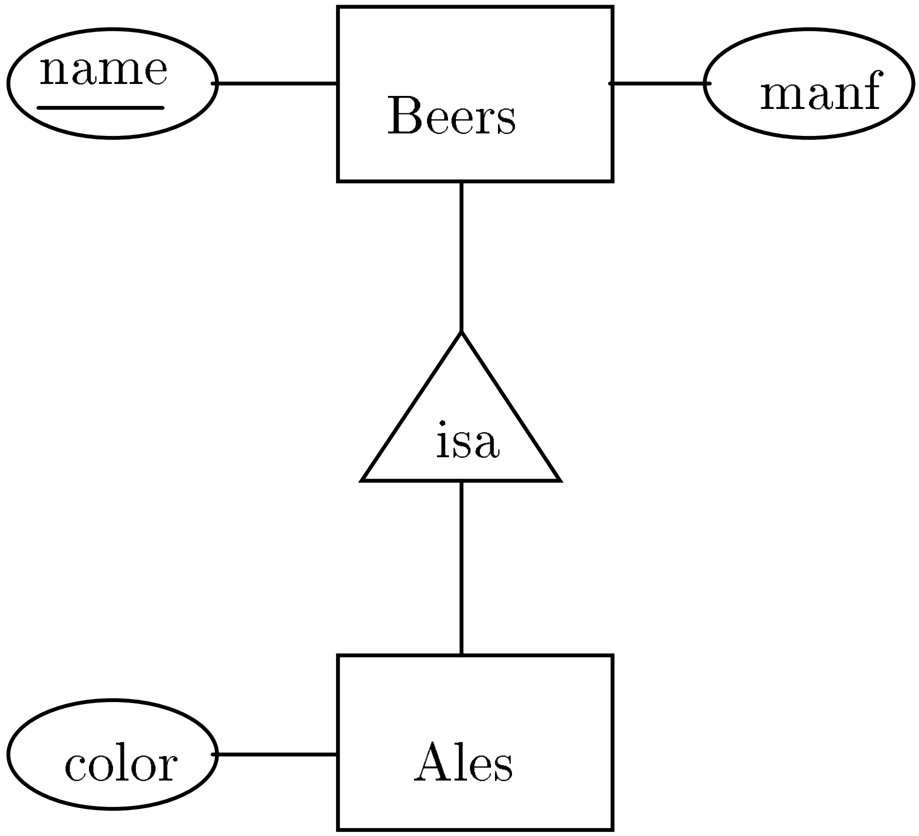
Subclasses → Relations

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
Bud	A.B.

Beers

name	manf	color
SummerBrew	Pete's	dark

Ales

E/R Style

name	manf
Bud	A.B.
SummerBrew	Pete's

Beers

name	color
SummerBrew	dark

Ales

Using Nulls

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

Beers

Functional Dependencies

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

- Important as a constraint on the data that may appear within a relation.
  - Schema-level control of data.
- Mathematical tool for explaining the process of "normalization" — vital for redesigning database schemas when original design has certain flaws.

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. `name` $\rightarrow$ `addr`
2. `name` $\rightarrow$ `favoriteBeer`
3. `beersLiked` $\rightarrow$ `manf`

- Note: These happen to imply the underlined key, but the FD's give more detail than the mere assertion of a key.

- Key (in general) functionally determines all attributes. In our example:

`name beersLiked` $\rightarrow$ `addr favoriteBeer beerManf`

- Shorthand: combine FD's with common left side by concatenating their right sides.
- When FD's are *not* of the form Key $\rightarrow$ other attribute(s), then there is typically an attempt to "cram" too much into one relation.
- Sometimes, several attributes jointly determine another attribute, although neither does by itself. Example:

  `beer bar` $\rightarrow$ `price`

Formal Notion of Key

K is a *key* for relation R if:

1. $K \rightarrow$ all attributes of R.
2. For no proper subset of K is (1) true.

- If K at least satisfies (1), then K is a *superkey*.

