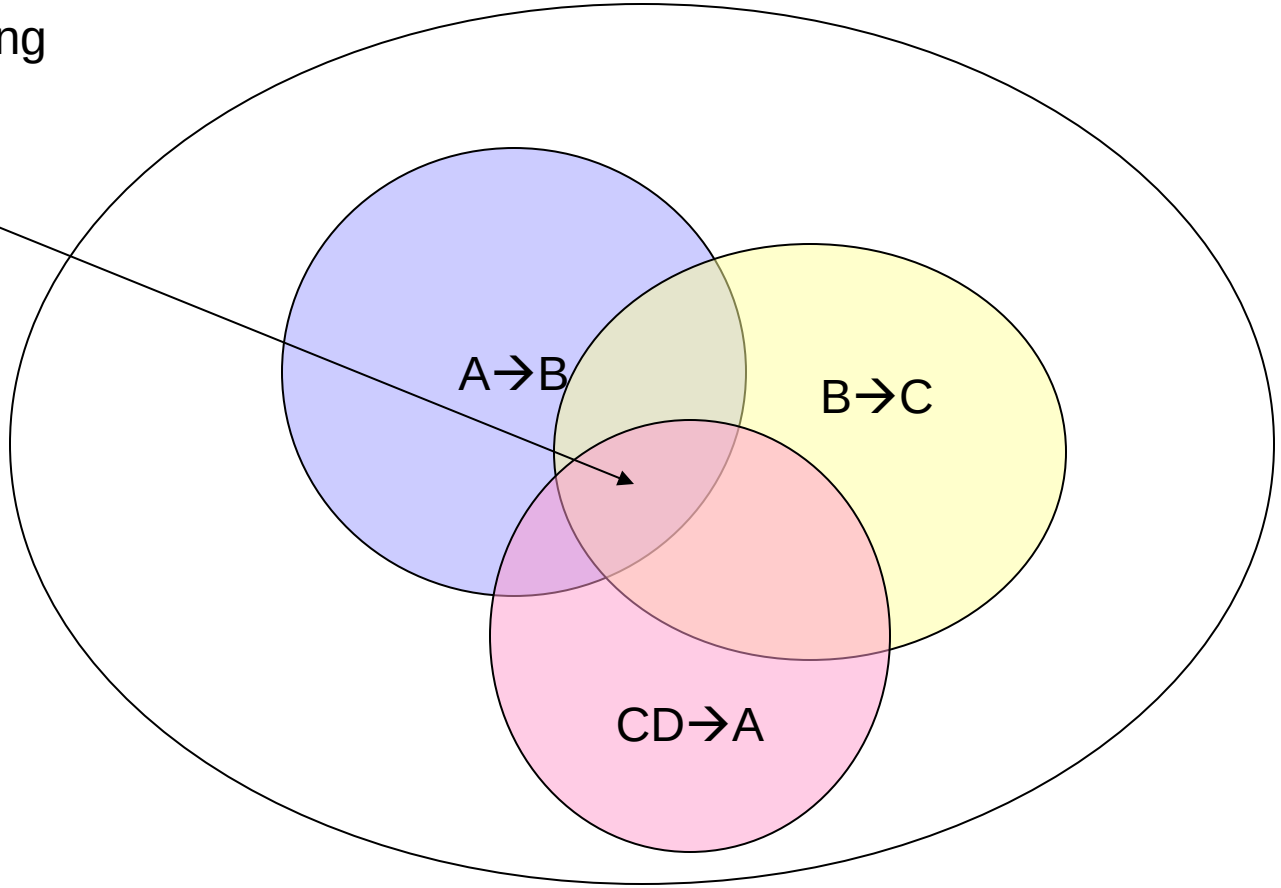


Representing Sets of FD's

- 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

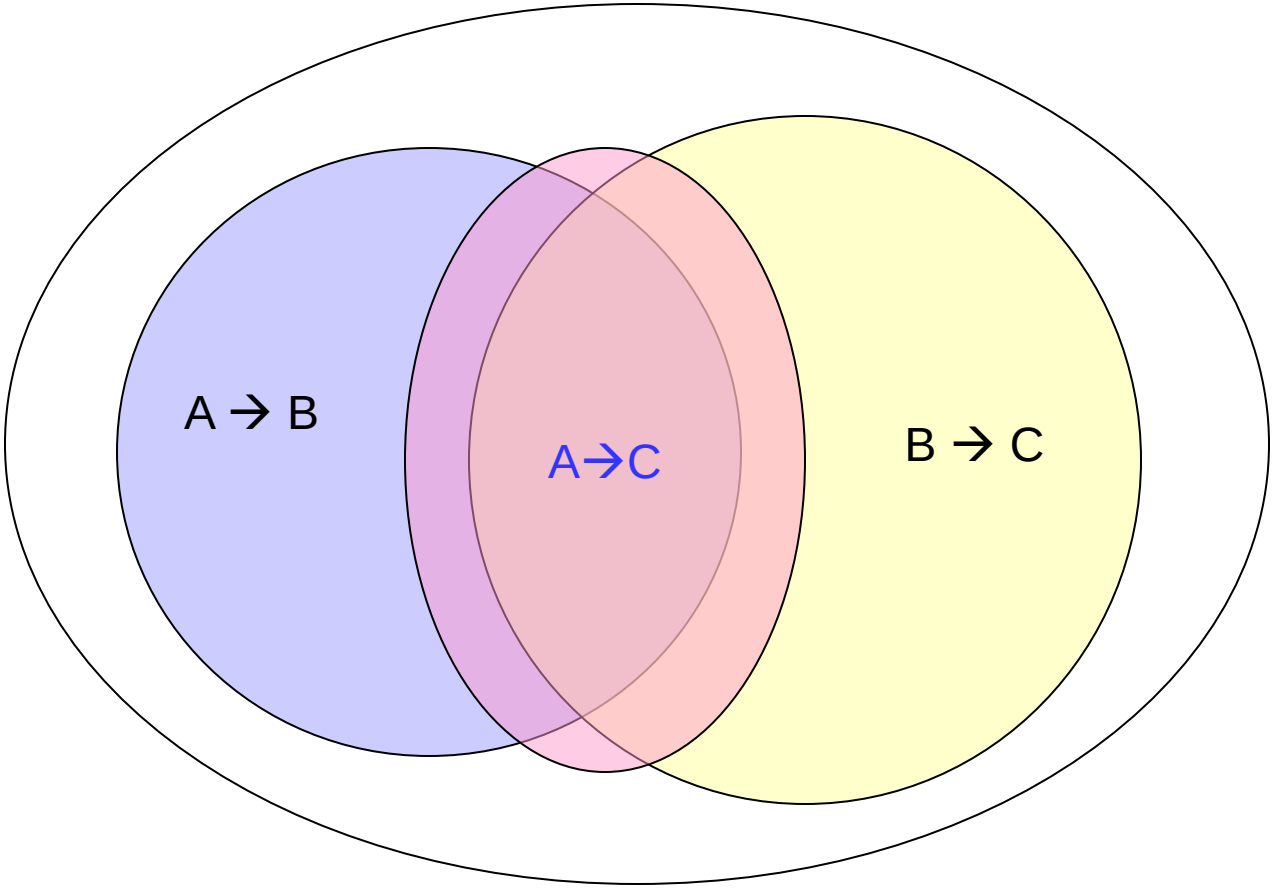
Instances satisfying
A→B, B→C, and
CD→A



Implication of FD's

- If a 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 $Y \rightarrow B$
  - But an instance could satisfy $Y \rightarrow B$, yet not be in this intersection

Example



Relational Schema Design

- **Goal of relational schema design is to avoid anomalies:**
  - ***Redundancy:*** info may be repeated unnecessarily in several tuples.
  - ***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	WickedAle
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

- **Redundancy:** info may be repeated unnecessarily (wasted storage) in several tuples
- **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 - BCNF

- **BCNF** is a condition on a relation schema that eliminates potential possible anomalies.

