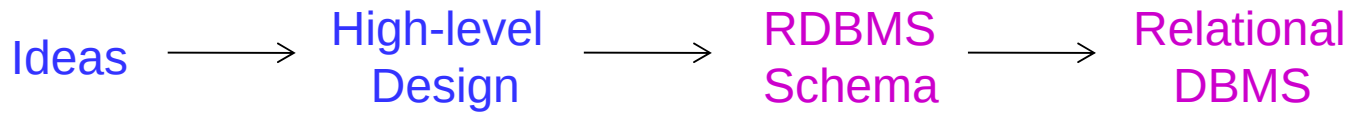
Database Management Systems - I, CS 157A

Logical Database Model: Entity Relationship Model

Ch. 4

High-level RDBMS Models



- High-level design is intended to cover what information need to be stored, what constraints such as keys & referential integrity may be needed, etc.
- This session is intended to cover multiple high-level design models (logical models) including:
  - Entity/Relationship Diagram Model - E/R
  - Unified Modeling Language - UML
  - Object Description Language – ODL
- Map this logical model into the physical model (relational model – relations)

Purpose of E/R Model

- The E/R model allows us to sketch database schema designs:
  - Includes some constraints, but not operations
- Designs are pictures called *entity-relationship diagrams*
- Later: convert E/R designs to relational DB designs (i.e., schema)

Framework for E/R

- Design is a serious business
- The “boss” knows they want a database, but they don’t know what they want in it
- Sketching the key components is an efficient way to develop a working database

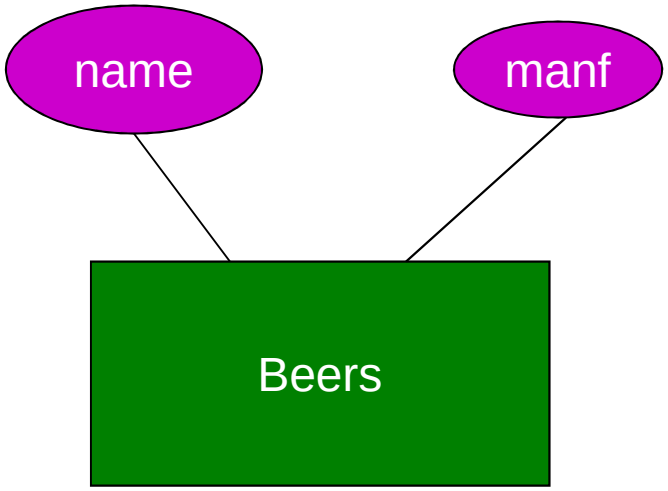
Entity Sets

- ***Entity*** = “thing” or object (tuple)
- ***Entity set*** = collection of similar entities; typically implemented as a relation (relation)
  - Similar to a class in object-oriented languages.
- ***Attribute*** = property of (the entities of) an entity set:
  - Attributes are simple values, e.g. integers or character strings, can be structs, sets, etc.
- **Relationships** = connection between two or more entity sets – binary or multi-way relationships

E/R Diagrams

- **In an entity-relationship diagram:**
  - Entity set = rectangle
  - Attribute = oval, with a line to the rectangle representing its entity set
  - Relationship = diamond, connecting two or more entity sets

Example:

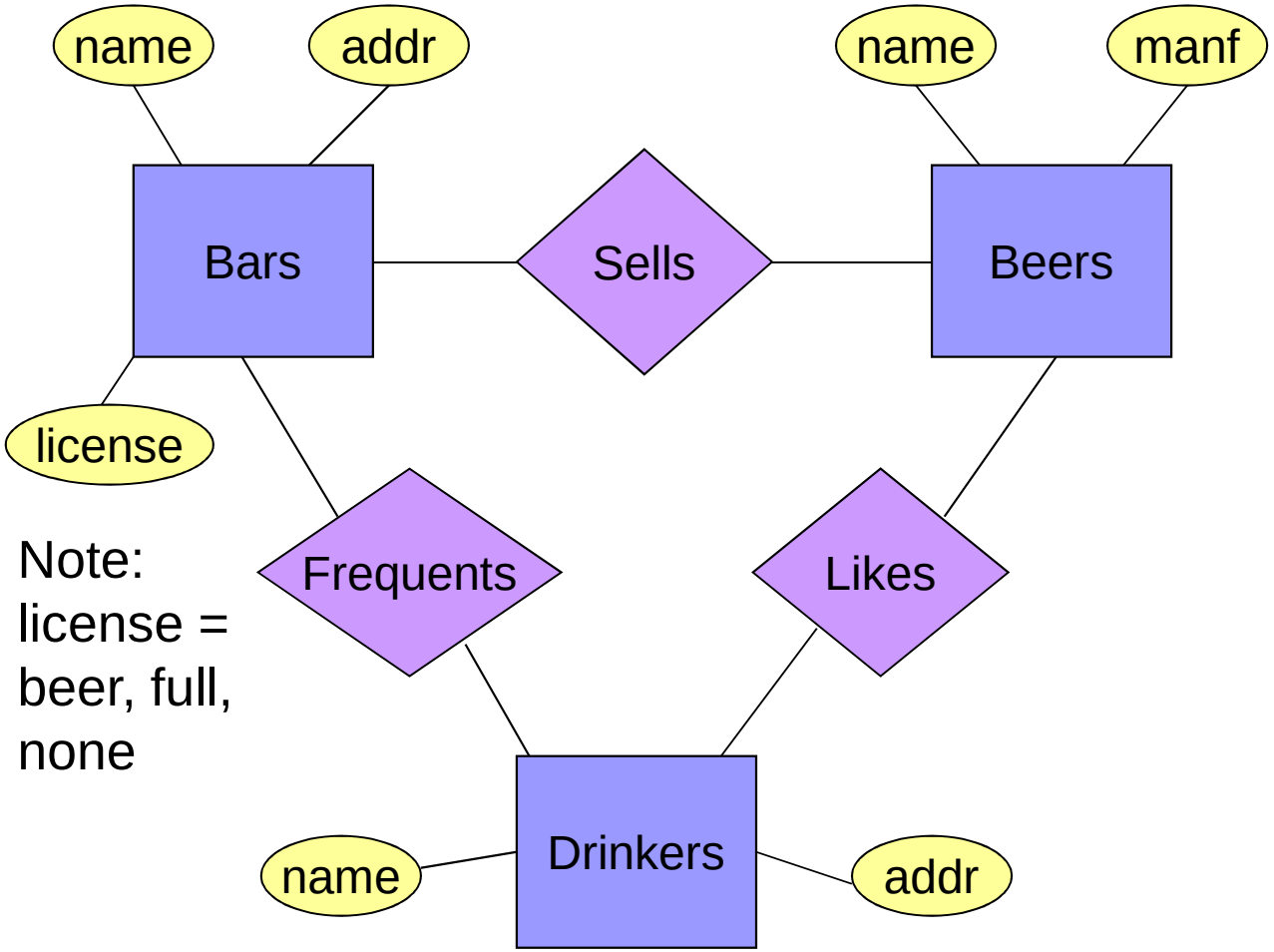


- Entity set Beers has two attributes, name and manf (manufacturer)
- Each Beers entity (tuple) has values for these two attributes, e.g. (Bud, Anheuser-Busch)

Relationships

- **A relationship** connects two or more entity sets
- Most common is binary relationship. Ternary (3-way) or more is rare
- It is represented by a diamond, with lines to each of the entity sets involved
- Relationship can be bi-directional

Example: Relationships



Note: license = beer, full, none

Bars sell some beers.

Drinkers like some beers.

Drinkers frequent some bars.

Relationship Set

- The current “value” of an entity set (relation) is the set of entities (tuples) that belong to it
  - **Example:** the set of all bars in our database.
- The “value” of a relationship is a ***relationship set***, a set of tuples with one tuple for each related entities from the relevant entity sets.

