

Schedule

- Today: Jan. 10 (TH)
 - ◆ Relational Model, Functional Dependencies.
 - ◆ Read Sections 3.1-3.5.
- Jan. 15 (T)
 - ◆ Normal Forms, Multivalued Dependencies.
 - ◆ Read Sections 3.6-3.7. Assignment 1 due.
- Jan. 17 (TH)
 - ◆ Relational Algebra.
 - ◆ Read Chapter 5. Project Part 1 due.
- Jan. 22 (T)
 - ◆ SQL Queries.
 - ◆ Read Sections 6.1-6.2. Assignment 2 due.

Relational Model

- Table = *relation*.
- Column headers = *attributes*.

- Row = *tuple*

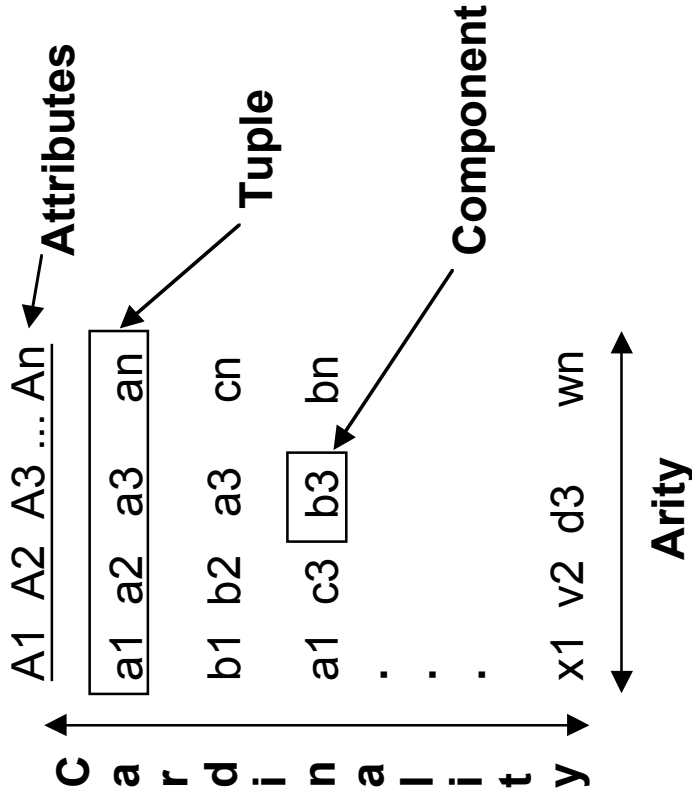
name	manf
WinterBrew	Pete's
BudLite	A.B.
...	...

Beers

- *Relation schema* = name(attributes) + other structure info., e.g., keys, other constraints. Example: `Beers (name, manf)`
 - ◆ Order of attributes is arbitrary, but in practice we need to assume the order given in the relation schema.
- *Relation instance* is current set of rows for a relation schema.
- *Database schema* = collection of relation schemas.