- We say a relation *R* is in ***BCNF*** if whenever ***X → Y*** is a nontrivial FD that holds in *R*, then *X* is a superkey:
  - **Remember:** *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

- 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 compliance

Another Example

Beers (name, manf, manfAddr)

FD's: name → manf, manf → manfAddr

- Only key is {name}
- Name → manf does not violate BCNF, but manf → manfAddr does as manf is not a superkey

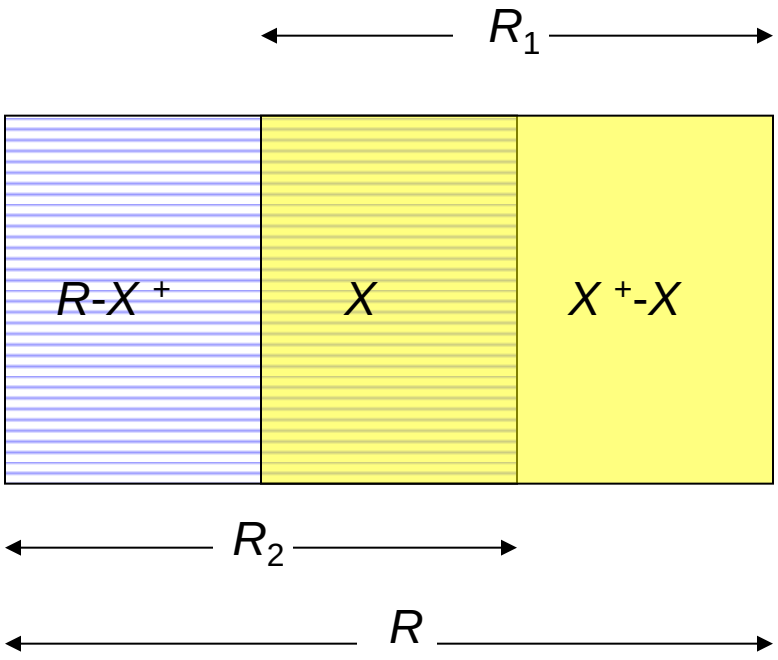
Decomposing Relations

- This involves splitting attributes of R to make the schemas of two or more new relations.
- Given $R(A_1,A_2,\ldots,A_n)$, we may decompose R into $S(B_1,B_2,\ldots,B_m)$ and $T(C_1,C_2,\ldots,C_k)$ such that:
  - $\{A_1,A_2,\ldots,A_n\} = \{B_1,B_2,\ldots,B_m\} \cup \{C_1,C_2,\ldots,C_k\}$
  - $\mathbf{S} = \prod_{B1,B2,\ldots,Bm}(\mathbf{R})$
  - $\mathbf{T} = \prod_{C1,C2,\ldots,Ck}(\mathbf{R})$
- Decomposition should eliminate anomalies if done right.

Decomposition into BCNF

- **Input:** relation R_0 with FD's F_0
- ***Output:*** *decomposition of R_0 into set of relations R, all of which are in BCNF*
- $R = R_0$ & $F = F_0$
- Check if R is in BCNF; done.
- If there are BCNF violations, assume violation is $X \rightarrow Y$
- Compute closure of X: X^+
  - Choose $R_1 = X^+$
  - And $R_2 = R - (X^+ - X)$
  - Project FDs for each new relation
- Recursively decompose R_1 and R_2
- Return the union of all these decompositions

Decomposition Picture



Decomposition: Is It All Good?

- A good decomposition should exhibits the following three properties:
  1. Elimination of anomalies
  2. Recoverability of information: recover original relation from the decomposed relations ← Lossless Join
  3. Preservation of Dependencies: if we joined the decomposed relations to reconstruct the original relation, do we satisfy the original FDs?
- BCNF decomposition gives us (1) and (2) but not always (3)!
- We will discuss later the *third normal form (3NF)* which will give us (2) and (3) but not necessarily (1)
- It is impossible to get all three at once!

Example: BCNF Decomposition

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

FD = name → addr, name → favBeer,
beersLiked → manf

- Pick BCNF violation name → addr
- Closure of the violation left side:
  $\{name\}^+ = \{name, addr, favBeer\}$
- Decomposed relations:
  1. R1 = $(name)^+$
     Drinkers1(name, addr, favBeer)
  2. R2 = (Drinkers) – ($name^+$ - name)
     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, {name} is the only key and Drinkers1 is in BCNF

Example -- Continued

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked → manf, and the only key is {name, beersLiked}
  - 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.
- Assume R(A,B,C): $AB \rightarrow C$ and $C \rightarrow B$
  - Example: *A* = street address, *B* = city, *C* = zip code
- 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

An Unenforceable FD

Example with *A* = street, *B* = city, and *C* = zip:
AB →*C* and *C* →*B*

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	city	zip
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

Although no FD's were violated in the decomposed relations (AC, BC), FD (AB → C) or 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 iff X is not a superkey, and also A is not prime.