Example: Relationship Set

- For the relationship Sells between Bars and Beers, we might have a relationship set like:

Bar	Beer
Joe's Bar	Bud
Joe's Bar	Miller
Sue's Bar	Bud
Sue's Bar	Pete's Ale
Sue's Bar	Bud Lite

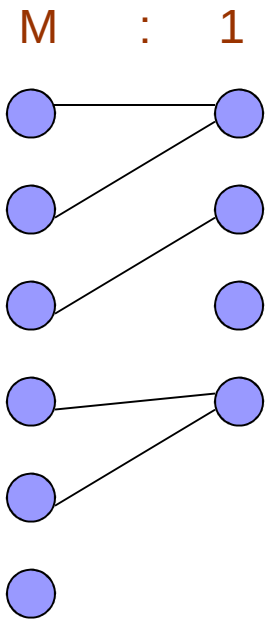
Many-One & One-Many Relationships

- In a **many-one** (E-F) relationship R connecting entity sets E and F: each entity in E is connected to one entity in F. Member in F can be connected to 0, 1, or many entities in E

- In a **one-many** (E-F) relationship R connecting entity sets E and F: an entity in E is connected to 0, 1, or many entities in F. Member in F is connected to one entity in E

- If R is **many-one** from E to F, we can say R is **one-many** from F to E

In Pictures:



Binary Relationship: many-one

Example: Many-One Relationship

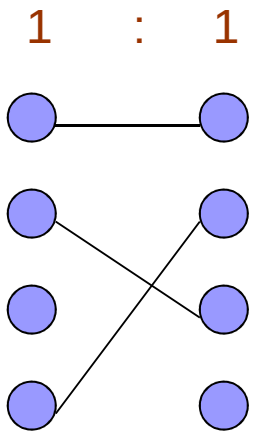
- **Favorite**, from Drinkers to Beers is many-one
- A drinker has at most one favorite beer
- But a beer can be the favorite of any number of drinkers, including zero

- P.S. There is no requirement however that a specific Beer will need to have a Drinker? To ensure that we will need a referential integrity constraint

One-One Relationship

- If R is both many-one from E to F and many-one from F to E, then we say R is one-one. In one-one relationship an entity in E or F can be connected to at most (i.e., 0 or 1) one entity of the other set

In Pictures:



Binary Relationship: one-one

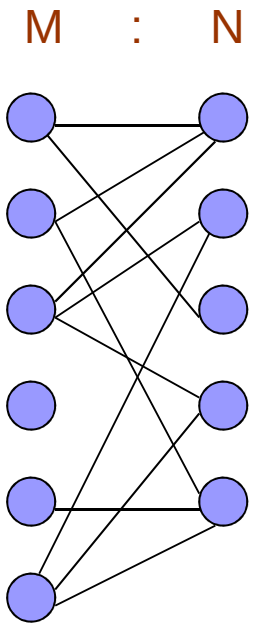
One-One Relationships

- **Example:** Relationship Best-seller between entity sets Manfs (manufacturer) and Beers
  - A beer cannot be made by more than one manufacturer, and no manufacturer can have more than one best-seller (assume no ties)

Many-Many Relationships

- In a ***many-many* relationship**, an entity of either set can be connected to many entities of the other set

In Pictures:



Binary Relationship: many-many

Many-Many Relationships

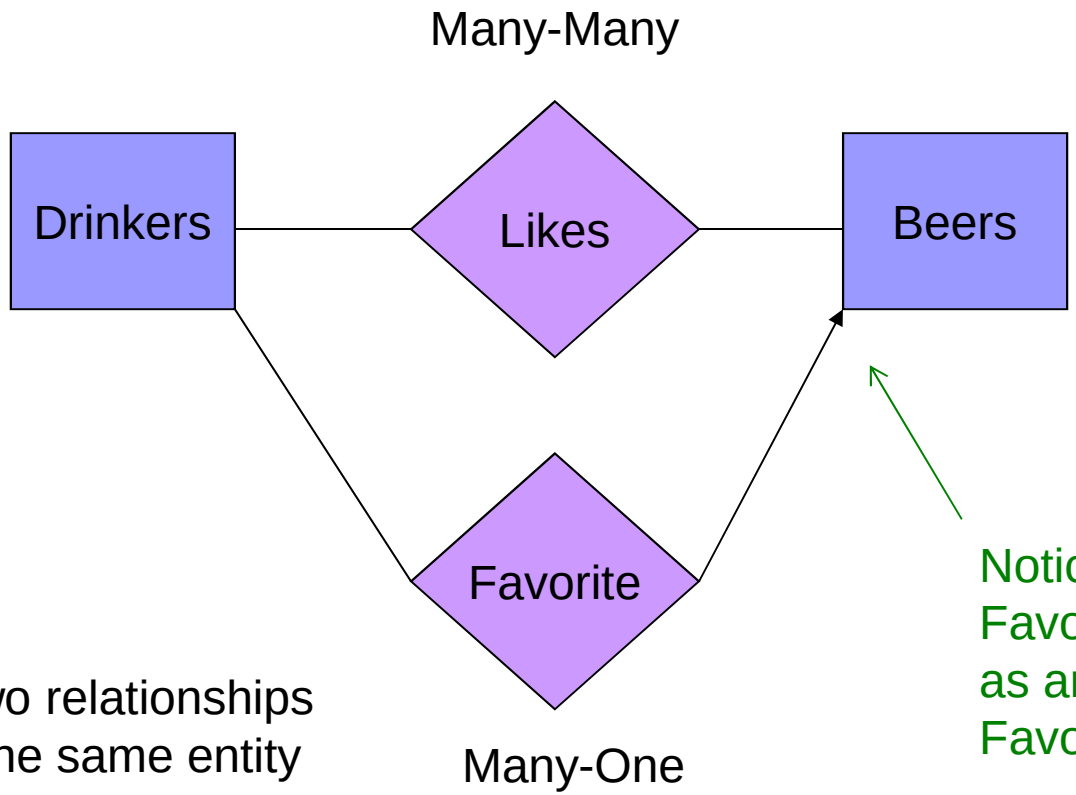
- **Example:** Relationship Sells between Bars and Beers
  - A Bar sells many Beer, and a Beer is sold by many Bars

- **Observation:** many-one is a special case of many-many, and one-one is a special case of a many-one relationship

Representing “Multiplicity”

- Show a many-one relationship by an arrow entering the “one” side
  - Remember: Like a functional dependency
- Show a one-one relationship by arrows entering both entity sets – an arrow means “at most one”
- Rounded arrow $\rightarrow$ equal “exactly one,” i.e., each entity of the first set is related to exactly one entity of the target set
- An arrow connecting relationship to an entity set serves as a “functional dependency”

Example: Many-One Relationship



Notice: two relationships connect the same entity sets, but are different.

Notice: This arrow from Favorite to Beers serves as an FD (Drinkers ➔ Favorite_Beer).

Example: One-One Relationship

- Consider **Best-seller** between Manfs and Beers
- Some beers are not the best-seller of any manufacturer, so a rounded arrow to Manfs would be inappropriate
- But a beer manufacturer has to have a best-seller

In the E/R Diagram

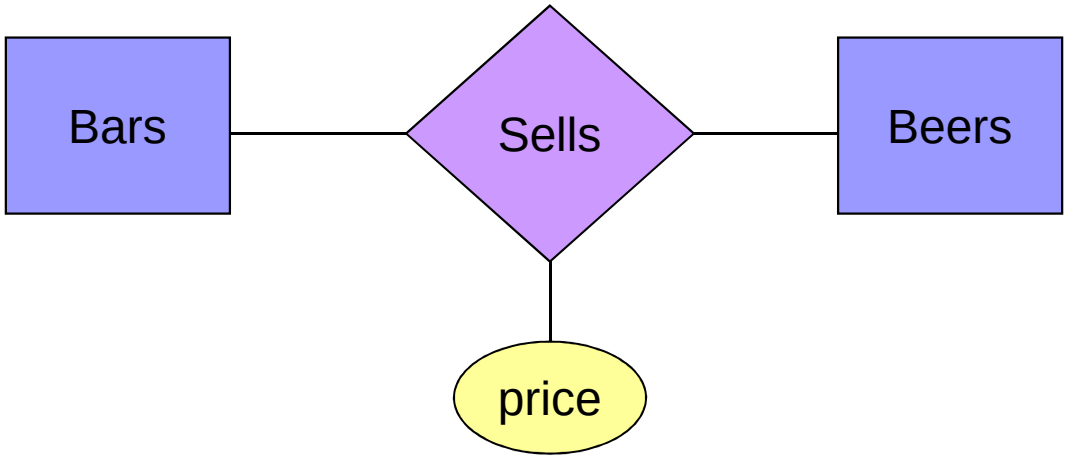
A beer is the best-seller for “0 or 1” manufacturer.

A manufacturer has “exactly one” best seller.