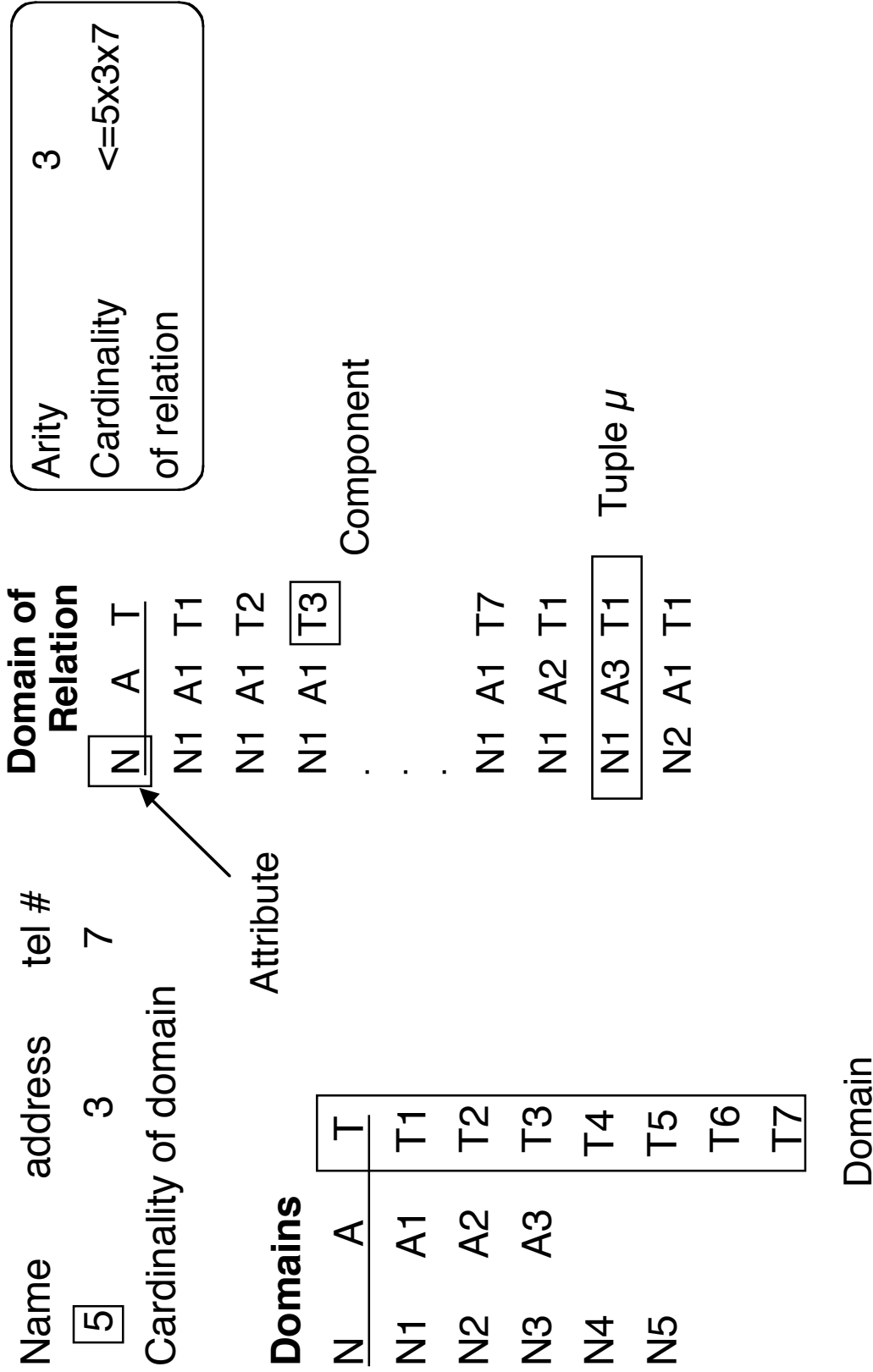
Relational Data Model

Relation as table
 Rows = tuples
 Columns = components
 Names of columns = attributes
 Set of attribute names = schema
 REL ($A_1, A_2, \dots, A_n$)



Set theoretic
 Domain — set of values
 like a data type
 Cartesian product (or product)
 $D_1 \times D_2 \times \dots \times D_n$
 k-tuples ($V_1, V_2, \dots, V_n$)
 s.t., $V_1 \in D_1, V_2 \in D_2, \dots, V_n \in D_n$
 Relation-subset of cartesian product
 of one or more domains
 FINITE only; empty set allowed
 Tuples = members of a relation inst.
 Arity = number of domains
 Components = values in a tuple
 Domains — corresp. with attributes
 Cardinality = number of tuples

Relation: Example



Relation Instance

<u>Name</u>	<u>Address</u>	<u>Telephone</u>
Bob	123 Main St	555-1234
Bob	128 Main St	555-1235
Pat	123 Main St	555-1235
Harry	456 Main St	555-2221
Sally	456 Main St	555-2221
Sally	456 Main St	555-2223
Pat	12 State St	555-1235

About Relational Model

Order of tuples not important

Order of attributes not important (in theory)

Collection of relation schemas (intension)

Relational database schema

Corresponding relation instances (extension)

Relational database

intension vs. extension

schema vs. data

metadata

includes schema

Why Relations?

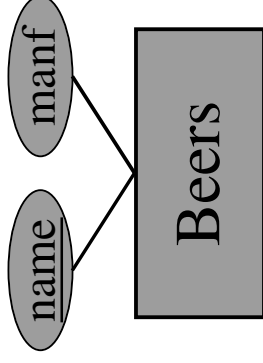
- Very simple model.
- *Often* a good match for the way we think about our data.
- Abstract model that underlies SQL, the most important language in DBMS's today.
 - ◆ But SQL uses “bags” while the abstract relational model is set-oriented.