FD Conventions

- 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(`<u>`name`</u>`, addr, `<u>`beersLiked`</u>`, manf, favoriteBeer)`

- {`name, beersLiked`} FD's all attributes, as seen.
  - ❖ Shows {`name, beersLiked`} is a superkey.
- `name` → `beersLiked` is false, so `name` not a superkey.
- `beersLiked` → `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.

Who Determines Keys/FD's?

- We could define a relation schema by simply giving a single 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`.

Normalization

Goal = BCNF = Boyce-Codd Normal Form = all FD's follow from the fact "key $\rightarrow$ everything."

- Formally, R is in BCNF if every nontrivial FD for R, say $X \rightarrow A$, has X a superkey.
  - "Nontrivial" = right-side attribute not in left side.

Why?

1. Guarantees no redundancy due to FD's.
2. Guarantees no *update anomalies* = one occurrence of a fact is updated, not all.
3. Guarantees no *deletion anomalies* = valid fact is lost when tuple is deleted.

Example of Problems

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

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

FD's:

1. `name` $\rightarrow$ `addr`
2. `name` $\rightarrow$ `favoriteBeer`
3. `beersLiked` $\rightarrow$ `manf`

- ???'s are redundant, since we can figure them out from the FD's.
- Update anomalies: If Janeway gets transferred to the *Intrepid*, will we change `addr` in each of her tuples?
- Deletion anomalies: If nobody likes Bud, we lose track of Bud's manufacturer.

Each of the given FD's is a BCNF violation:

- Key = {`name`, `beersLiked`}
    - Each of the given FD's has a left side a proper subset of the key.

Another Example

`Beers(name, manf, manfAddr)`.

- FD's = `name` $\rightarrow$ `manf`, `manf` $\rightarrow$ `manfAddr`.
- Only key is `name`.
    - `manf` $\rightarrow$ `manfAddr` violates BCNF with a left side unrelated to any key.

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?"

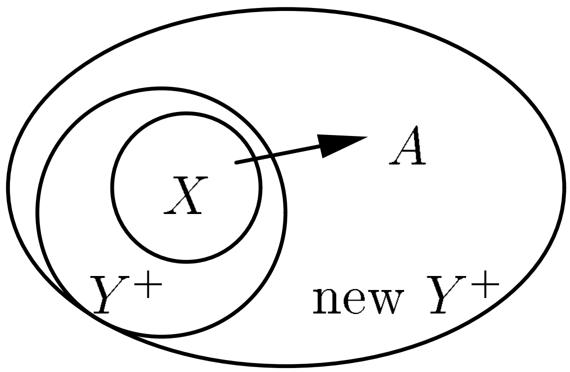
Given FD's $X1 \rightarrow A1$, $X2 \rightarrow A2 \cdots Xn \rightarrow An$, does FD $Y \rightarrow B$ necessarily hold in the same relation?

- 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^+ = *closure* of Y:

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

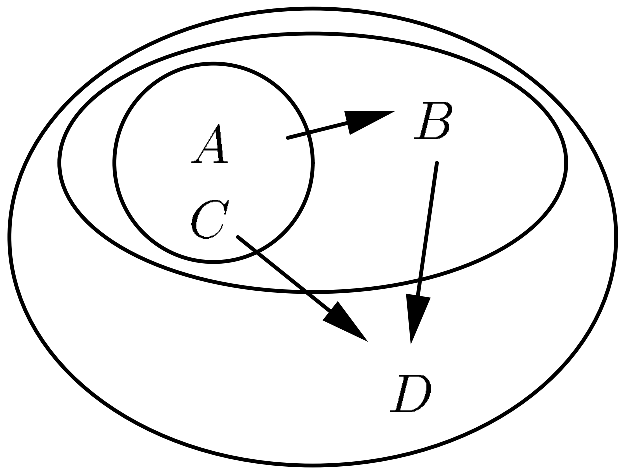


- End when Y^+ cannot be changed. Then Y functionally determines all members of Y^+, and no other attributes.

Example

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

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



- Thus, AC is a key.