Weak Entity Sets

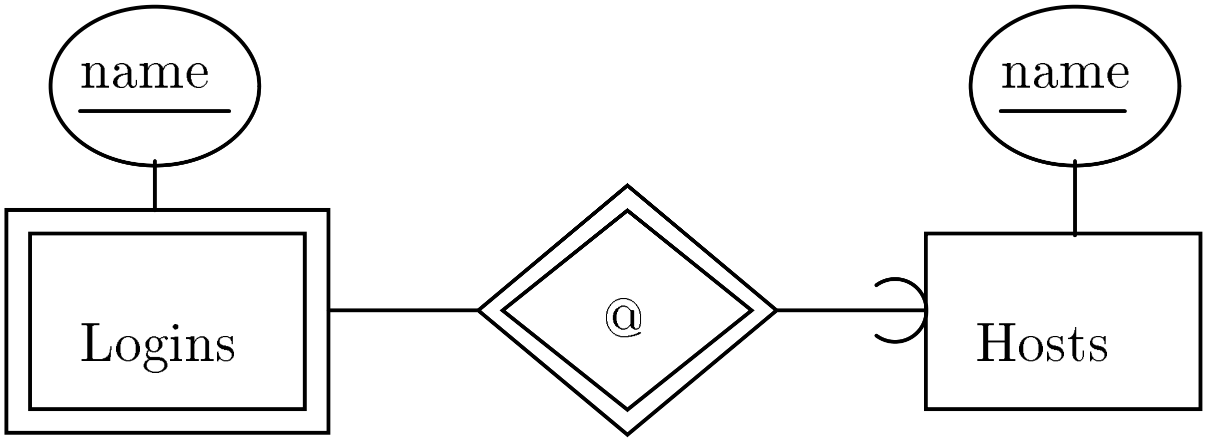
Sometimes an E.S. E's key comes not (completely) from its own attributes, but from the keys of one or more E.S's to which E is linked by a *supporting* many-one relationship.

- Called a *weak* E.S.
- Represented by putting double rectangle around E and a double diamond around each supporting relationship.
- Many-one-ness of supporting relationship (includes 1-1) essential.
    - With many-many, we wouldn't know which entity provided the key value.
- "Exactly one" also essential, or else we might not be able to extract key attributes by following the supporting relationship.

Example: Logins (Email Addresses)

Login name = user name + host name, e.g., `ullman@shalmaneser.stanford.edu`.

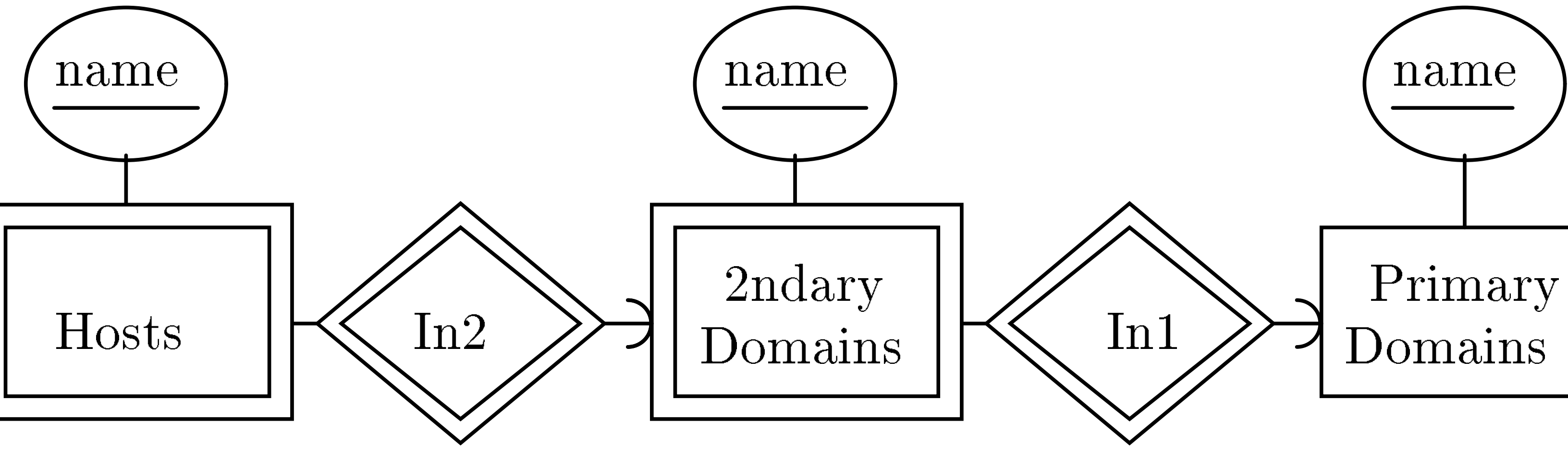
- A "login" entity corresponds to a user name on a particular host, but the passwd table doesn't record the host, just the user name, e.g. `ullman`.
- Key for a login = the user name at the host (which is unique for that host only) + the IP address of the host (which is unique globally).



