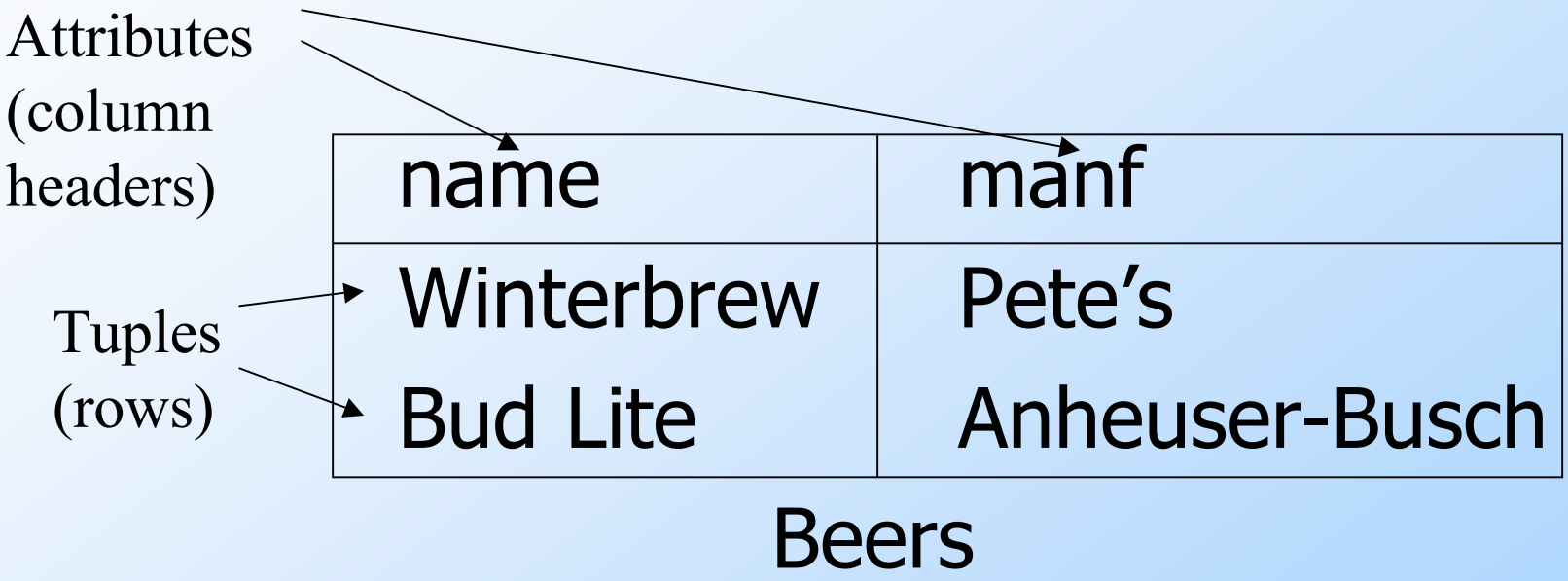
The Relational Data Model

Tables

Schemas

Conversion from E/R to Relations

A Relation is a Table



Schemas

- *Relation schema* = relation name + attributes, in order (+ types of attributes).
  - Example: Beers(name, manf)
    or Beers(name: string, manf: string)
- *Relational Database* = collection of relations.
- *Relational Database schema* = set of all relation schemas in the database.

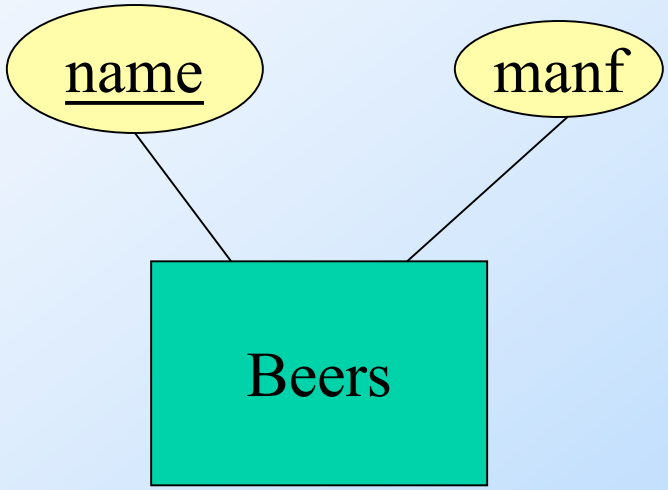
Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.
  - But SQL uses bags, while the relational model is a set-based model.