Relational Design

Simplest approach (not always best): convert each E.S. to a relation and each relationship to a relation.

Entity Set $\rightarrow$ Relation

E.S. attributes become relational attributes.



Becomes:

Beers (name, manf)

Keys in Relations

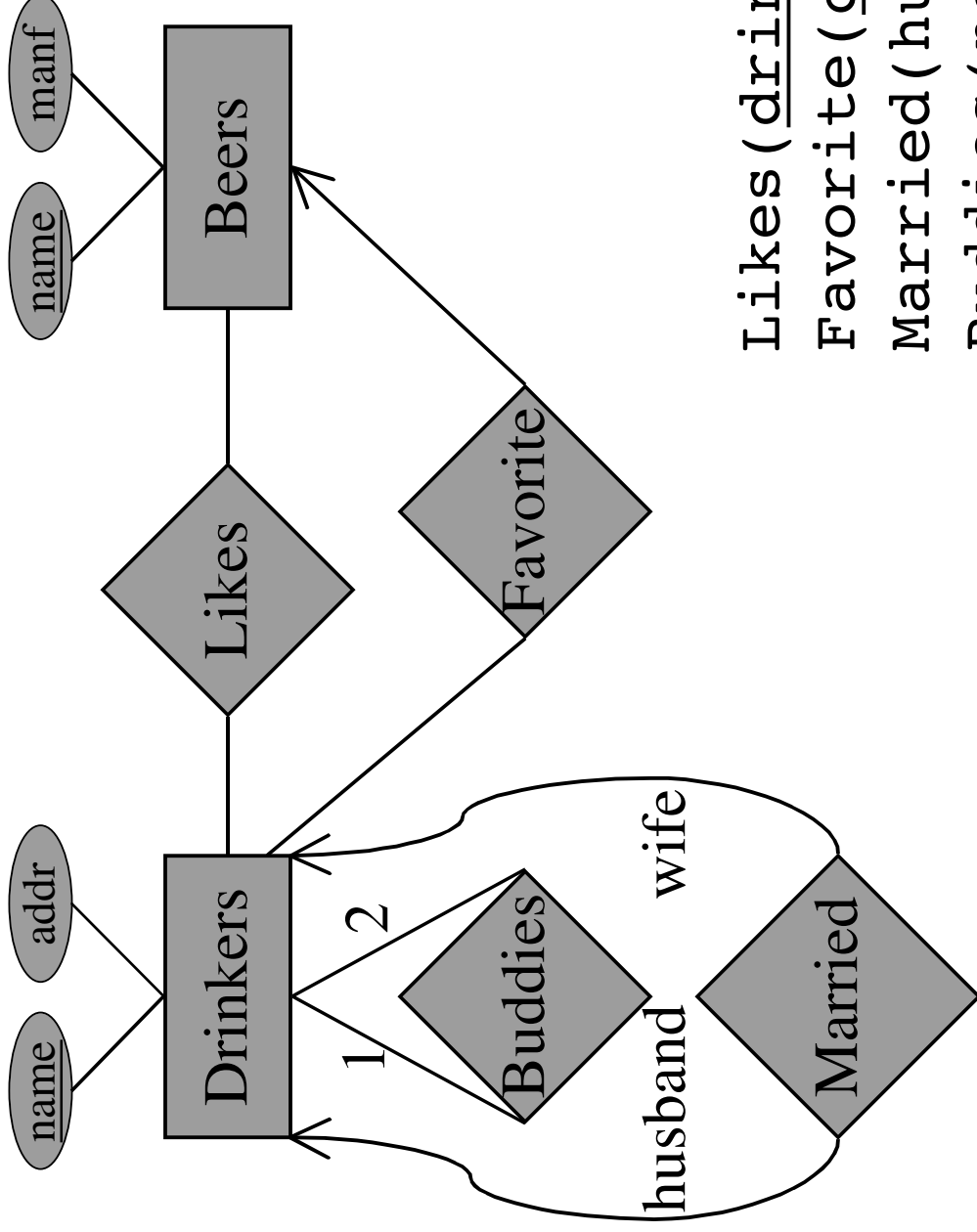
An attribute or set of attributes K is a *key* for a relation R if we expect that in no instance of R will two different tuples agree on all the attributes of K .