- Design issue: Under what circumstances could we simply make login-name and host-name be attributes of logins, and dispense with the weak E.S.?

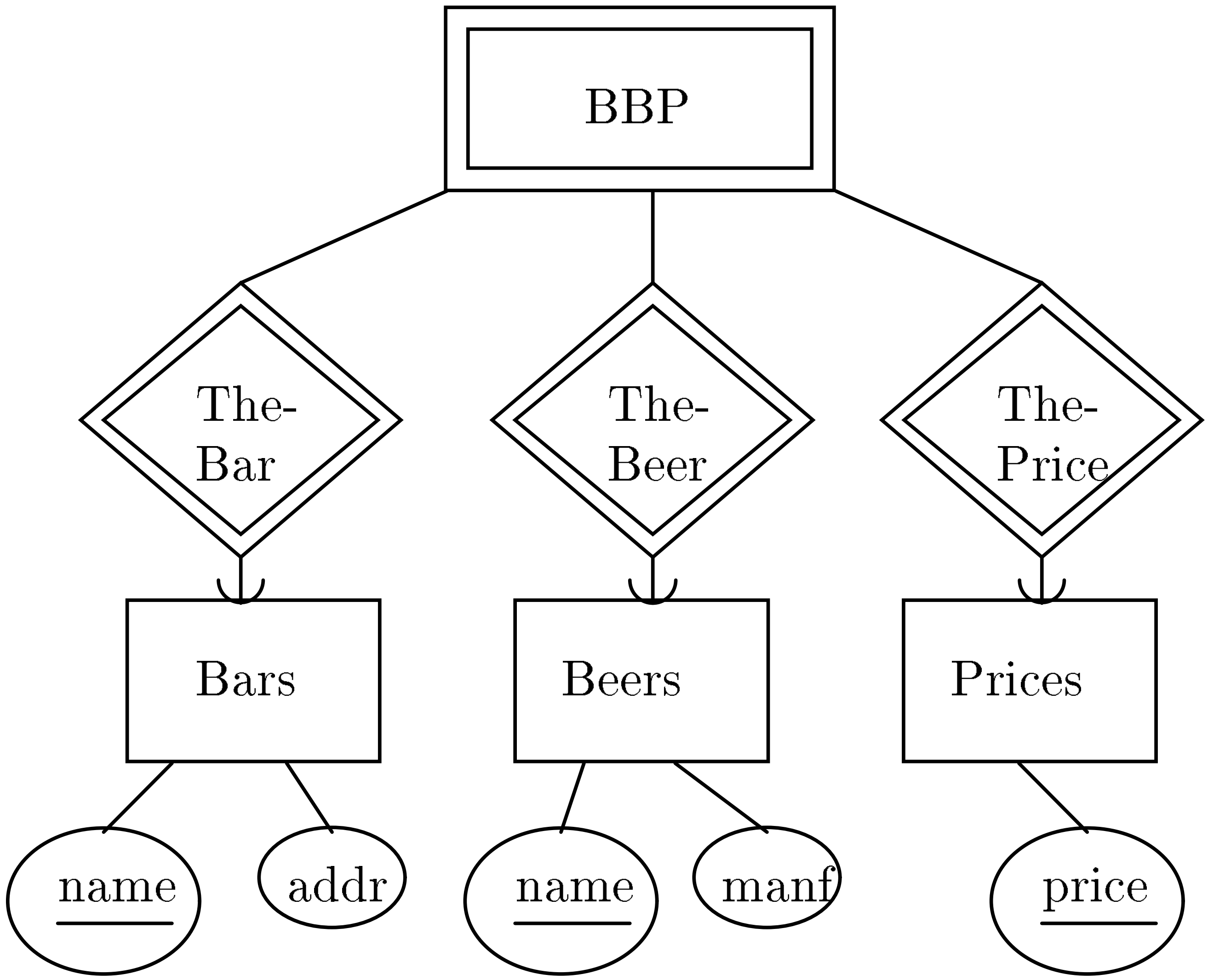
Example: Chain of "Weakness"