- Relation R is in 3NF for nontrivial FD X → A iff: either X is a superkey or A's attributes are each a member of some key.

Example: 3NF

- In our problem situation with FD's $AB \rightarrow C$ and $C \rightarrow B$, we have keys AB and AC
- Thus A, B, and C are each prime
- Although $C \rightarrow B$ violates BCNF (i.e., C is not a superkey), it does not violate 3NF (while C is not a superkey but B is a prime)

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

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*?

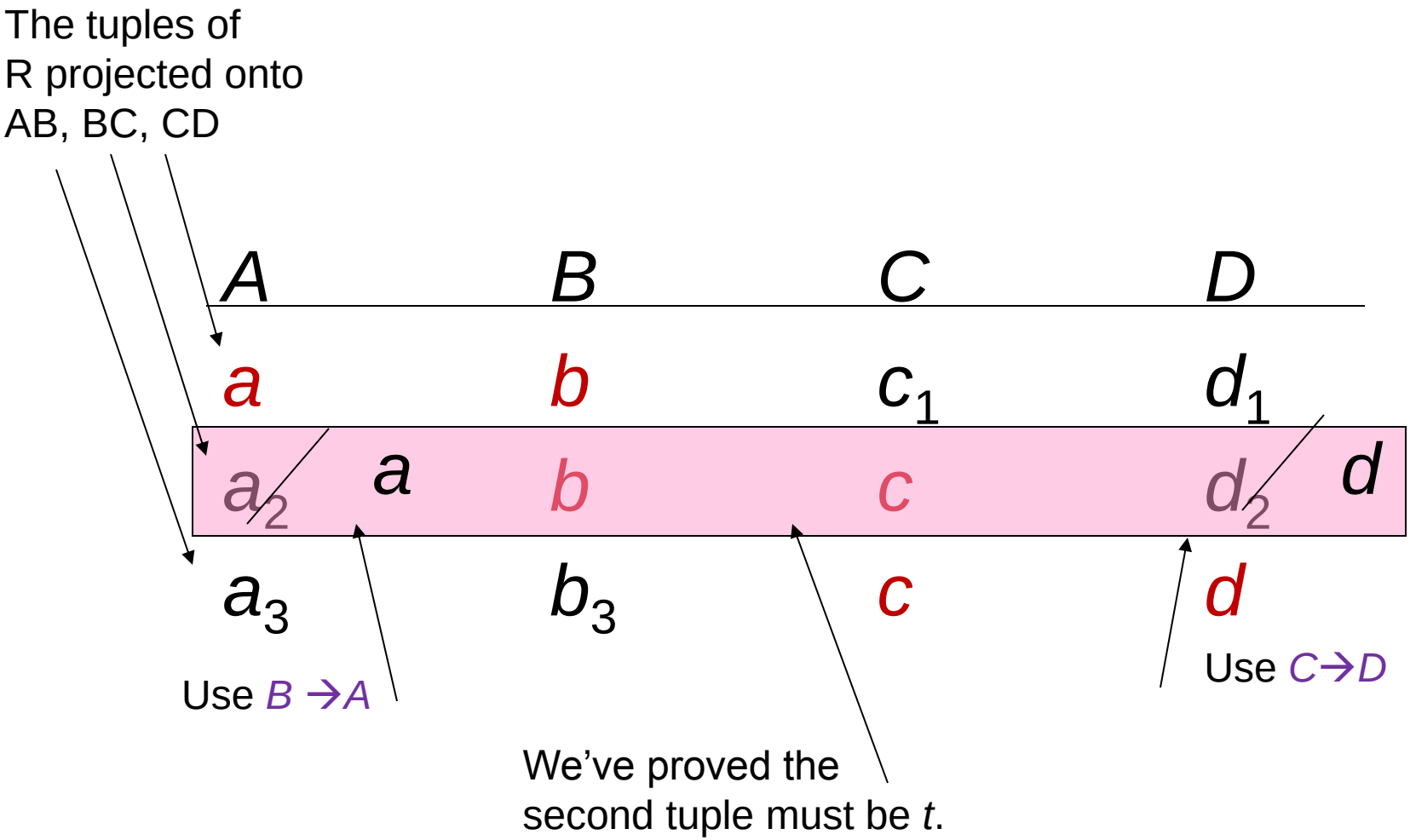
The Chase – (2)