- Indicate a key by underlining the key attributes.
- Example: If **name** is a key for Beers:
Beers(name, manf)

E/R Relationships $\rightarrow$ Relations

Relation has attribute for *key* attributes of each E.S. that participates in the relationship.

- Add any attributes that belong to the relationship itself.
- Renaming attributes OK.
- ♦ Essential if multiple roles for an E.S.



Likes(drinker, beer)
 Favorite(drinker, beer)
 Married(husband, wife)
 Buddies(name1, name2)

- For one-one relation Married, we can choose either husband or wife as key.

Combining Relations

Sometimes it makes sense to combine relations.

- Common case: Relation for an E.S. E plus the relation for some many-one relationship from E to another E.S.

Example

Combine `Drinker(name, addr)` with `Favorite(drinker, beer)` to get `Drinker1(name, addr, favBeer)`.

- Danger in pushing this idea too far: redundancy.
- *e.g.*, combining `Drinker` with `Likes` causes the drinker's address to be repeated, viz.:

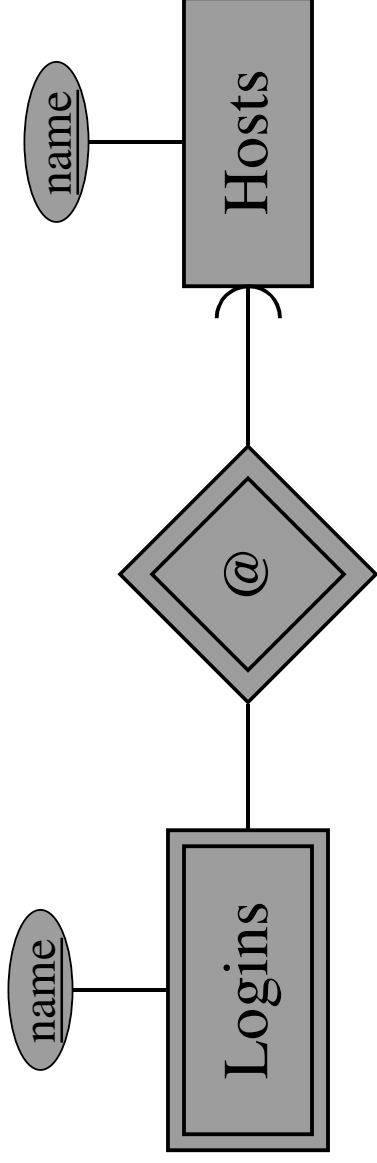
<u>name</u>	<u>addr</u>	<u>beer</u>
Sally	123 Maple	Bud
Sally	123 Maple	Miller

- Notice the difference: `Favorite` is many-one; `Likes` is many-many.

Weak Entity Sets, Relationships → Relations

- Relation for a weak E.S. must include its full key (*i.e.*, attributes of related entity sets) as well as its own attributes.
- A supporting (double-diamond) relationship yields a relation that is actually redundant and should be deleted from the database schema.

Example



Hosts (hostName)

Logins (loginName, hostName)

At (loginName, hostName, hostName2)