Consider IP addresses consisting of a primary domain (e.g., `edu`) subdomain (e.g., `stanford`), and host (e.g. `shalmaneser`).



- Key for primary domain = its name.
- Key for secondary domain = its name + name of primary domain.
- Key for host = its name + key of secondary domain = its name + name of secondary domain + name of primary domain.

All "Connecting" Entity Sets Are Weak



- In this special case, where bar and beer determine a price, we can omit `price` from the key, and remove the double diamond from `ThePrice`.
- Better: `price` is attribute of `BBP`.

Design Principles

Setting: client has (possibly vague) idea of what he/she wants. You must design a database that represents these thoughts and only these thoughts.

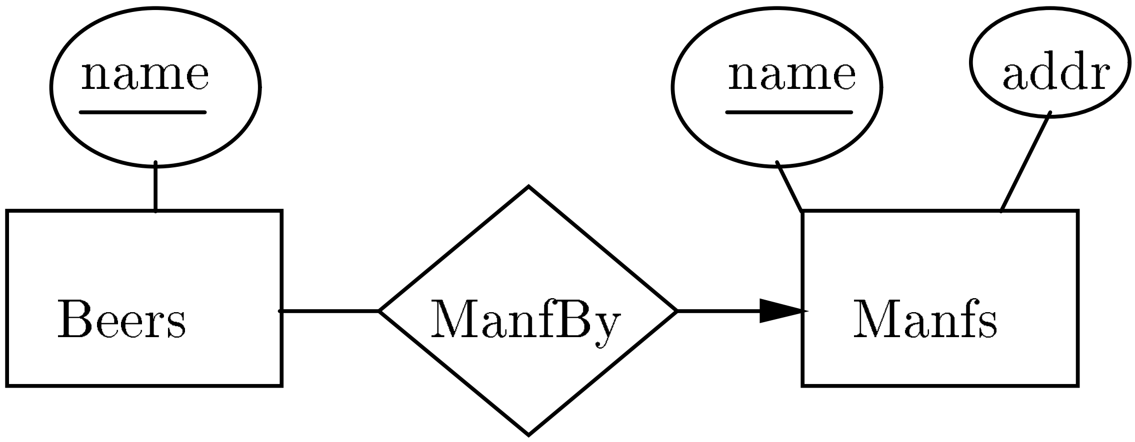
Avoid Redundancy

= saying the same thing more than once.

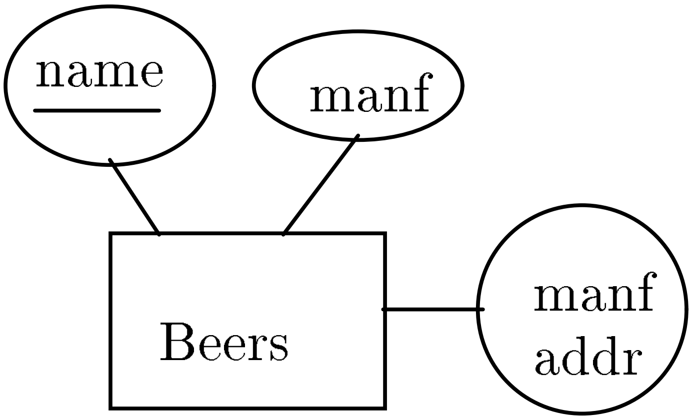
- Wastes space and encourages inconsistency.