From E/R Diagrams to Relations

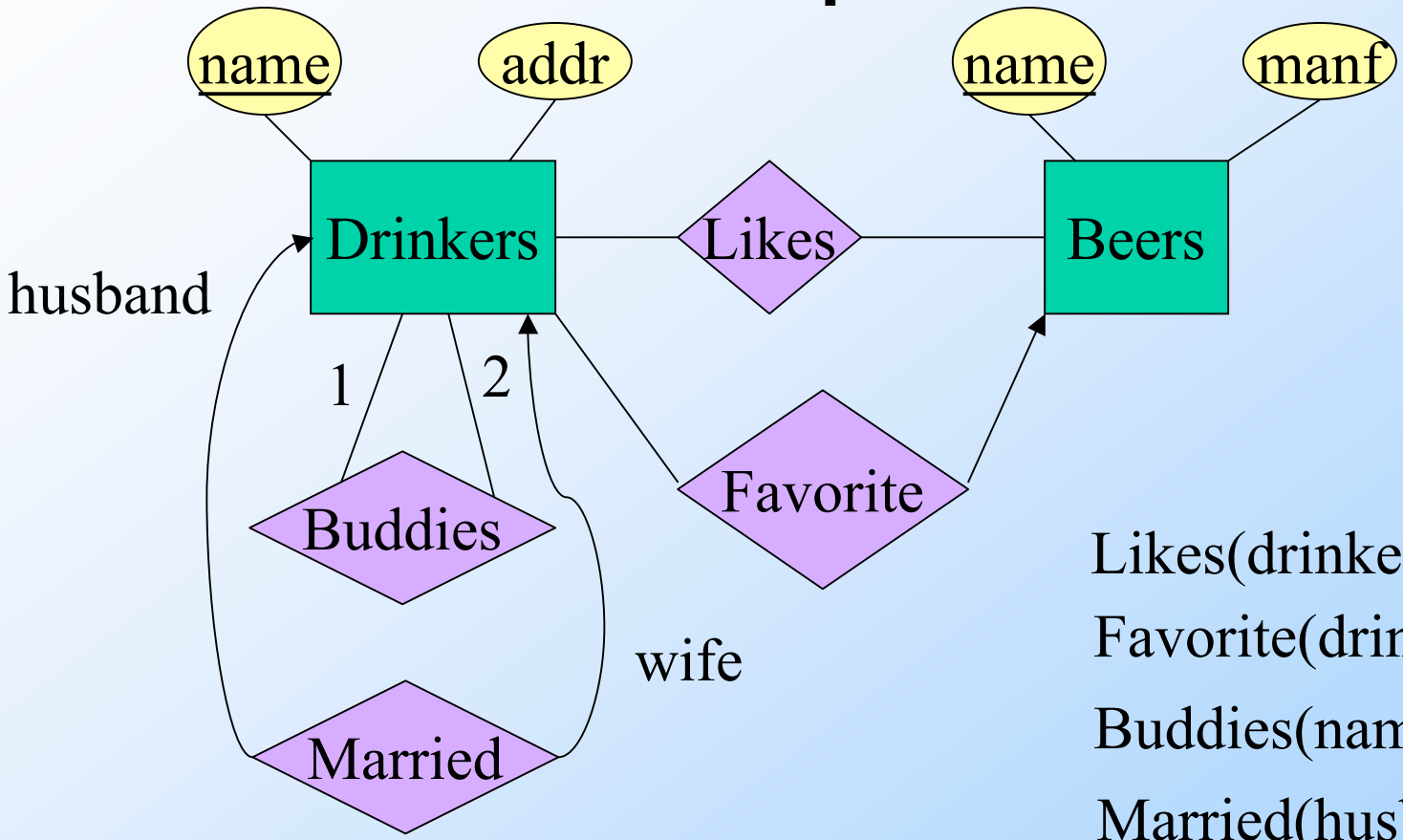
- Entity sets become relations with the same set of attributes.
- Relationship sets become relations whose attributes are only:
  - The **keys** of connected entity sets.
  - Attributes of the relationship itself.