Attributes on Relationships

- Sometimes it is useful to attach an attribute to a relationship, i.e., not to an entity set
- Think of this attribute as a property of tuples in the relationship set

Example: Attribute on Relationship

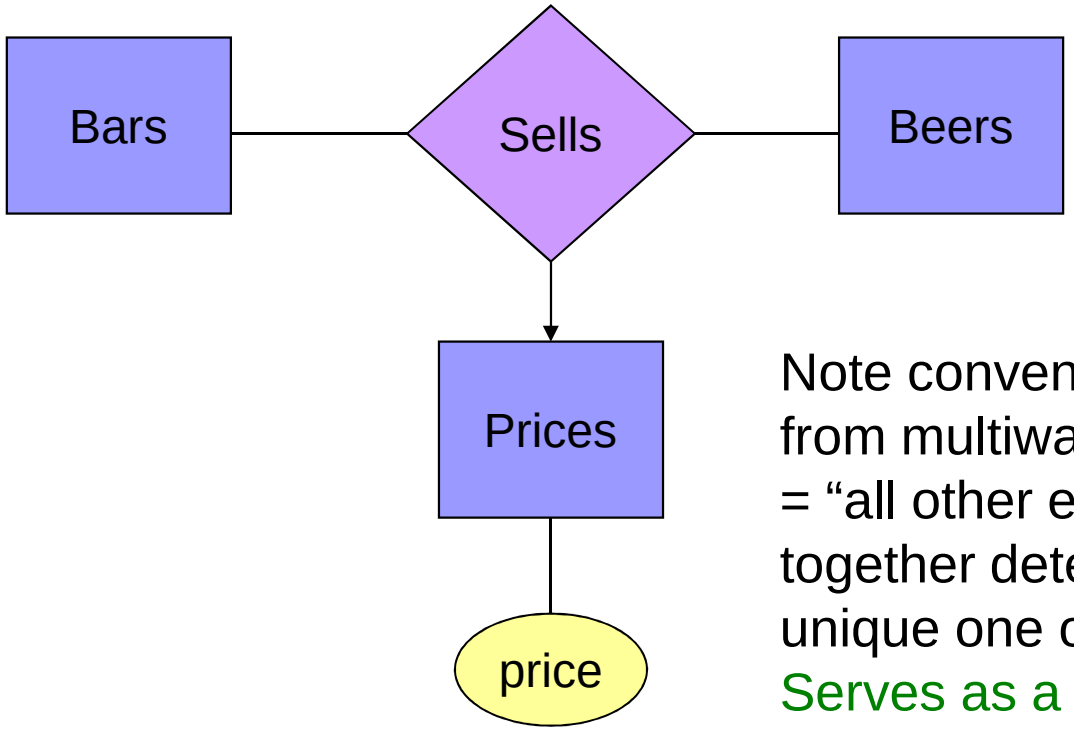


Price is a function of both the bar and the beer, not of one alone.

Equivalent Diagrams Without Attributes on Relationships

- Create an entity set representing values of the attribute
- Make that entity set participate in the relationship

Example: Removing an Attribute from a Relationship

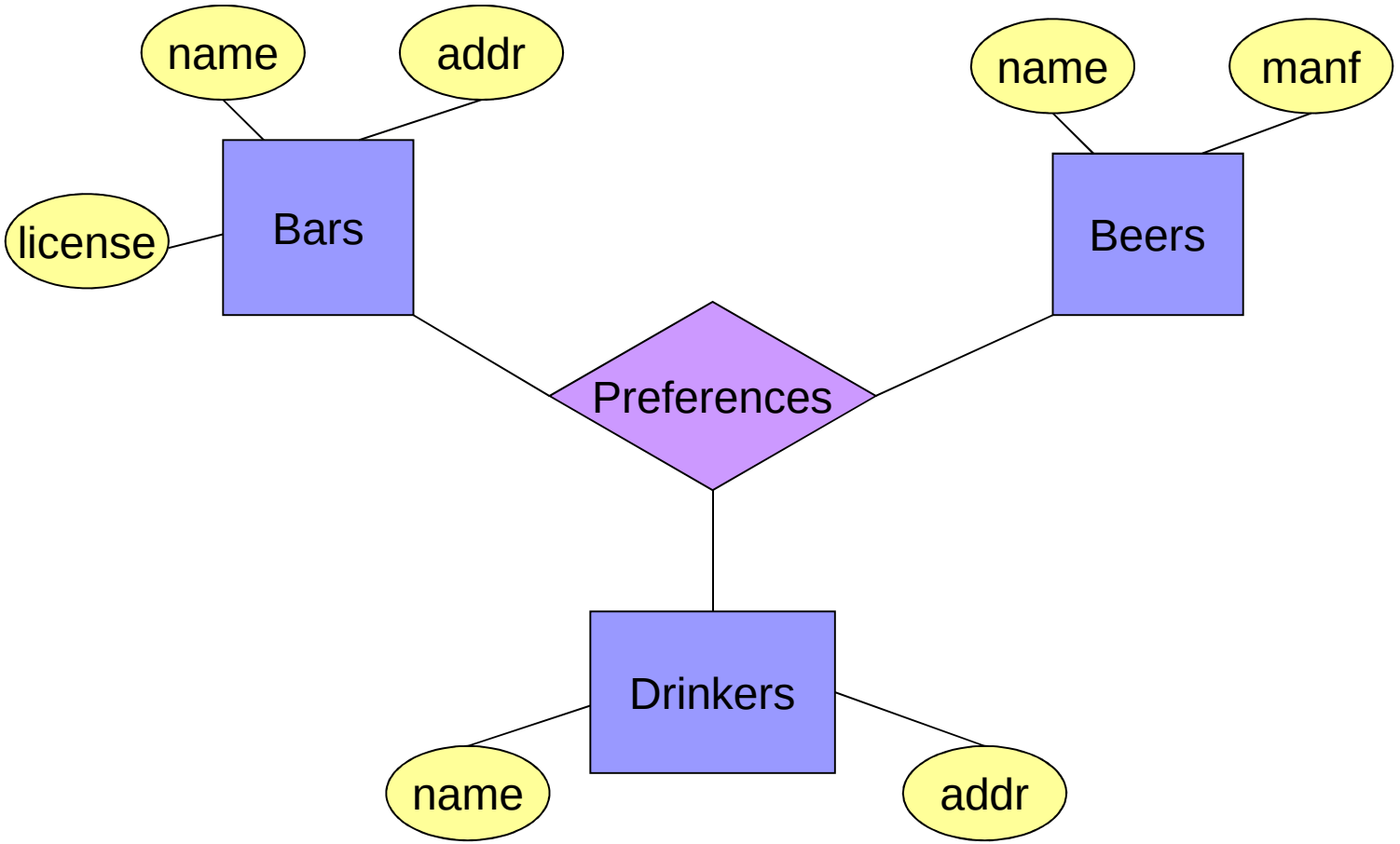


Note convention: arrow from multiway relationship = "all other entity sets together determine a unique one of these."
Serves as a "functional dependency" –
(Bar, Beer ➔ price)

Multi-way Relationships

- Sometimes, we need a relationship that connects more than two entity sets
- Suppose that drinkers will only drink certain beers at certain bars
  - Our three binary relationships Likes, Sells, and Frequents do not allow us to make this distinction
  - But a 3-way relationship would

Example: 3-Way Relationship



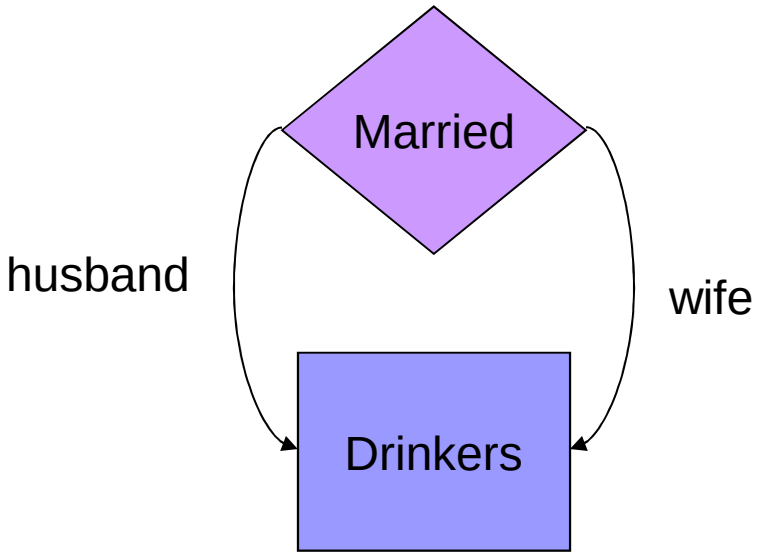
A Typical Relationship Set

Bar	Drinker	Beer
Joe's Bar	Ann	Miller
Sue's Bar	Ann	Bud
Sue's Bar	Ann	Pete's Ale
Joe's Bar	Bob	Bud
Joe's Bar	Bob	Miller
Joe's Bar	Cal	Miller
Sue's Bar	Cal	Bud Lite