Example

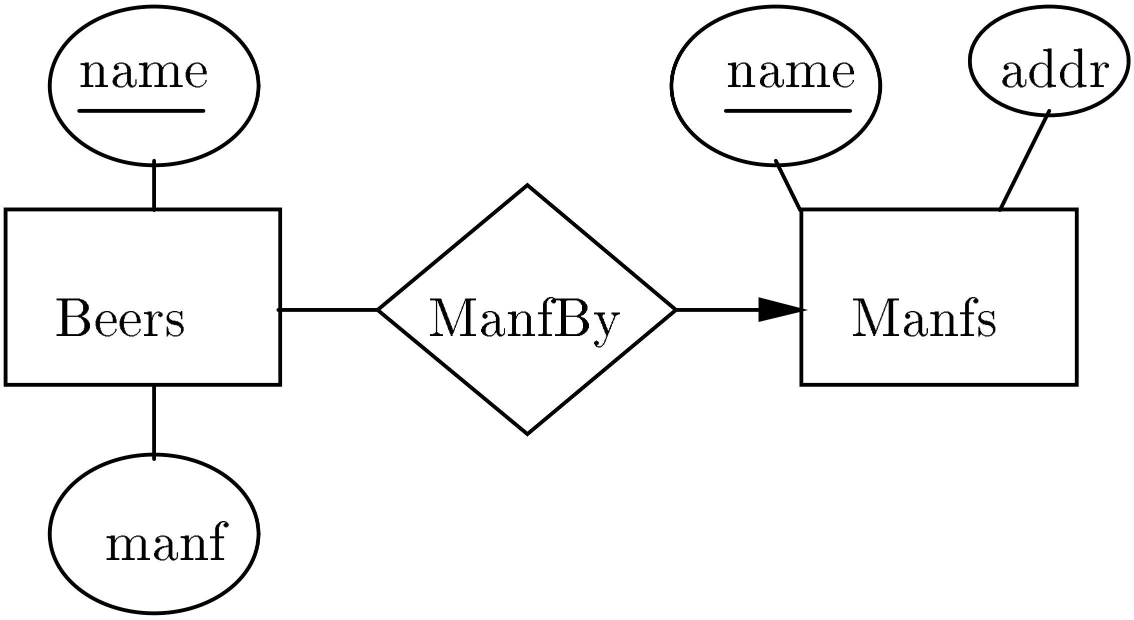
Good:



Bad: repeats manufacturer address for each beer they manufacture.



Bad: manufacturer's name said twice.



Use Schema to Enforce Constraints

The design *schema* should enforce as many constraints as possible.

- Don't rely on users to follow assumptions.

Example

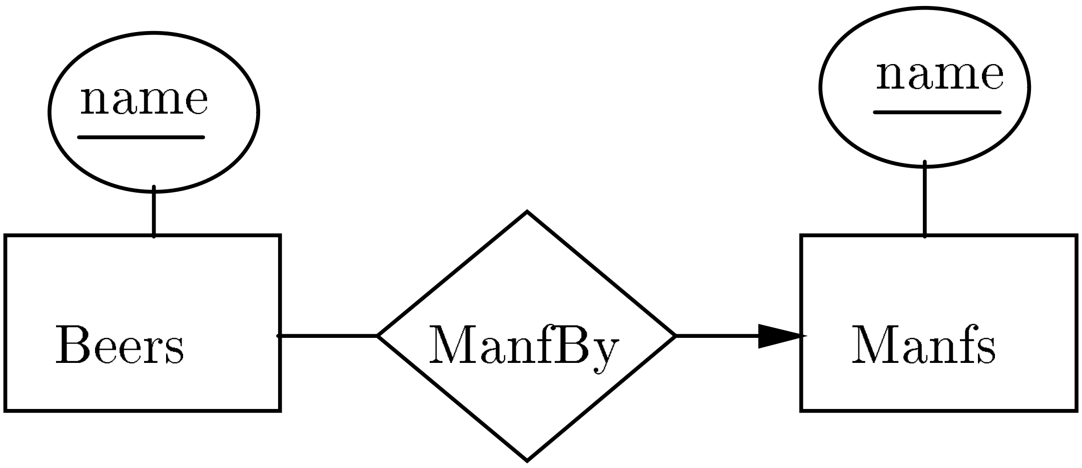
If registrar wants to associate only one instructor with a course, don't allow sets of instructors and count on departments to enter only one instructor per course.