- Start by assuming R has $t = abc\ldots$.
- For each decomposition R_i, there is a tuple s_i of R_i that has the corresponding a, b, c,… in the attributes of R_i *and the rest of the attributes of* s_i *can be of any value.*
- 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* → *D* and *B* → *A*
- Suppose the tuple t = (abcd) is the join of tuples projected onto *AB*, *BC*, *CD*.
- In other words: particular decomposition of R into AB, BC, CD with AB decomposition has tuple (abxx), BC has tuple (xbcx) and CD has tuple (xxcd), and given a set of FDs (C → D and B → A) then we will have a lossless join, i.e., we can reconstruct from the join of the decompositions a tuple in R = (a,b,c,d).

The *Tableau*


The tuples of
R projected onto
AB, BC, CD
A
B
C
D
a
b
c1
d1
a2
a
b
c
d2
d
a3
b3
c
d
Use B →A
We've proved the
second tuple must be t.
Use C→D

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 counter example.

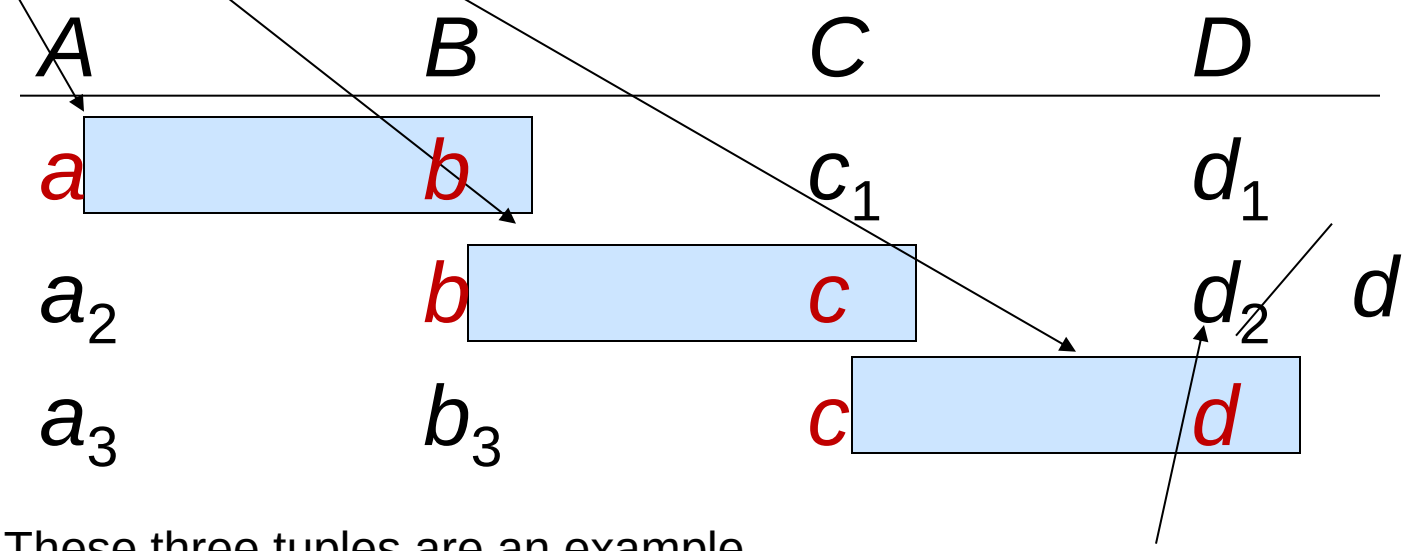
Example: Lossy Join

- Same relation $R = ABCD$ and same decomposition: AB, BC, CD

- But with only the FD $C \rightarrow D$, we will have Lossy Join

The *Tableau*

These projections
AB, BC, CD
rejoin to form *abcd*



These three tuples are an example *R* that shows the join is lossy. *abcd* is not in *R*.