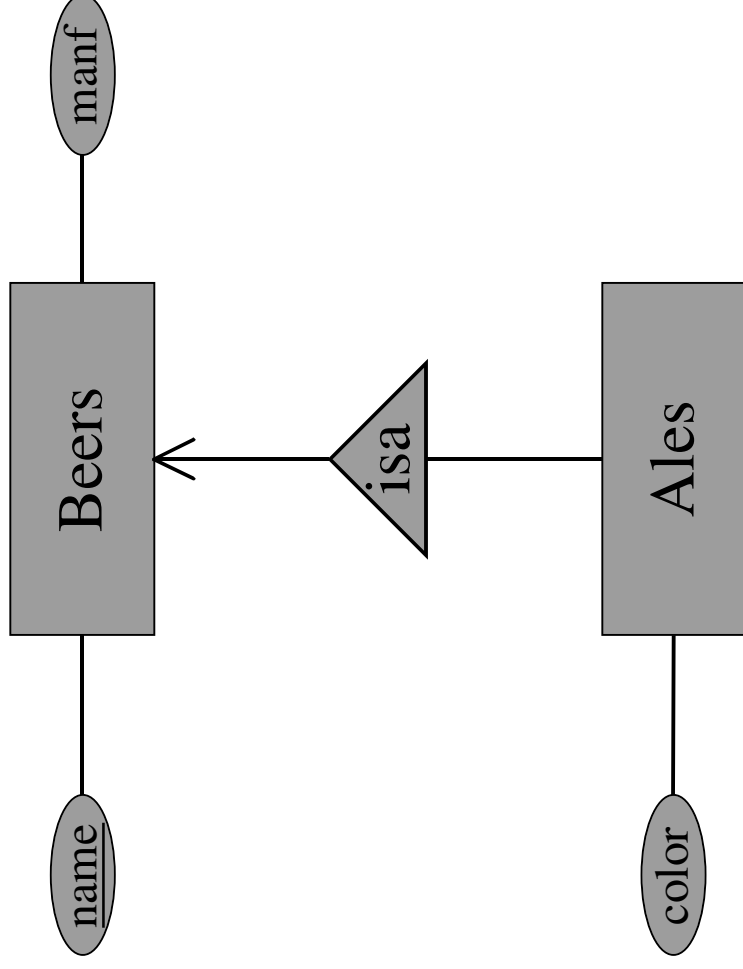
- In **At**, **hostName** and **hostName2** must be the same host, so delete one of them.
- Then, **Logins** and **At** become the same relation; delete one of them.
- In this case, **Hosts**' schema is a subset of **Logins**' schema. Delete **Hosts**?

Subclasses $\rightarrow$ Relations

Three approaches:

1. Object-oriented: each entity is in one class. Create a relation for each class, with all the attributes for that class.
 - ◆ Don't forget inherited attributes.
2. E/R style: an entity is in a network of classes related by *isa*.
Create one relation for each E.S.
 - ◆ An entity is represented in the relation for each subclass to which it belongs.
 - ◆ Relation has only the attributes attached to that E.S. + key.
3. Use nulls. Create one relation for the root class or root E.S., with all attributes found anywhere in its network of subclasses.
 - ◆ Put NULL in attributes not relevant to a given entity.

Example



OO-Style

name	manf	name	manf	color
Bud	A.B.	SummerBrew	Pete's	dark

Beers

Ates

E/R Style

name	manf	name	Color
Bud	A.B.	SummerBrew	dark
SummerBrew	Pete's		

Bears

Atles

Using NULLS

name	manf	color
Bud	A.B.	NULL
SummerBrew	Pete's	dark

Bears

Functional Dependencies

$X \twoheadrightarrow A$ = assertion about a relation R that
whenever two tuples agree on all the
attributes of X , then they must also agree on
attribute A .

Example

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

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

- Reasonable FD's to assert:

1. $\text{name} \rightarrow \text{addr}$
2. $\text{name} \rightarrow \text{favoriteBeer}$
3. $\text{beersLiked} \rightarrow \text{manf}$

- Shorthand: combine FD's with common left side by concatenating their right sides.
- Sometimes, several attributes jointly determine another attribute, although neither does by itself. Example:

beer bar $\rightarrow$ price

Keys of Relations

K is a *key* for relation R if:

1. $K \rightarrow$ all attributes of R . (**Uniqueness**)
 2. For no proper subset of K is (1) true. (**Minimality**)
- If K at least satisfies (1), then K is a *superkey*.

Conventions

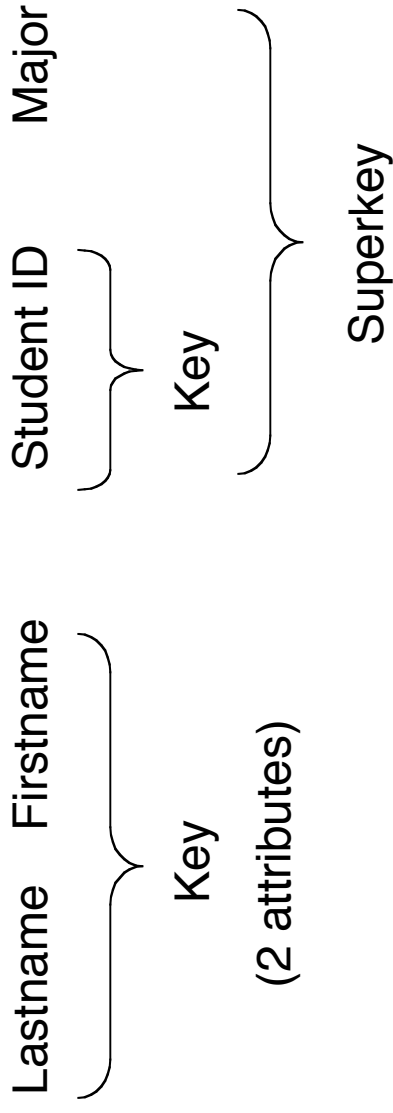
- Pick one key; underline key attributes in the relation schema.
- X , etc., represent sets of attributes; A etc., represent single attributes.
- No set formers in FD's, e.g., ABC instead of $\{A, B, C\}$.