Entity Set -> Relation



Relation: Beers(name, manf)

Relationship -> Relation



Likes(drinker, beer)

Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Combining Relations

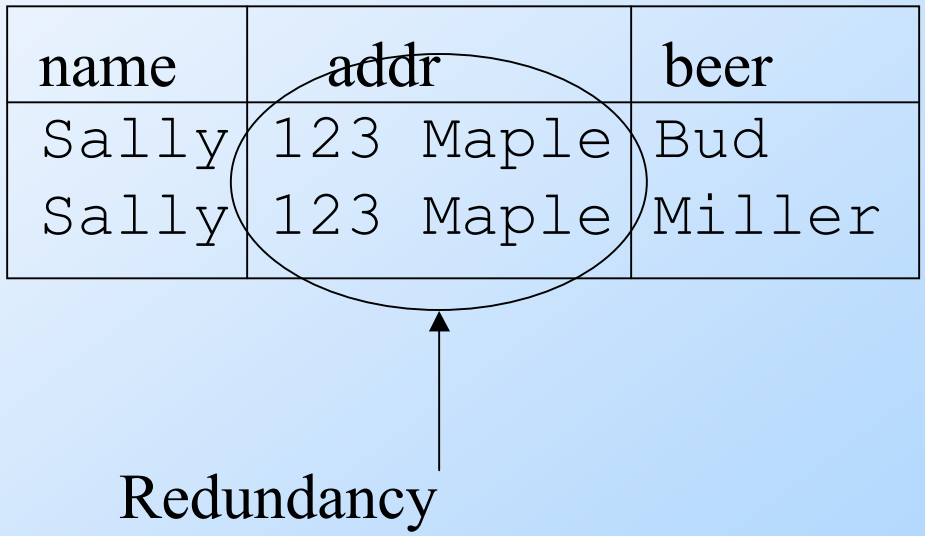
- It is OK to combine the relation for an entity-set *E* with the relation *R* for a many-one relationship from *E* to another entity set E'.

- Example:
  E = Drinkers(name, addr)
  R = Favorite(drinker, beer) many-one

  E' = Drinker1(name, addr, favBeer).

Risk with **Many-Many** Relationships

- Combining Drinkers with Likes would be a mistake. It leads to redundancy, as:

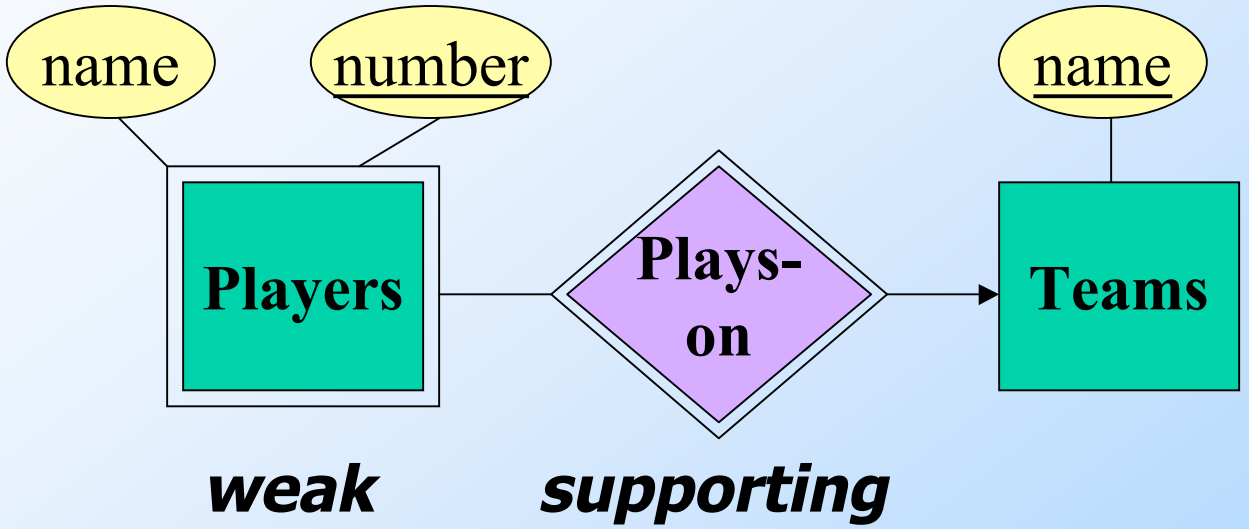


Handling Weak Entity Sets

- The ***relation*** for a weak ***entity*** set **must *include* attributes for its complete key** (including those belonging to other entity sets), **as well as its own**, nonkey attributes.