Roles

- Sometimes an entity set appears more than once in a relationship
- Label the edges between the relationship and the entity set with names called *roles*

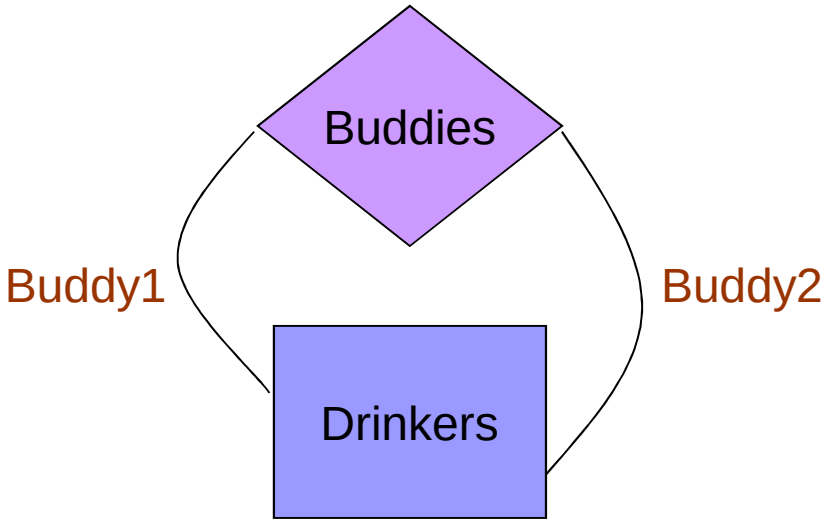
Example: Roles



Relationship Set

Husband	Wife
Bob	Ann
Joe	Sue
…	…

Example: Roles



Relationship Set

Buddy1	Buddy2
Bob	Ann
Joe	Sue
Ann	Bob
Joe	Moe
…	…

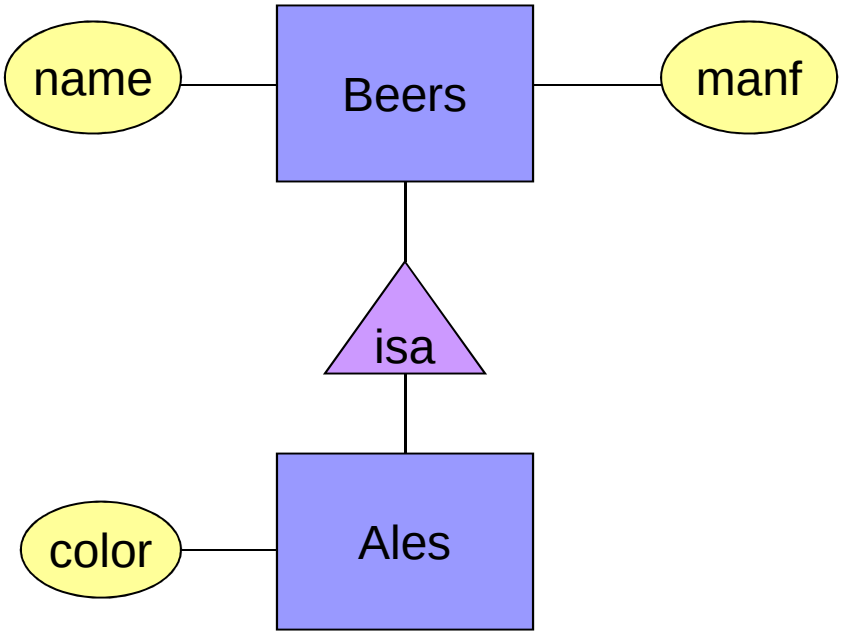
Subclasses

- *Subclass* = special case = fewer entities = more properties
- Example: Ales are a kind of beer
  - Not every beer is an Ales, but some are
  - Let us suppose that in addition to all the *properties* (attributes and relationships) of beers, Ales also have the attribute color

Subclasses in E/R Diagrams

- Assume subclasses form a tree
  - I.e., no multiple inheritance
- isa triangles indicate the subclass relationship of type one-one relationship w/o arrows
  - Point to the superclass

Example: Subclasses



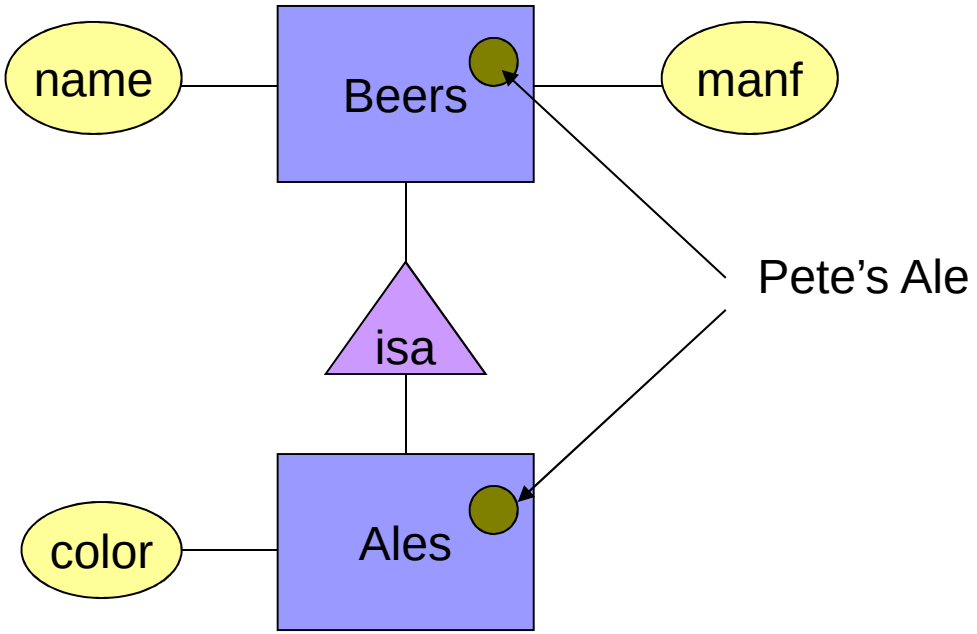
- Ales is a special-case entity set (subclass) of the Beers entity set with its own attributes and relationships
- Ales isa Beers but have special “color” attribute

E/R Vs. Object-Oriented Subclasses

- In OO, objects' state are in one class only
  - Subclasses inherit from superclasses

- In contrast, E/R entities have *representatives* in all subclasses to which they belong
  - **Rule:** for a tree entity set connected by isa relationships, a single entity in a subclass consists of components from that subclass and components in the superclass and recursively up to the root of the tree

Example: Representatives of Entities



Key Constraint

- A ***key*** is a set of attributes for one entity set such that no two entities in this entity set agree on all the attributes of the key:
  - It is allowed for two entities to agree on some, but not all, of the key attributes

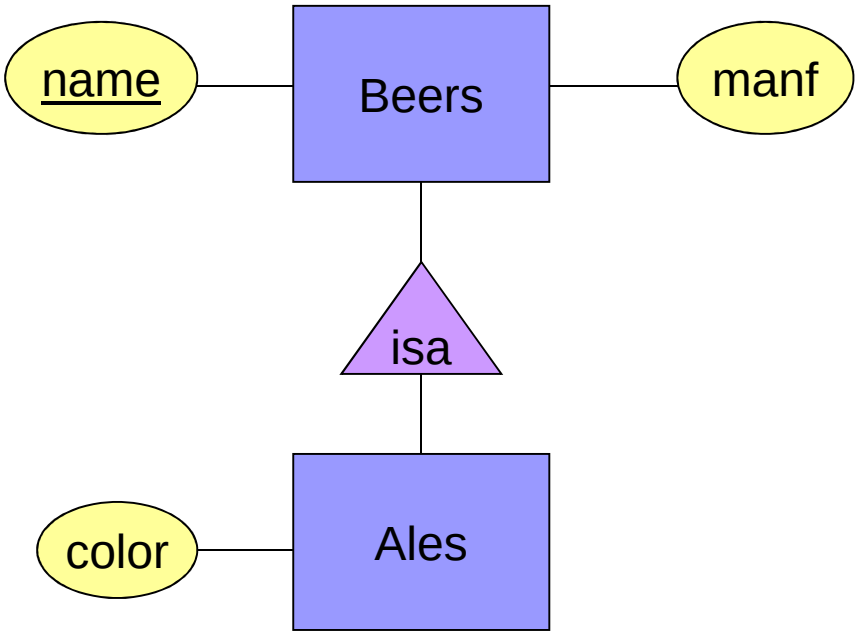
- We must designate a key for every entity set