Example

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

- {name, beersLiked} FD's all attributes, as seen.
 - ◆ Shows {name, beersLiked} is a superkey.
- $\text{name} \rightarrow \text{beersLiked}$ is false, so name not a superkey.
- $\text{beersLiked} \rightarrow \text{name}$ also false, so beersLiked not a superkey.
- Thus, {name, beersLiked} is a key.
- No other keys in this example.
 - ◆ Neither name nor beersLiked is on the right of any observed FD, so they must be part of *any* superkey.
- Important point: “key” in a relation refers to tuples, not the entities they represent. If an entity is represented by several tuples, then entity-key will not be the same as relation-key.

Example 2



Note: There are alternate keys

- **Keys are {Lastname, Firstname} and {StudentID}**

Who Determines Keys/FD's?

- We could assert a key K .
 - ◆ Then the only FD's asserted are that $K \rightarrow A$ for every attribute A .
 - ◆ No surprise: K is then the only key for those FD's, according to the formal definition of “key.”
- Or, we could assert some FD's and *deduce* one or more keys by the formal definition.
 - ◆ E/R diagram implies FD's by key declarations and many-one relationship declarations.
- Rule of thumb: FD's either come from keyness, many-1 relationship, or from physics.
 - ◆ *E.g.*, “no two courses can meet in the same room at the same time”
yields **room time** $\rightarrow$ **course**.

Functional Dependencies (FD's) and Many-One Relationships

- Consider $R(A_1, \dots, A_n)$ and X is a key then $X \rightarrow Y$ for any attributes Y in $A_1, \dots, A_n$ even if they overlap with X . Why?
- Suppose R is used to represent a many $\rightarrow$ one relationship:
 E_1 entity set $\rightarrow E_2$ entity set
 where X key for E_1 , Y key for E_2 ,
 Then, $X \rightarrow Y$ holds,
 And $Y \rightarrow X$ does not hold unless the relationship is one-one.
- What about many-many relationships?

Inferring FD's

And this is important because ...

- When we talk about improving relational designs, we often need to ask “does this FD hold in this relation?”

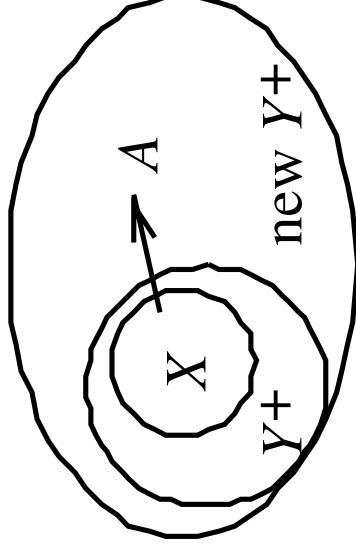
Given FD's $X1 \rightarrow A1$, $X2 \rightarrow A2, \dots, Xn \rightarrow An$, does FD $Y \rightarrow B$ necessarily hold in the same relation?

- Start by assuming two tuples agree in Y . Use given FD's to infer other attributes on which they must agree. If B is among them, then yes, else no.

Algorithm

Define $Y^+ = \text{closure of } Y = \text{set of attributes functionally determined by } Y$:

- Basis: $Y^+ := Y$.
- Induction: If $X \subseteq Y^+$, and $X \rightarrow A$ is a given FD, then add A to Y^+ .

