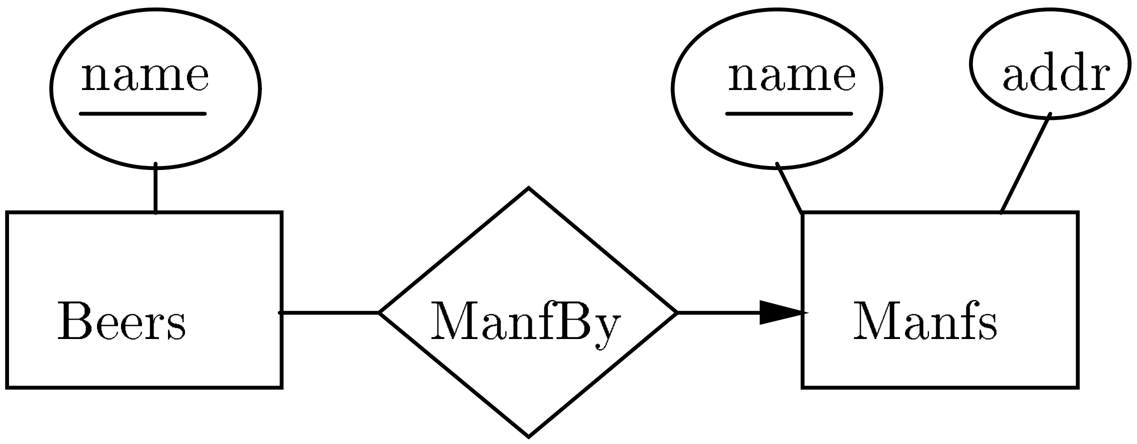
Design Principles

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

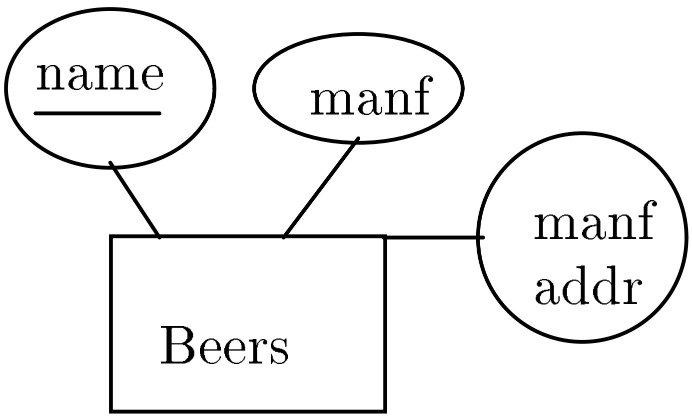
- Avoid redundancy.
  - Wastes space and encourages inconsistency.
- KISS = keep it simple, students.
  - Avoid intermediate concepts.
- Faithfulness to requirements.
  - Remember the design *schema* should enforce as many constraints as possible. Don't rely on future data to follow assumptions.
  - 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.

Example

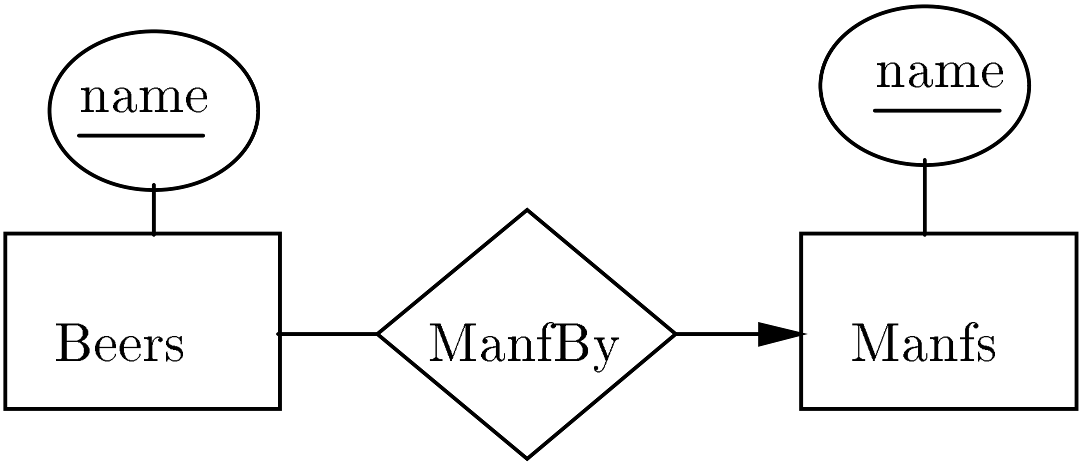
Good:



Bad (redundancy): repeats manufacturer address for each beer they manufacture.