Use *C*→*D*

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 in the decomposition
  3. No attribute can be removed from a left side

Constructing a Minimal Basis

1. Split right sides
2. Repeatedly try to remove a 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 $A \rightarrow B$ and $A \rightarrow C$
- Decomposition: AB and AC from the FD's.
  - A cannot be by itself a key?
  - We add a 3rd table where the schema for that 3rd table is the Key – (AD)
  - Decomposition is: AB, AC, AD with the key AD.

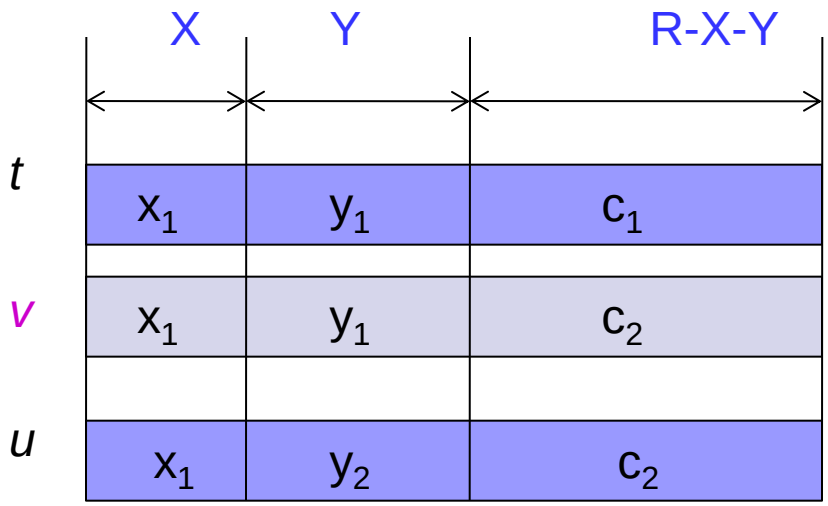
Why It Works

- 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 basis

Definition of MVD ← 4NF

- **Multivalued dependency (MVD)** is an assertion that two attributes or set of attributes (X and Y) are independent of each other; generalization of FD. If we fixed X the Y is different:
  X →→ Y is an MVD for relation R
- The value of an attribute or set of attributes in MVD is called Set-valued properties
- For R and two tuples *t* and *u* that agree on X but not on Y (i.e., X is independent of Y or t and u may agree or not on Y), we can find tuple *v* in R that agrees:
  - With both *t* and *u* on X
  - With *t* on Y, and
  - With *u* on all attributes of R–X-Y

Definition of MVD



X →→ y is an MVD for Relation (R)

- Trivial MVD: {X} →→ {Y} iff {Y} ⊆ {X}
- Transitive rule: {X} →→ {Y} and {Y} →→ {Z} then {X} →→ {Z}
- FD promotion: every FD is an MVD. That is if {X} → {Y} then {X} →→ {Y}
- Complementation rule: if {X} →→ {Y} is an MVD for relation R then R also satisfies {X} →→ R-X-Y

Example: MVD

Drinkers(name, addr, phones, beersLiked)

- A drinker's phones are independent of the beers-they-like:
  - name→→phones and name →→ beersLiked
- Thus, each of a drinker's phones appears with each of the beers-they-like in all combinations.
- This repetition is unlike FD redundancy:
  - Name → addr is the only FD

Tuples Implied by name→→phones

If we have tuples:

name	addr	phones	beersLiked
sue	a	p1	b1
sue	a	p2	b2
sue	a	p2	b1
sue	a	p1	b2

Then these tuples must also be in the relation

Splitting Doesn't Hold

- Like FD's, we cannot generally split the left side of an MVD
- But unlike FD's, we cannot split the right side either --- sometimes you have to leave several attributes on the right side

Example: Multi-attribute Right Sides

Drinkers(name, areaCode, phone, beersLiked, manf)

- A drinker can have several phones, with the number divided between areaCode and phone (last 7 digits)
- A drinker can like several beers, each with its own manufacturer

