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.

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(<u>name</u>, manf)

Why Relations?

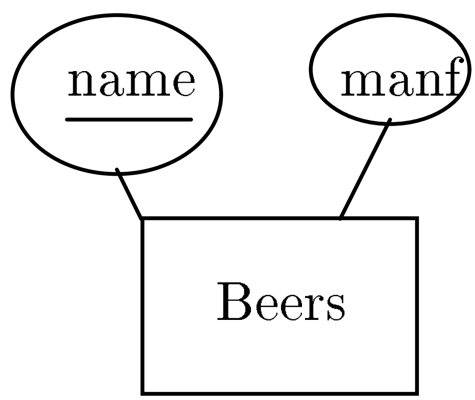
- Very simple model.
- *Often* a good match for the way we think about our data.
- Abstract model that underlies SQL, the most important language in DBMS's today.
  - But SQL uses "bags," while the abstract relational model is set-oriented.

Relational Design

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

Entity Set → Relation

E.S. attributes become relational attributes.



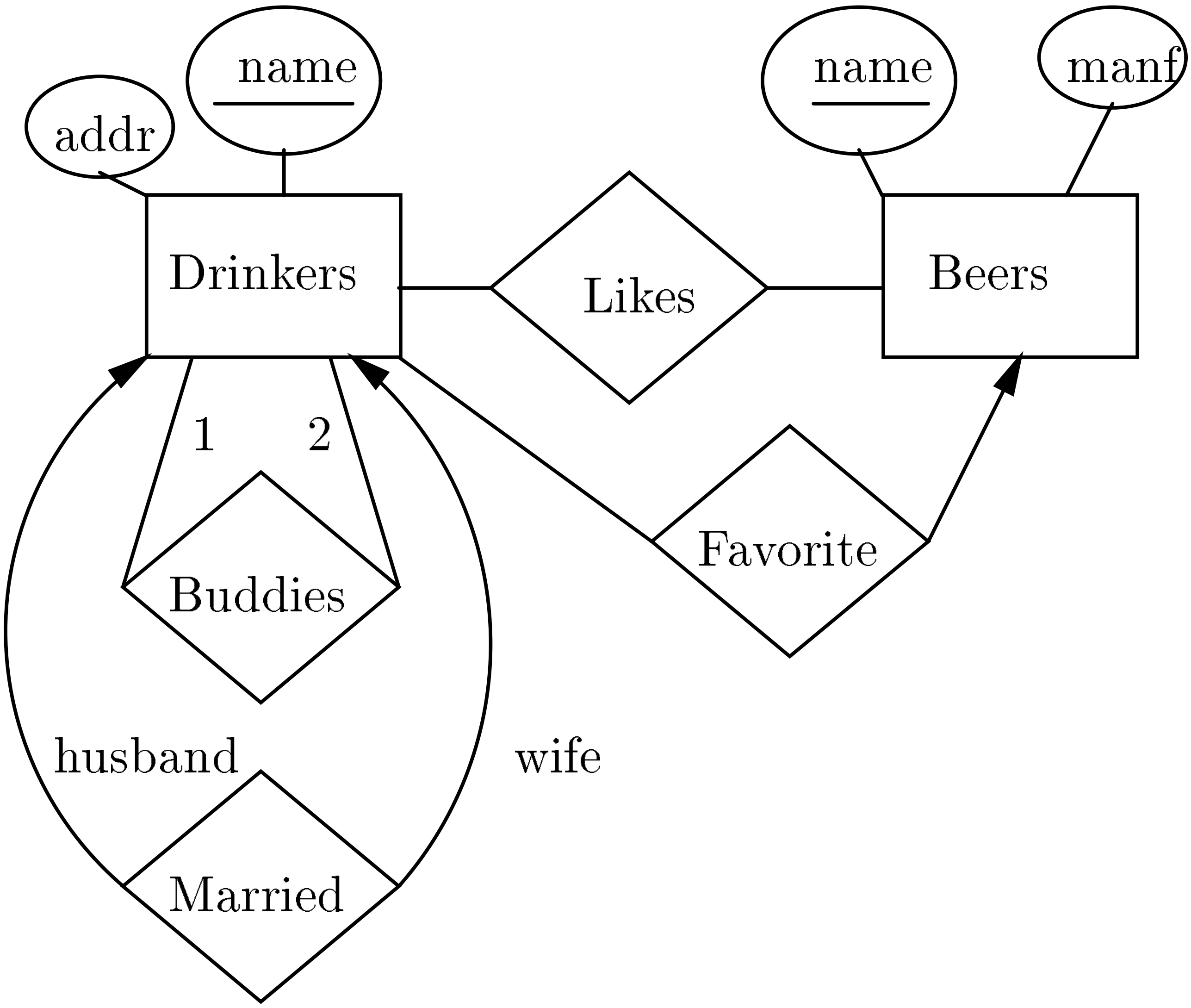
Becomes:

```
Beers(name, manf)
```

E/R Relationships → 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.:

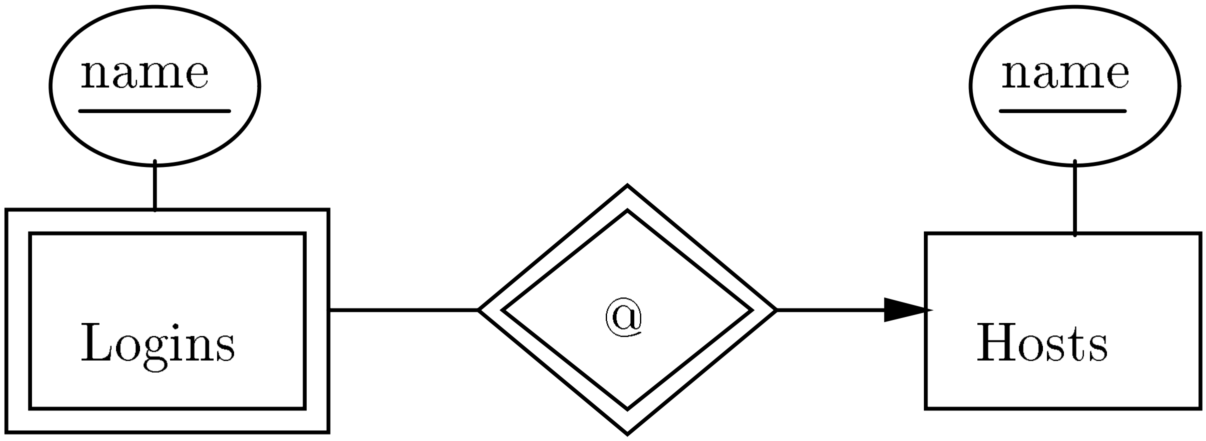
name	addr	beer
Sally	123 Maple	Bud
Sally	123 Maple	Miller

- 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)

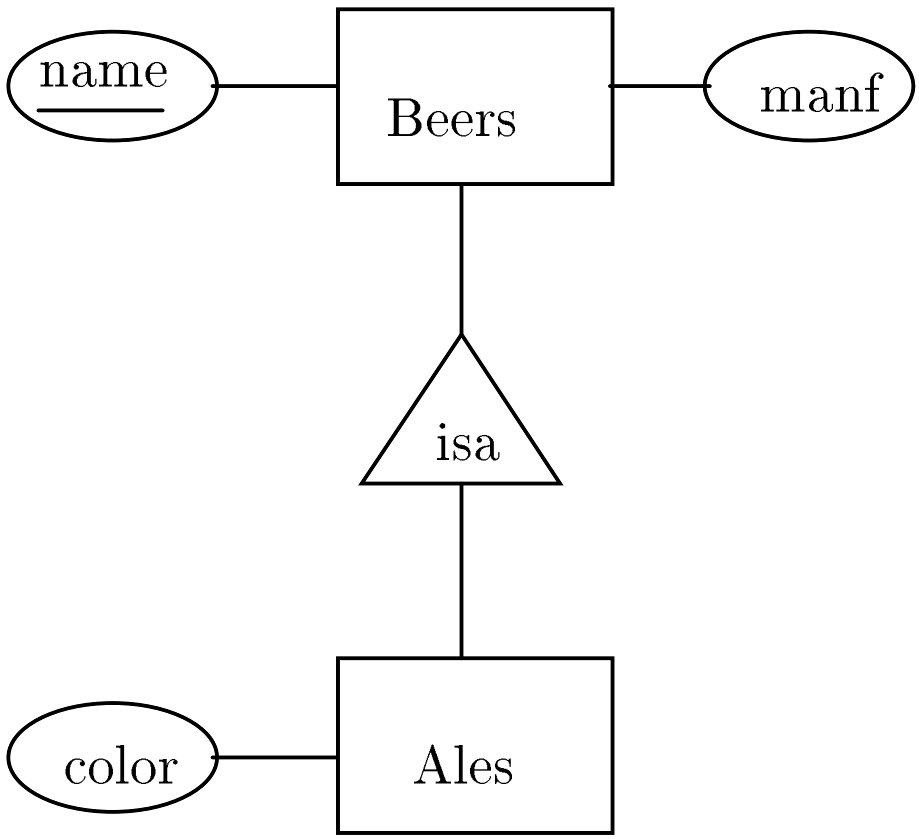
- In `At`, `hostName` and `hostName2` must be the same host, so delete one of them.
- Then, `Logins` and `At` become the same relation; delete one of them.
- In this case, `Hosts`' schema is a subset of `Logins`' schema. Delete `Hosts`?

Subclasses → 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

$X \to A$ = 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(`<u>`name`</u>`, addr, `<u>`beersLiked`</u>`, 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` → `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 → 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` → `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` $\rightarrow$ `beersLiked` is false, so `name` not a superkey.
- `beersLiked` $\rightarrow$ `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` → `course`.