Bad (needless intermediate):



- Question: Why is it OK to have *Beers* with just its key as attribute? Why not make set of beers an attribute of manufacturers?
    - Questionable in E/R, but clearly legal in ODL:

```
interface Manfs (key name) {
    attribute string name;
    attribute string addr;
    attribute Set<string> beersManfed;
}
```

A Design Problem

We wish to design a database representing cities, counties, and states in the US.

- For states, we wish to record the name, population, and state capital (which is a city).
- For counties, we wish to record the name, the population, and the state in which it is located.
- For cities, we wish to record the name, the population, the state in which it is located and the county or counties in which it is located.

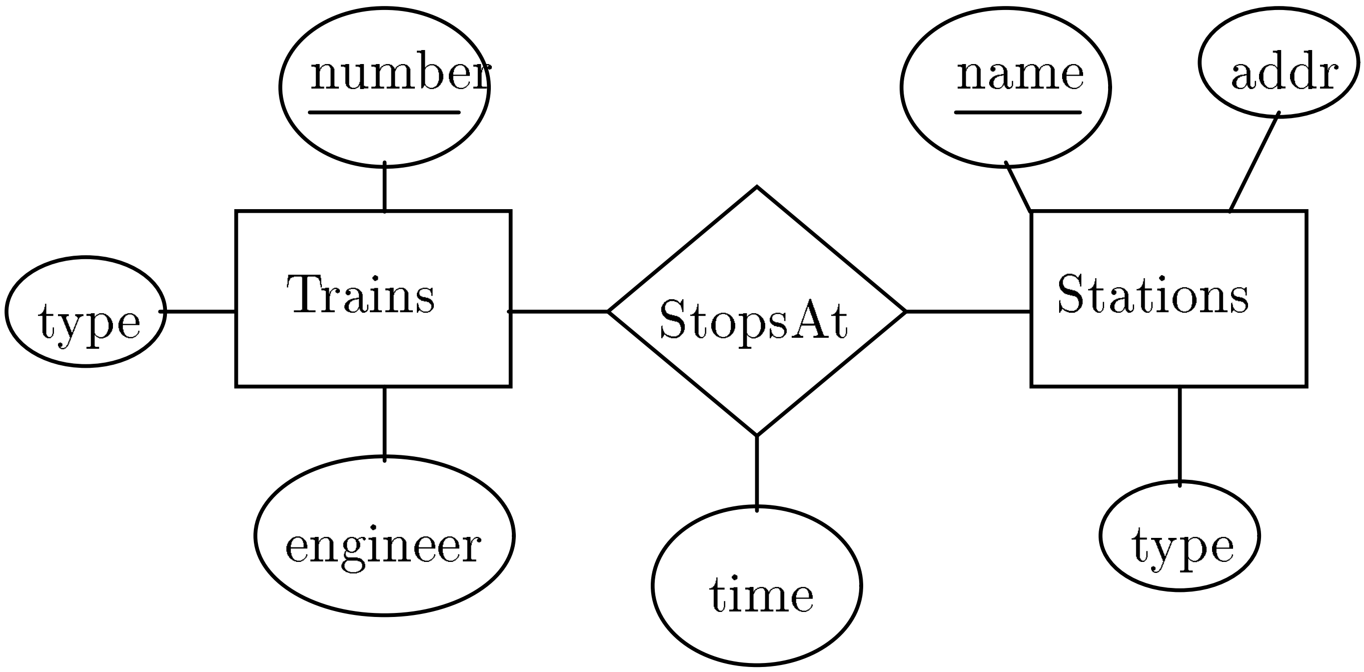
- Uniqueness assumptions:
  - Names of states are unique.
  - Names of counties are only unique within a state (e.g., 26 states have Washington Counties).
  - Cities are likewise unique only within a state (e.g., there are 24 Springfields among the 50 states).
  - Some counties and cities have the same name, even within a state (example: San Francisco).
  - Almost all cities are located within a single county, but some (e.g., New York City) extend over several counties.

Another Design Problem

We wish to design a database consistent with the following facts.

- Trains are either local trains or express trains, but never both.
- A train has a unique number and an engineer.
- Stations are either express stops or local stops, but never both.
- A station has a name (assumed unique) and an address.
- All local trains stop at all stations.
- Express trains stop only at express stations.
- For each train and each station the train stops at, there is a time.

Strawman: What's Wrong With This?



In the Begining . . .

(Historical data models: Network and Hierarchical)

- The mid 1960's saw the first systems that used secondary storage for *querying* = retrieval by value, not by file name.

- Big difference: *secondary storage model of data* = data in blocks/pages, and major cost is retrieving or storing a *block*.

- *Unsolved problem*: Storing many-many relationships so they can be traversed efficiently in both directions.