Example Continued

- Since the areaCode-phone combinations for a drinker are independent of the beersLiked-manf combinations, we expect that the following MVD's hold:

  name →→ areaCode phone

  name →→ beersLiked manf

Example Data

Here is possible data satisfying these MVD's:

name	areaCode	phone	beersLiked	manf
Sue	650	555-1111	Bud	A.B.
Sue	650	555-1111	WickedAle	Pete's
Sue	415	555-9999	Bud	A.B.
Sue	415	555-9999	WickedAle	Pete's

But we cannot split area codes or phones by themselves. That is, neither name→→areaCode nor name→→phone holds for this relation

Fourth Normal Form

- The redundancy that comes from MVD's is not removable by putting the database schema in BCNF

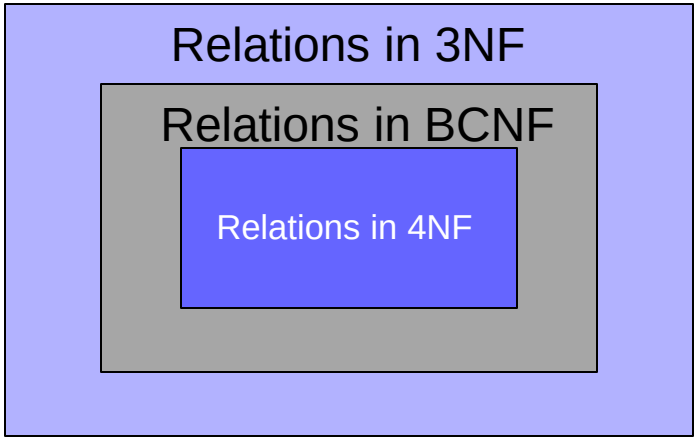
- There is a stronger normal form, called 4NF, that (intuitively) treats MVD's as FD's when it comes to decomposition, but not when determining keys of the relation

4NF Definition

- A relation *R* is in *4NF* if: whenever $X \rightarrow\rightarrow Y$ is a nontrivial MVD, then *X* is a superkey:
  - *Nontrivial MVD* means that:
    1. *Y* is not a subset of *X*, and
    2. *X* and *Y* are not, together, represent all the attributes
  - Note that the definition of "superkey" still depends on FD's only

BCNF Versus 4NF

- Remember that every FD $X \rightarrow Y$ is also an MVD, $X \rightarrow\rightarrow Y$
- Thus, if *R* is in 4NF, it is certainly in BCNF:
  - Because any BCNF violation is a 4NF violation (after conversion to an MVD)
- But *R* could be in BCNF and not 4NF, because MVD's are "invisible" to BCNF



Decomposition and 4NF

- If $X \rightarrow\rightarrow Y$ is a 4NF violation for relation R, we can decompose R using the same technique as for BCNF:
  1. XY is one of the decomposed relations
  2. X U {All but X – Y (i.e., R-X-Y)} is the other i.e., X U (R-X-Y)

- If R has no 4NF violations, return; R by itself is a suitable decomposition

Example: 4NF Decomposition

Drinkers(name, addr, phones, beersLiked)

FD: name → addr

MVD's: name →→ phones

name →→ beersLiked

- Key is {name, phones, beersLiked}
- All dependencies violate 4NF (FD is also an MVD). Left side for all our MVDs (i.e., name) are not a superkey

Example Continued

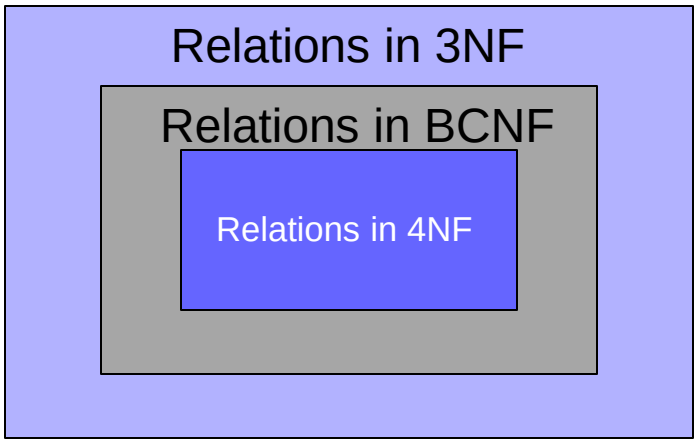
- Decompose using name → addr:

1. **Drinkers1**(name, addr)
   - In 4NF; only dependency is name → addr
2. **Drinkers2**(name, phones, beersLiked)
   - MVD's name →→ phones and name →→ beersLiked are not in 4NF. No FD's, so all three attributes form the key.