Keys in E/R Diagrams

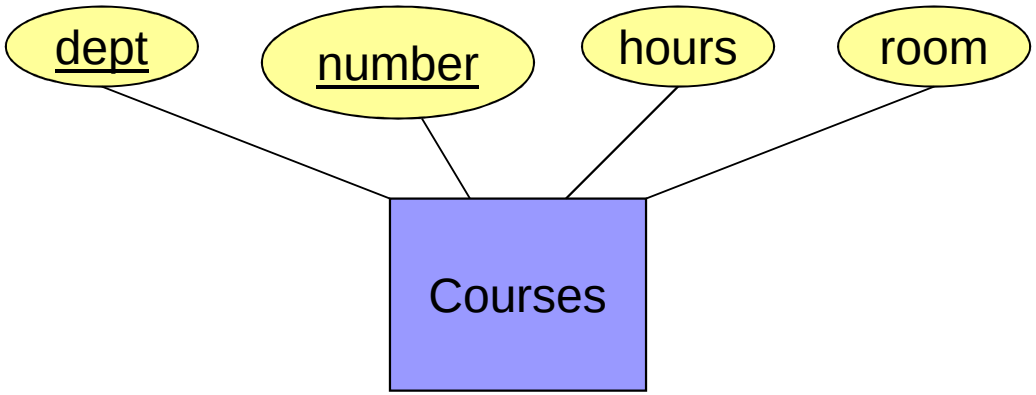
- Underline the key attribute(s)

- In an **isa** hierarchy, only the root entity set has a key (a constraint), and it must serve as the key for all entities in the hierarchy

Example: name is Key for Beers



Example: a Multi-attribute Key

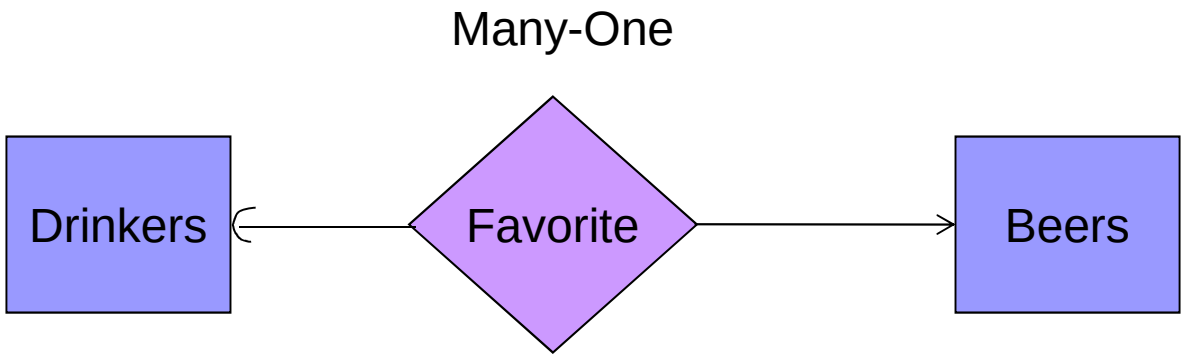


- Note that hours and room could also serve as a key, but we must select only one key.

Referential Integrity Constraints

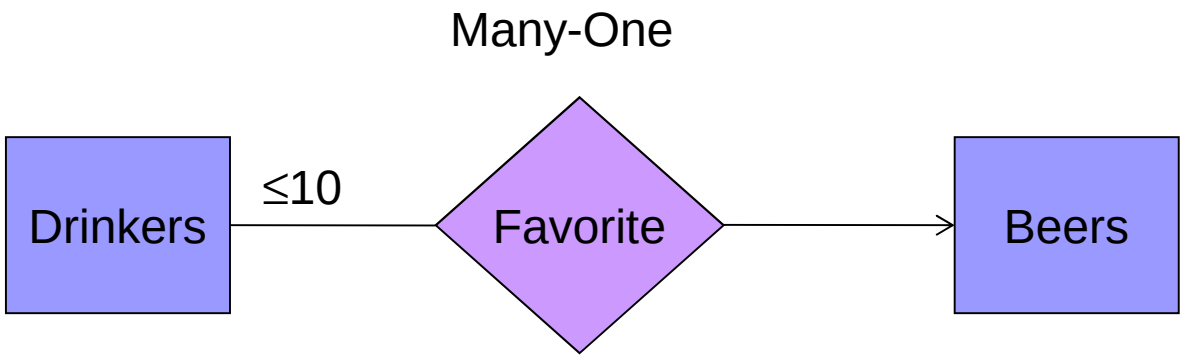
- A ***referential integrity constraint*** states that a value appearing in one context (column) will necessitate another value in a second context (column).
- **In a many-one relationship** like Favorite relationship between Drinkers and Beers. The many-one does not require that given a Beer value we must have a Drinker that this Beer is their favorite!
- A **referential integrity constraint** on the relationship Favorite is that for each Beer, a Drinker (entity referenced by the Favorite relationship) must exist in the database! A Beer must be favorite beer to at least one Drinker.

Referential Integrity Constraints



- Favorite relationship is many-one from entity set Drinkers to entity set Beers
- Arrow to Beers means many drinkers can have same Beer as their favorite
- Rounded arrow (exactly once) to Drinkers means that every Beer must have at least one Favorite Drinker in the database

Degree Constraints



- We can attach a bounding number to the edge connecting a relationship to an entity set
- The attached number indicates limits on the number of entities that can be connected to any single entity of the related entity set
- In the above example: only ≤10 Drinkers can have the same Favorite Beer

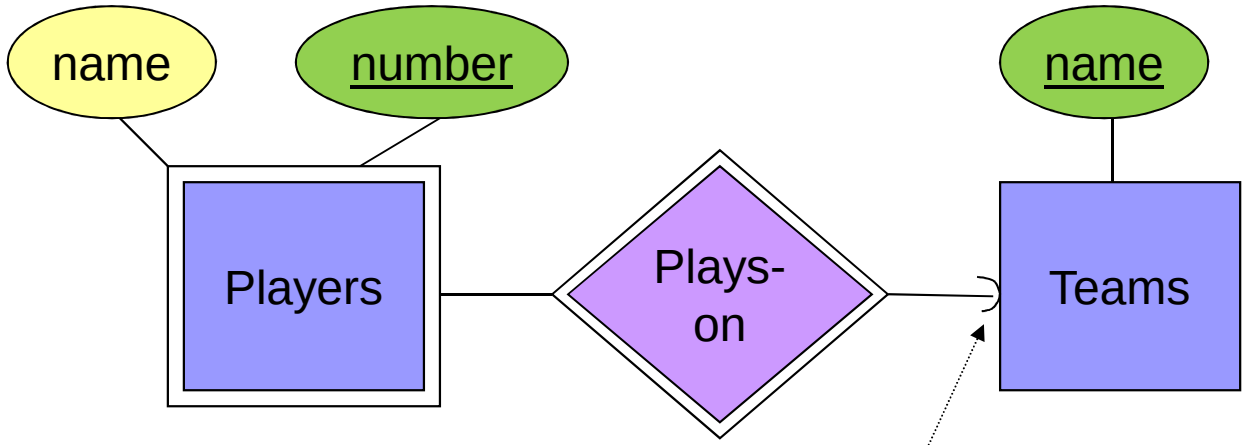
Weak Entity Sets

- Occasionally, entities of an entity set need “help” to identify them uniquely
- **Weak entity** set is an entity set with its key’s attributes, some or all belong to other entity sets
- Entity set E is said to be ***weak*** if in order to identify entities of E uniquely, we need to follow one or more many-one relationships from E and include the key of the related entities from the connected entity sets

Example: Weak Entity Set

- Player **name** is almost a key for football players, but there might be two with the same name
- **number** is certainly not a key, since players on two teams could have the same number
- But **number**, together with the **team name** related to the player by Plays-on should be unique

In E/R Diagrams



Note: “Players” is a week entity set because player name & number are not unique unless associated with a team

Note: must be rounded because a player can be a member of only one team

- To identify player uniquely, we need player number + team’s name as the key!
- Double diamond for *supporting* many-one relationship
- Double rectangle for the weak entity set

Weak Entity-Set Rules

- **A weak entity** set has one or more many-one supporting relationships to other supporting entity sets:
  - Not every many-one relationship from a weak entity set needs to be supporting relationship
  - But supporting relationships must have a rounded arrow (entity at the “one” end needs to be guaranteed)

Weak Entity-Set Rules – (2)

- The key for a weak entity set is its own underlined attributes and the keys for the supporting entity sets:
  - E.g., (player) **number** and (team) **name** is a key for Players in the previous example