    - ❖ Easy in RAM model: linked lists of successors, predecessors, e.g.

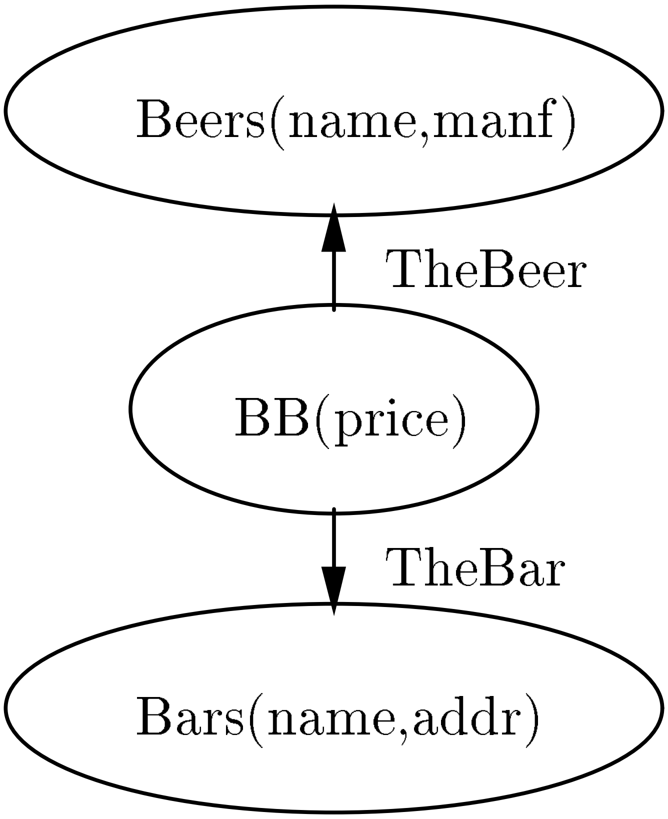
    - ❖ Many-one is easy in secondary-storage model, e.g., store each beer following its manufacturer, so "find the Anheuser-Busch beers" can be answered by retrieving them all on one or a few blocks.

. . .	A.B.	BudLite
Bud	Michelob	. . .

Network Model

Essentially entity sets and binary, many-one relationships.

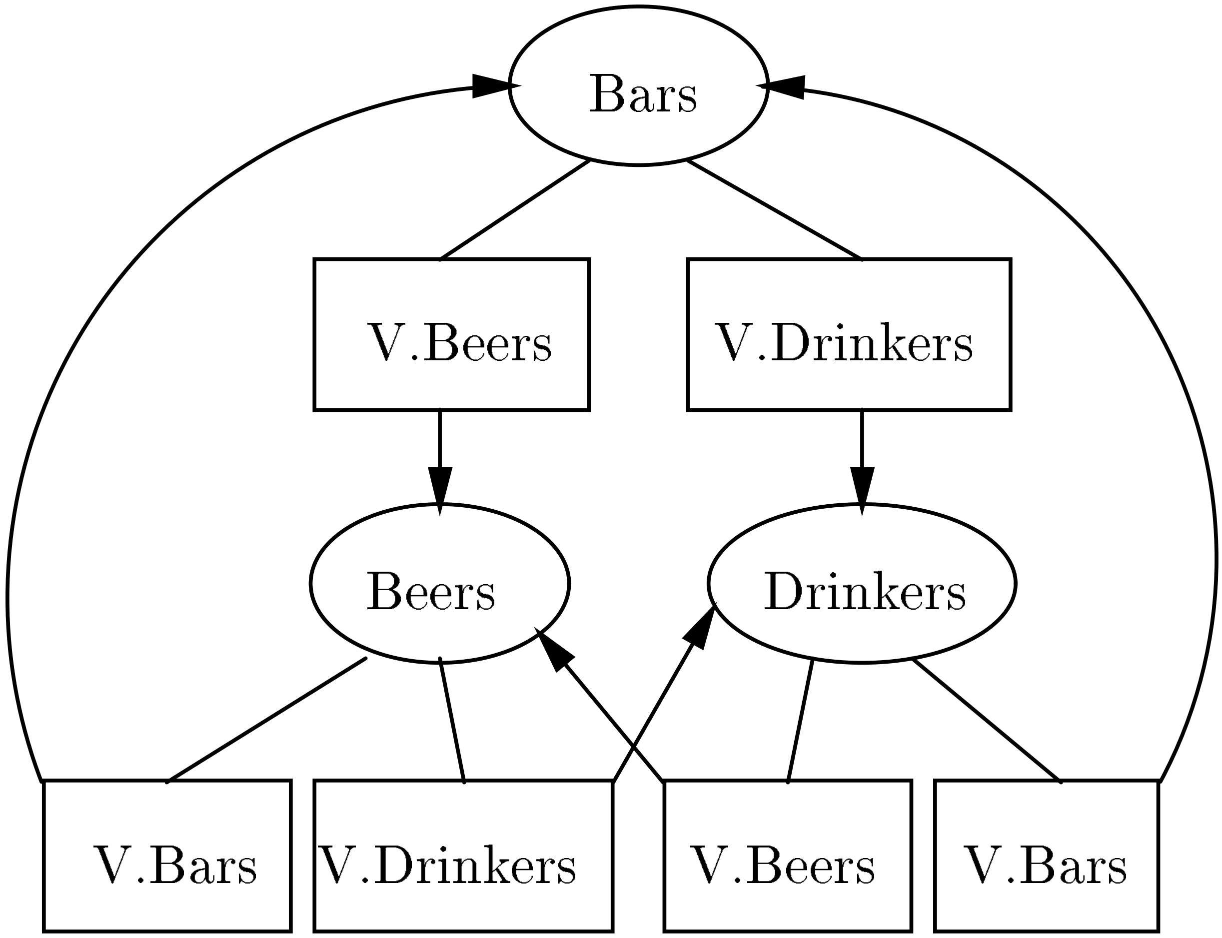
- Replace a many-many relationship by a connecting E.S. and two many-one relationships.
- Entity set → *Logical Record Type* (LRT).
- Many-one relationship → *Link*.
  - ❖ Terminology useful to this day: *owner* = one, *member* = many, e.g., a manufacturer record "owns" beer records.