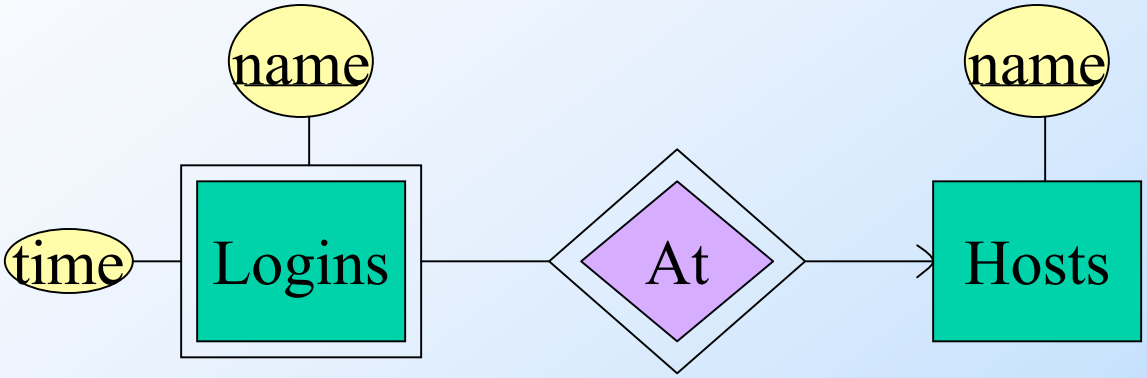
- A relation corresponding to supporting (double-diamond) relationship is thus **redundant**, and should not be included in the relational schema.

Example



Teams(teamName)
Players(name, number, teamName) ***including key attributes***
Playson(name, number, teamName) ***redundant relation***

Ullman's Example



Hosts(hostName)

Logins(loginName, hostName, time)

~~At(loginName, hostName, hostName2)~~

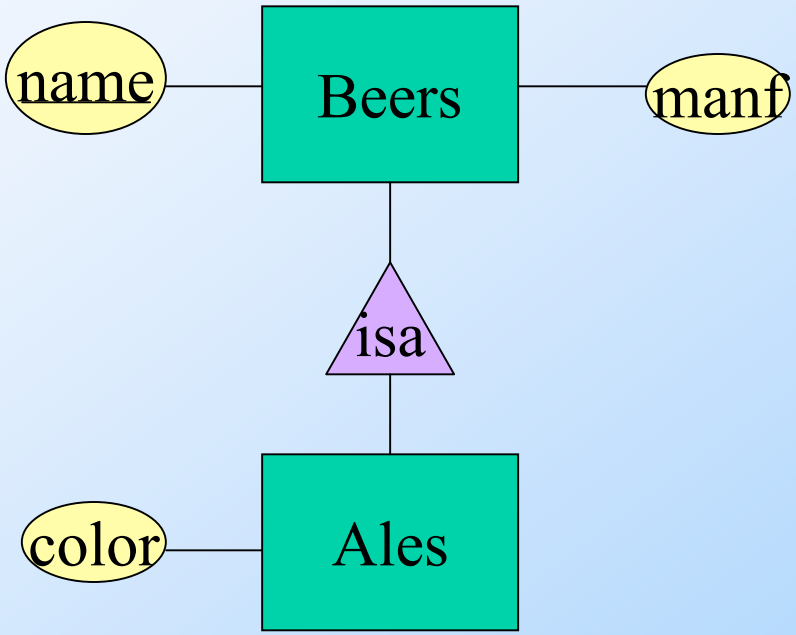
At becomes part of Logins

Must be the same

Entity Sets With Subclasses

- Three approaches:
  1. *Object-oriented* : each entity belongs to exactly one class; create a relation for each class, with all its attributes. [??]
  2. *E/R style* : create one relation for each subclass, with only the key attribute(s) and attributes attached to that E.S.; entity represented in all relations to whose subclass/E.S. it belongs.
  3. *Use nulls* : create one relation; entities have null in attributes that don't belong to them.

Example



Object-Oriented Style [??]

[knowing that every Ale isa Beer]

Beers

name	manf
Bud	Anheuser-Busch

Ales

name	manf	color
Summerbrew	Pete's	dark

E/R Style

Beers

name	manf
Bud	Anheuser-Busch
Summerbrew	Pete's

Ales

name	color
Summerbrew	dark

Using Nulls

Beers

name	manf	color
Bud	Anheuser-Busch	NULL
Summerbrew	Pete's	dark

Comparisons

- O-O approach good for queries like "find the color of ales made by Pete's."
  - Just look in Ales relation.
- E/R approach good for queries like "find all beers (including ales) made by Pete's."
  - Just look in Beers relation.
- Using nulls saves space unless there are *lots* of attributes that are usually null.