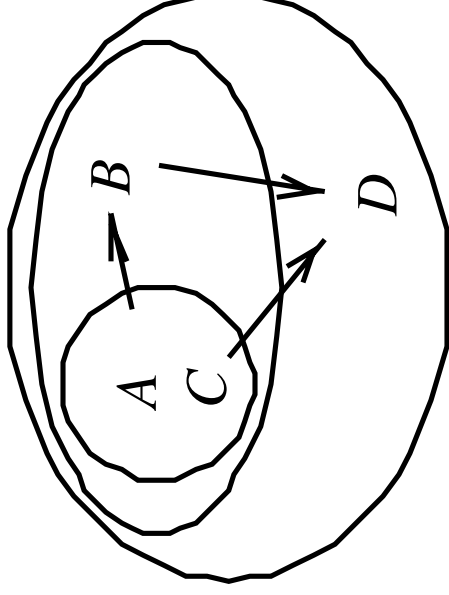


- End when Y^+ cannot be changed.

Example

$A \rightarrow B, BC \rightarrow D.$

- $A^+ = AB.$
- $C^+ = C.$
- $(AC)^+ = ABCD.$



Given Versus Implied FD's

Typically, we state a few FD's that are known to hold for a relation R .

- Other FD's may follow logically from the given FD's; these are *implied FD's*.
- We are free to choose any *basis* for the FD's of R – a set of FD's that imply all the FD's that hold for R .

Finding All Implied FD's

Motivation: Suppose we have a relation $ABCD$ with some FD's F . If we decide to decompose $ABCD$ into ABC and AD , what are the FD's for ABC , AD ?

- Example: $F = AB \rightarrow C, C \rightarrow D, D \rightarrow A$. It looks like just $AB \rightarrow C$ holds in ABC , but in fact $C \rightarrow A$ follows from F and applies to relation ABC .
- Problem is exponential in worst case.

Algorithm

- For each set of attributes X compute X^+ .
 - ◆ But skip $X = \emptyset$, $X =$ all attributes.
 - ◆ Add $X \rightarrow A$ for each A in $X^+ - X$.
- Drop $XY \rightarrow A$ if $X \rightarrow A$ holds.
- ◆ Consequence: If X^+ is all attributes, then there is no point in computing closure of supersets of X .
- Finally, project the FD's by selecting only those FD's that involve only the attributes of the projection.
- ◆ Notice that after we project the discovered FD's onto some relation, the eliminated FD's can be inferred in the *projected relation*.

Example

$F = AB \rightarrow C, C \rightarrow D, D \rightarrow A$. What FD's follow?

- $A^+ = A; B^+ = B$ (nothing).
- $C^+ = ACD$ (add $C \rightarrow A$).
- $D^+ = AD$ (nothing new).
- $(AB)^+ = ABCD$ (add $AB \rightarrow D$; skip all supersets of AB).
- $(BC)^+ = ABCD$ (nothing new; skip all supersets of BC).
- $(BD)^+ = ABCD$ (add $BD \rightarrow C$; skip all supersets of BD).
- $(AC)^+ = ACD; (AD)^+ = AD; (CD)^+ = ACD$ (nothing new).
- $(ACD)^+ = ACD$ (nothing new).
- All other sets contain AB, BC , or BD , so skip.
- Thus, the only interesting FD's that follow from F are:
 $C \rightarrow A, AB \rightarrow D, BD \rightarrow C$.

Example 2

- Set of FD's in $ABCGHI$:

$$A \rightarrow B$$

$$A \rightarrow C$$

$$CG \rightarrow H$$

$$CG \rightarrow I$$

$$B \rightarrow H$$

- Compute $(CG)^+$, $(BG)^+$, $(AG)^+$

Example 3

In ABC with FD's $A \rightarrow B, B \rightarrow C$, project onto AC .

1. $A^+ = ABC$; yields $A \rightarrow B, A \rightarrow C$.
 2. $B^+ = BC$; yields $B \rightarrow C$.
 3. $AB^+ = ABC$; yields $AB \rightarrow C$; drop in favor of $A \rightarrow C$.
 4. $AC^+ = ABC$ yields $AC \rightarrow B$; drop in favor of $A \rightarrow B$.
 5. $C^+ = C$ and $BC^+ = BC$; adds nothing.
- Resulting FD's: $A \rightarrow B, A \rightarrow C, B \rightarrow C$.
 - Projection onto AC : $A \rightarrow C$.