Design Techniques

1. Avoid redundancy
2. Limit the use of weak entity sets
3. Don’t use an entity set when an attribute will do

Avoiding Redundancy

- *Redundancy* = saying the same thing in two (or more) different ways
- Wastes space and (more importantly) encourages inconsistency:
  - Two representations of the same fact become inconsistent if we change one and forget to change the other
  - Recall anomalies due to FD's

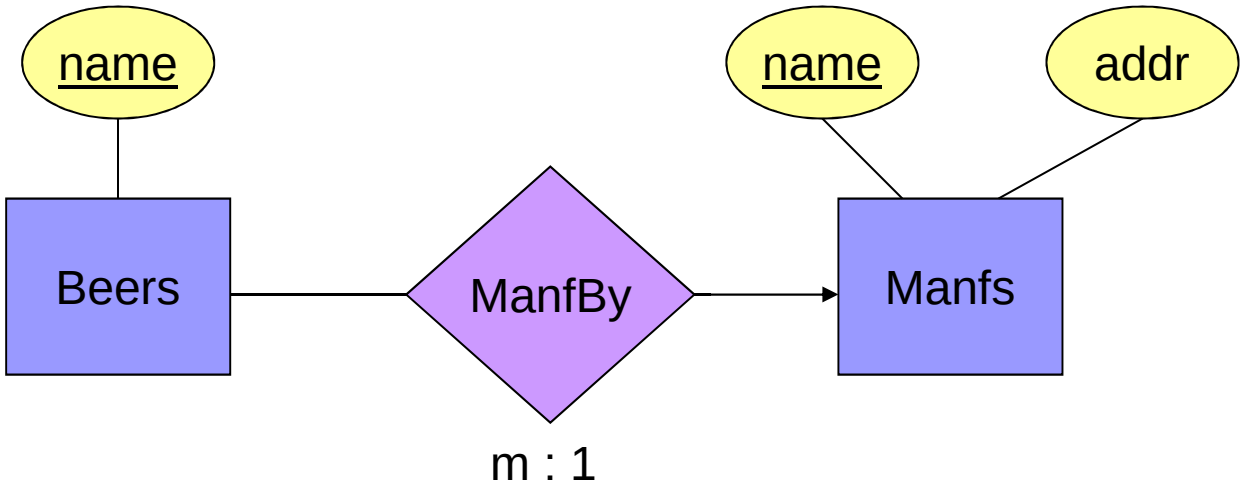
Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one non-key attribute

    or

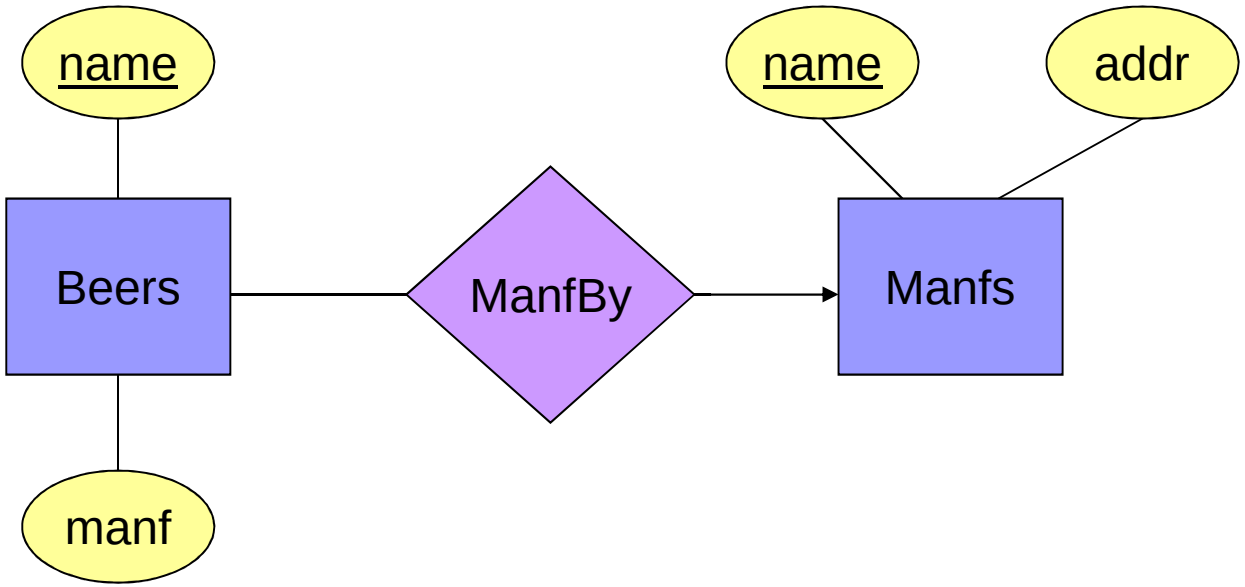
  - It is the "many" in a many-one or many-many relationship

Example: Good



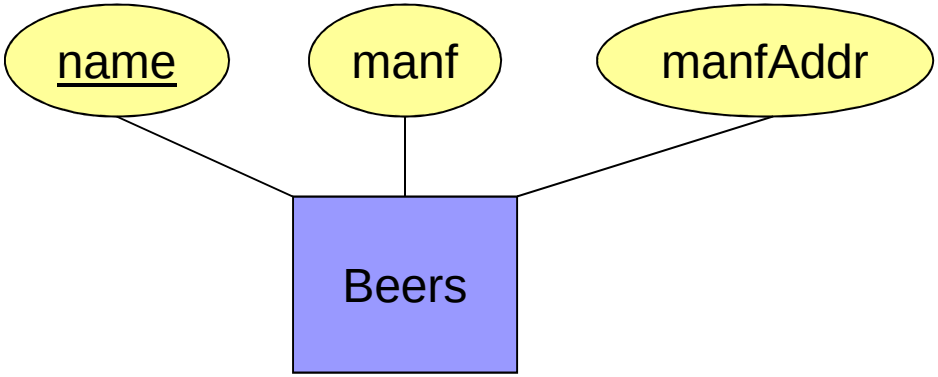
This design gives the address of each manufacturer exactly once

Example: Bad



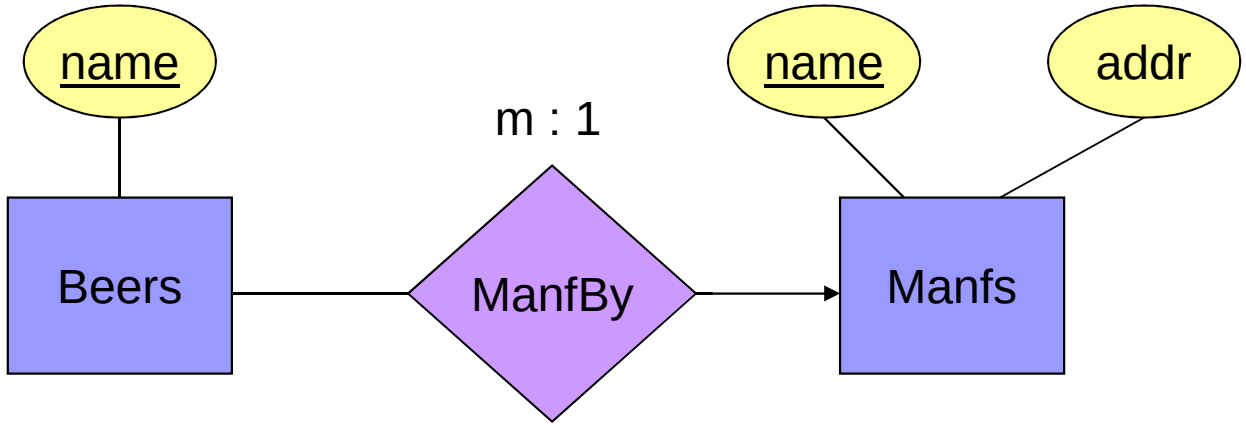
This design states the manufacturer of a beer twice: as an attribute and as a related entity

Example: Bad



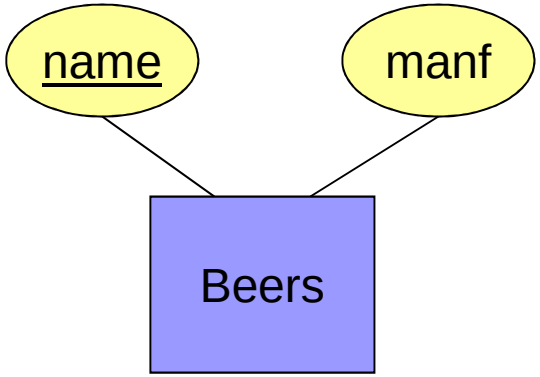
This design repeats the manufacturer's address once for each beer and loses the address if there are temporarily no beers for a manufacturer