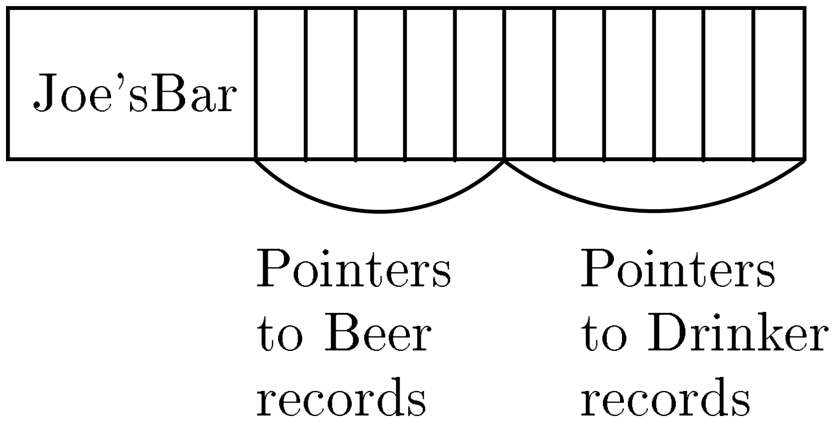
Hierarchical Model

Used in major early DBMS's, including IBM's IMS, which is still supported today.

- Network model, restricted to a forest, where owners are parents of children.
- Adds *Virtual LRT* to handle many-many relationships.
    - ❖ Think of V. LRT as representing pointers.

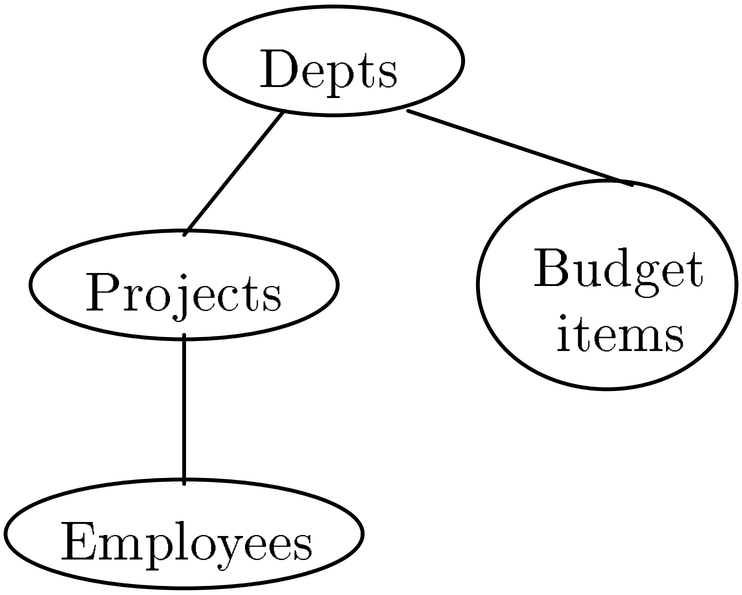


Typical Bar Record



- No help in secondary storage model when going from a bar to either its beers or its drinkers.

Example Where Hierarchical Model Wins



- Typical stored records make efficient queries that go Dept → Budget Item or Project → Employees.