Entity Sets Vs. Attributes

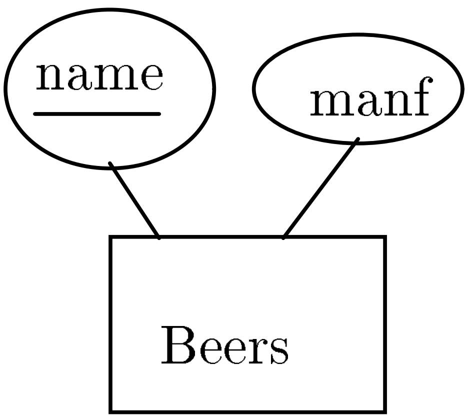
You may be unsure which concepts are worthy of being entity sets, and which are handled more simply as attributes.

- Especially tricky for the class design project, since there is a temptation to create needless entity sets to make project "larger."

Wrong:



Right:



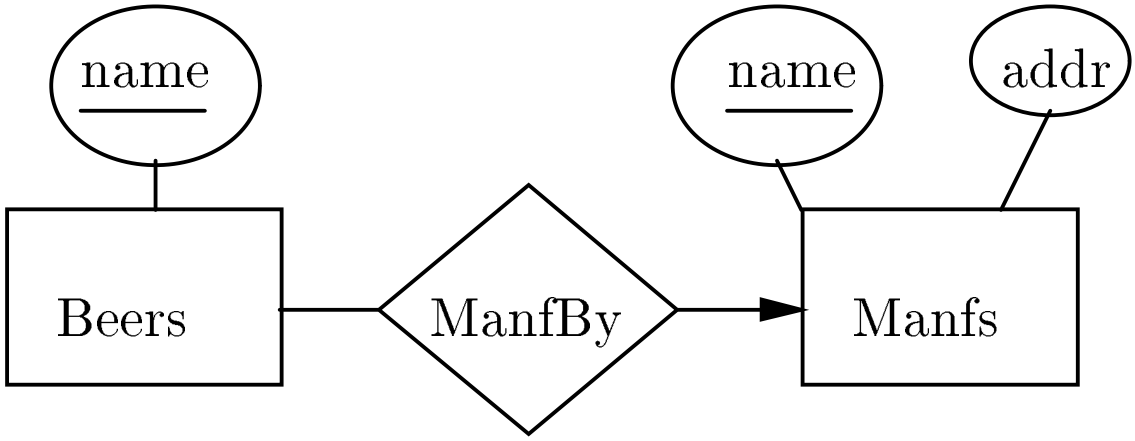
Intuitive Rule for E.S. Vs. Attribute

Make an entity set only if it either:

1. Is more than a name of something; i.e., it has nonkey attributes or relationships with a number of different entity sets, or
2. Is the “many” in a many-one relationship.

Example

The following design illustrates both points:



- *Manfs* deserves to be an E.S. because we record *addr*, a nonkey attribute.
- *Beers* deserves to be an E.S. because it is at the “many” end.
    - ❖ If not, we would have to make “set of beers” an attribute of *Manfs* — something we avoid doing, although some may tell you it is OK in E/R model.

Don't Overuse Weak E.S.

- There is a tendency to feel that no E.S. has its entities uniquely determined without following some relationships.
- However, in practice, we almost always create unique ID's to compensate: social-security numbers, VIN's, etc.
- The only times weak E.S.'s seem necessary are when:

  a) We can't easily create such ID's; e.g., no one is going to accept a "species ID" as part of the standard nomenclature (species is a weak E.S. supported by membership in a genus).

  b) There is no global authority to create them, e.g., crews and studios.

Classroom Design Exercise

Imagine we are creating a database for a dorm, which includes a cooperative kitchen.

- We want to record certain information about each resident. What?
- Not all residents belong to the kitchen coop. Those that do interact in various ways:
    1. They take turns at various jobs: preparer, cleanup, buyer (for supplies). No one should have two jobs on one day.
    2. They may or may not be vegetarian. Each meal must have at least one vegetarian entry.
    3. They pay fees to the coop.
- For each meal, there is a menu. Each menu item requires certain ingredients, which must be on hand.

If There's Time $\cdots$

Suppose we only need to have a vegetarian choice for a given meal if there is at least one vegetarian taking that meal. How would we modify the database schema?