Example: Good



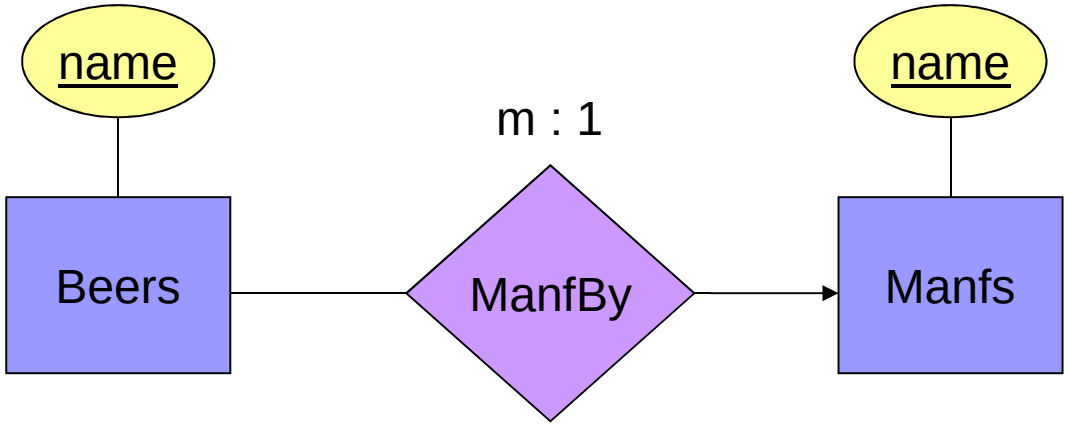
- Manfs deserves to be an entity set because of the non-key attribute addr
- Beers deserves to be an entity set because it is the “many” of the many-one relationship ManfBy

Example: Good



There is no need to make the manufacturer an entity set, because we record nothing about manufacturers besides their name

Example: Bad



Since the manufacturer is nothing but a name (i.e., no additional attributes), and is not at the “many” end of any relationship, it should not be an entity set

Don’t Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself:
  - They make all entity sets weak, supported by all other entity sets to which they are linked
- In reality, we usually create unique ID’s for entity sets:
  - Examples include social-security numbers, automobile VIN’s etc.

When Do We Need Weak Entity Sets?

- The usual reason is that there is no global authority capable of creating unique ID's

- **Example:** it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world

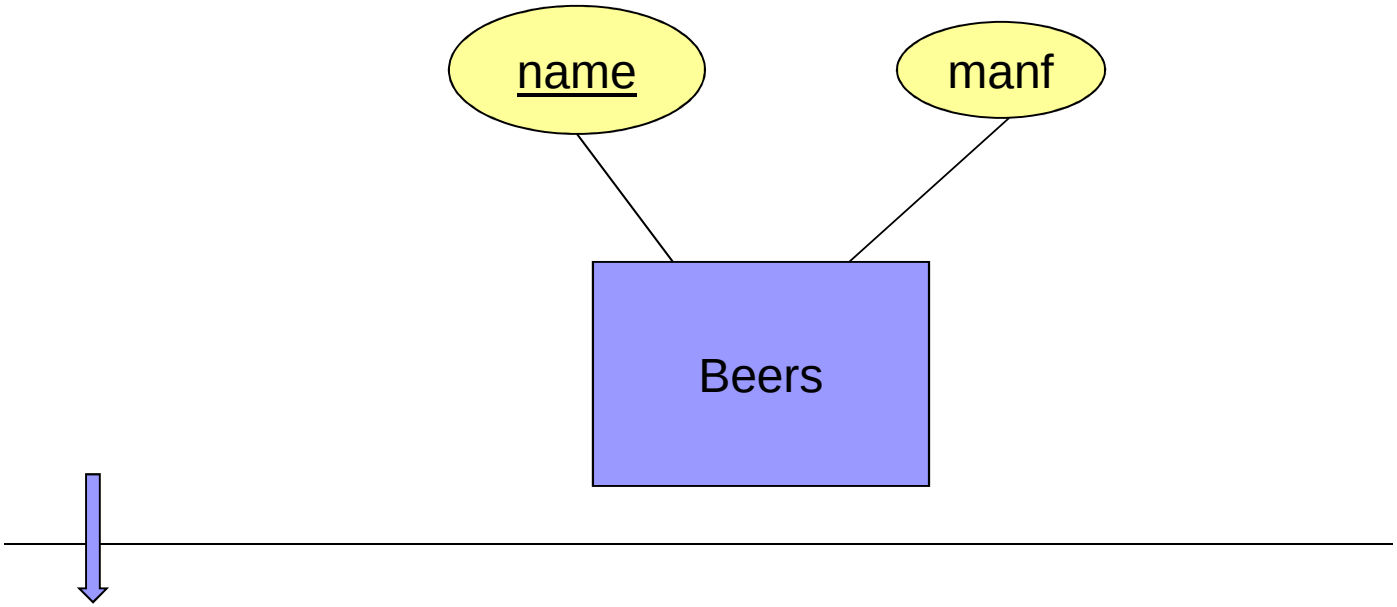
From E/R Diagrams to Relations

- Entity set → relation
  - with same: Attributes → attributes
- Relationship → relation whose attributes are only:
  - The keys of the connected entity sets
  - Attributes of the relationship itself

From E/R Diagrams to Relations (2): Special handling

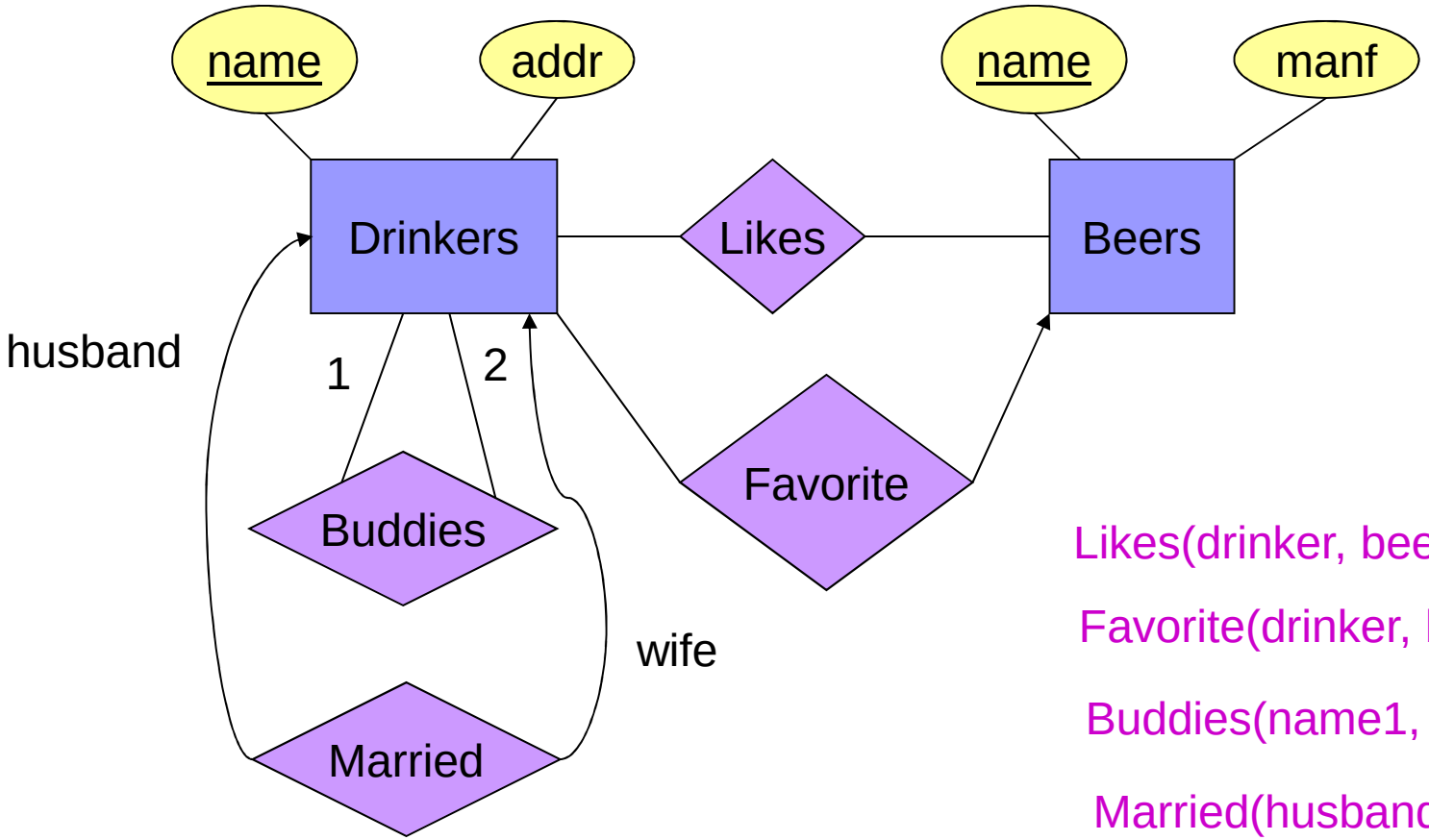
- Weak entity sets are not translated directly to relations
- "Isa" relationships and subclasses require careful treatment
- Sometime, we may combine two relations, especially entity set E and the relation that comes from a many-one relationship (where E is the many) from E to entity set F