Example: Decompose Drinkers2

- Either MVD name →→ phones or name →→ beersLiked tells us to decompose to (given no FDs, all attributes of every MVD forms the Key for a new table):
  - **Drinkers3**(name, phones)
  - **Drinkers4**(name, beersLiked)

Relationships Among Normal Forms



4NF implies BCNF implies 3NF

Property	3NF	BCNF	4NF
Eliminates redundancy due to FD's	No	Yes	Yes
Eliminates redundancy due to MVD's	No	No	Yes
Preserves FD's	Yes	No	No
Preserves MVD's	No	No	No

Reasoning About MVD's + FD's

- **Problem:** given a set of MVD's and/or FD's that hold for a relation *R*, does a certain (inferred) FD or MVD also hold in *R*?

- **Solution:** Use a tableau to explore all inferences from the given set, to see if you can prove the target dependency

Why Do We Care?

1. **4NF technically requires an MVD violation:**
   - Need to infer MVD's from given FD's and MVD's that may not be violations themselves

2. When we decompose, we need to project FD's + MVD's

Example: Chasing a Tableau With MVD's and FD's

- To apply a FD, equate symbols, as before
- To apply an MVD, generate one or both of the tuples we know must also be in the relation represented by the tableau
- We'll prove: if A→→BC and D→C, then A→C
- Proof: start with 2 rows agree on A (left side of FD to be inferred) and disagree on every thing else

The Tableau for A→C

Goal: prove that $c_1 = c_2$.

A	*B*	*C*	*D*
a	b_1	~~c_1~~ c_2	d_1
a	b_2	c_2	d_2
a	b_2	c_2	d_1

Use A→→BC (first row's *D* with second row's *BC*).

Use D→C (first and third row agree on *D*, therefore agree on *C*).

Then you can see A→C.

Example: Transitive Law for MVD's

- If A→→B and B→→C, then A→→C:
  - Obvious from the complementation rule if the Schema is *ABC*.
  - But it holds no matter what the schema; we'll assume *ABCD*.

The Tableau for A→→C

Goal: derive tuple (a, b_1, c_2, d_1) ≡ A →→ C

A	*B*	*C*	*D*
a	b_1	c_1	d_1
a	b_2	c_2	d_2
a	b_1	c_2	d_2
a	b_1	c_2	d_1

Rows 1,2

Use A→→B to swap *B* from the first row into the second.

Rows 1,3

Use B→→C to swap *C* from the third row into the first.

You can see A→→C from row 4 and rows 1& 2

Exercise:

- If A→B and B→→C, prove that A→→C?

Rules for Inferring MVD's + FD's

- Start with a tableau of two rows:
  - These rows agree on the attributes of the left side of the dependency to be inferred
  - And they disagree on all other attributes
  - Use unsubscripted variables where they agree, subscripts where they disagree

Inference: Applying an FD – pg 80

- Infer an FD X→Y by finding rows that agree on all attributes of X. Force the rows to agree on all attributes of Y:

  A→→BC and D→C, then A→C

  - Replace one variable by the other (D & BC)
  - If the replaced variable (C) is part of the goal tuple (A → C), replace it there too

Inference: Applying a MVD – pg 82

- Apply an MVD X→→Y by finding two rows that agree in X:
  - Add to the tableau one or both rows that are formed by swapping the Y-components of these two rows

Inference: Goals

- To test whether U→V holds, we succeed by inferring that the two variables in each column of V are actually the same

- If we are testing U→→V, we succeed if we infer in the tableau a row that is the original two rows with the components of V swapped

Inference: Endgame

- Apply all the given FD's and MVD's until we cannot change the tableau
- If we meet the goal, then the dependency is inferred
- If not, then the final tableau is a counterexample relation:
  - Satisfies all given dependencies
  - Original two rows violate target dependency

END