Entity Set → Relation



Relation: Beers(name, manf)

Relationship → Relation



Likes(drinker, beer)

Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Combining Relations

- **OK to combine into one relation:**
  1. The relation for an entity-set *E*
  2. The relations for many-one relationships of which *E* is the "many"

- **Example:**

  Drinkers(name, addr)

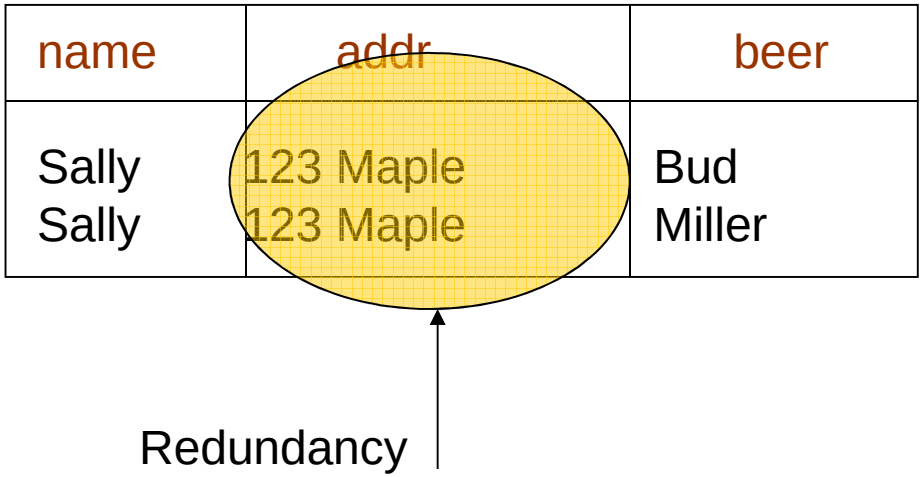
  Favorite(drinker, beer) will be combined with Drinkers

  Beer(…..)

  Drinker1(name, addr, favBeer) this is the new Drinkers

Risk with Many-Many Relationships

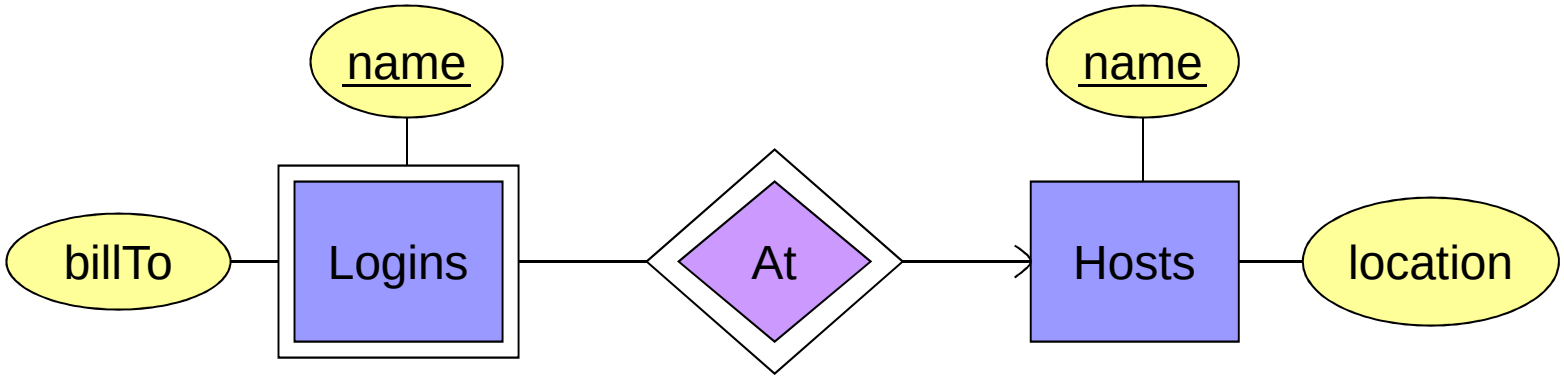
- Combining Drinkers (name, addr) with Likes (drinker, beer) would be a mistake. It leads to redundancy, as:



Handling Weak Entity Sets

- 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 supporting relationship is redundant and yields no relation (unless *it* has attributes).

Example: Weak Entity Set -> Relation



Hosts(hostName, location)

Logins(loginName, hostName, billTo)

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

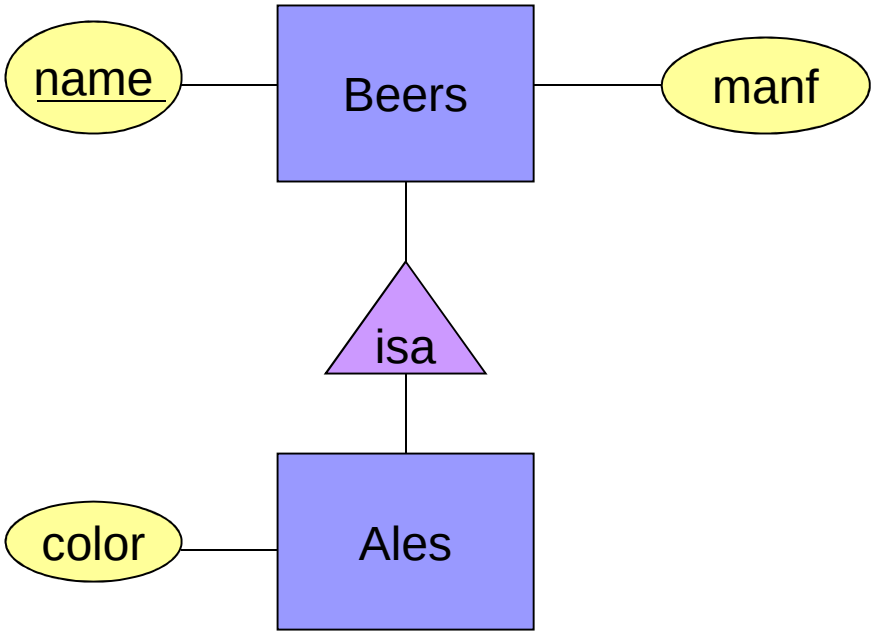
Must be the same

At becomes part of Logins

Subclasses: Three Approaches

1. *Object-oriented*: Treat entities as objects belonging to a single class. For each subtree that includes the root, create a relation, whose schema includes all attributes of all entity sets in the subtree
2. *E/R style*: One relation for each entity set E (subclass) in the hierarchy:
   - Key attribute(s) from the root
   - Attributes of that subclass
3. *Use nulls*: One relation including all attributes from all entities; entities represented as tuples with NULL in attributes that don't belong to them

Example: Subclass → Relations



1. Object-Oriented

name	manf
Bud	Anheuser-Busch

Beers

name	manf	color
Summerbrew	Pete's	dark

Ales

Good for queries like "find the color of ales made by Pete's"

2. E/R Style

name	manf
Bud	Anheuser-Busch
Summerbrew	Pete's

Beers

name	color
Summerbrew	dark

Ales

Good for queries like "find all beers (including ales) made by Pete's"

3. Using Nulls

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

Beers

Saves space unless there are *lots* of attributes that are usually NULL.

Other High-Level Design Languages

Object Description Language &
Unified Modeling Language

Object-Oriented DBMS's

- **Standards group:** ODMG -- Object Data Management Group
- **ODL** -- Object Description Language, like CREATE TABLE part of SQL
- **OQL** -- Object Query Language, tries to imitate SQL in an OO framework

Unified Modeling Language - UML

- **UML** is designed to model software, but has been adapted as a database modeling language

- **Midway between E/R and ODL:**
  - No multiway relationships as in E/R
  - But (E/R) allows attributes on binary relationships, which ODL doesn't
  - Has a graphical notation, unlike ODL

END