Relational Model

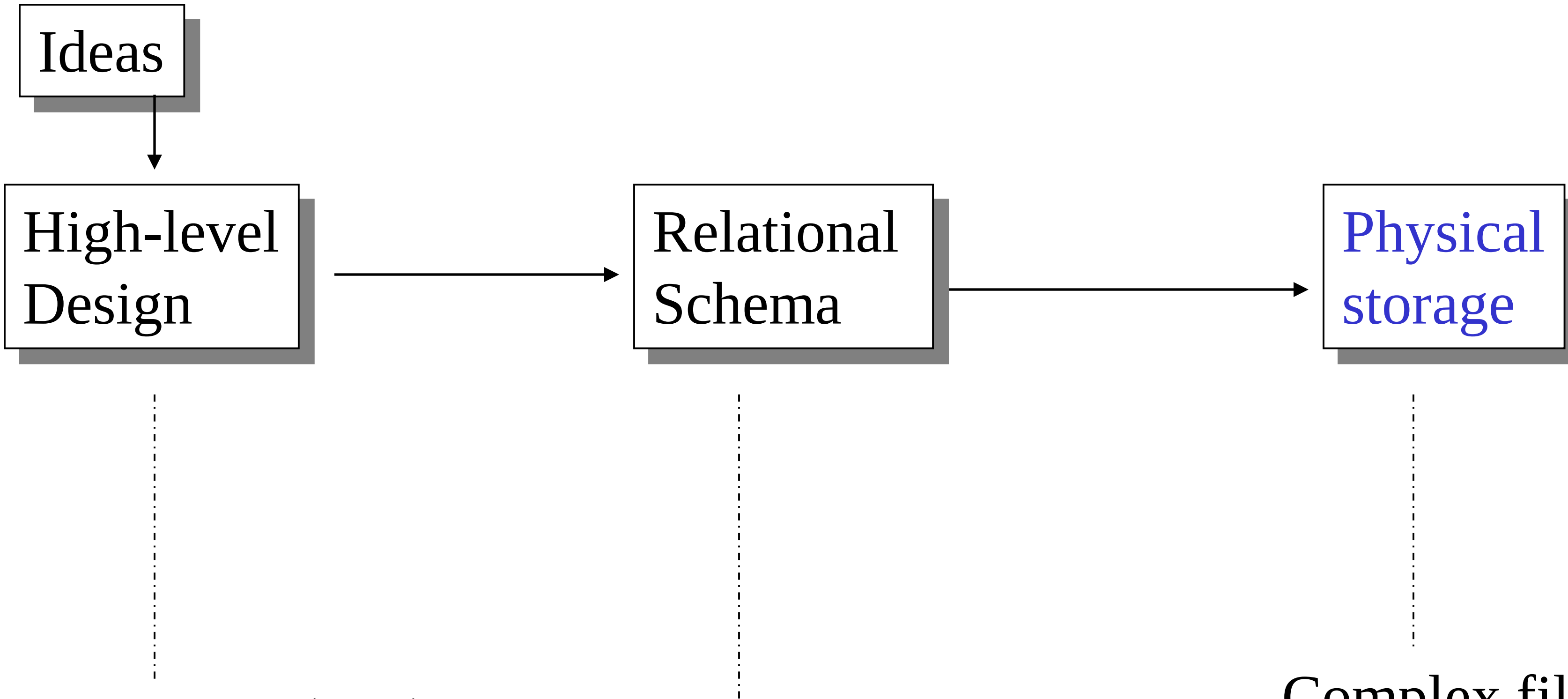
DSCI 551
Wensheng Wu

Lecture Outline

- Relational model
- Translating ER into relational model

Motivations & comparison of ER with relational model ...

Database Modeling & Implementation



Diagrams (E/R)

Tables:
columns: attributes
rows: tuples

Complex file organization and index structures.

ER Model vs. Relational Model

- Both are used to model data
- ER model has many concepts
  - entities, relationships, is-a, etc.
  - well-suited for capturing the app. requirements
  - not well-suited for computer implementation
- Relational model
  - has just a single concept: relation
  - world is represented with a collection of tables
  - well-suited for efficient manipulations on computers

The basics of the relational model ...

An Example of a Relation

Table/relation name

Products:

Attribute names

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

tuples

Domains

- Each attribute has a type
- Must be atomic type
- Called *domain*
- Examples:
  - Integer
  - String
  - Real
  - ...

Schemas vs. instances
(very important, make sure you know the difference)

Schemas

Schema: describe the structure of data

The Schema of a Relation:

- Relation name plus attribute names
- E.g. Product(Name, Price, Category, Manufacturer)
- In practice we add the domain for each attribute

The Schema of a Database

- A set of relational schemas
- E.g. Product(Name, Price, Category, Manufacturer),
  Vendor(Name, Address, Phone),
  . . . .

Instances

Schema instance = data

- **Relational schema** = R(A1, ..., Ak):
  Instance = relation (of "type" R) with a collection of tuples
  – Each has k values from the domains of their corresponding attributes

- **Database schema** = R1(...), R2(...), ..., Rn(...)
  Instance = n relations, of types R1, R2, ..., Rn

Example

Relational schema: Product(Name, Price, Category, Manufacturer)

Instance:

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Updates

The database maintains a current database state.

Updates to the data:

1) add a tuple
2) delete a tuple
3) modify the values of some attributes in a tuple

Updates to the data happen very frequently.

Updates to the schema: relatively rare. Rather painful. Why**?**

Schemas and Instances

- Analogy with programming languages:
  - Schema = type/class
  - Instance = value/instance
- Important distinction:
  - Database Schema = stable over long periods of time
  - Database Instance = changes constantly, as data is inserted/updated/deleted

How should we talk about relations (that is, represent them)?

Two Mathematical Definitions of Relations

Product(Name, Price, Category, Manufacturer)

Relation as a subset of Cartesian product

- Tuple = element of **string** x **int** x **string** x **string**
- E.g. t = ("gizmo", 19, "gadgets", "GizmoWorks")
- Relation = subset of **string** x **int** x **string** x **string**
- Order in the tuple is important!
  - ("gizmo", 19, "gadgets", "GizmoWorks")
  - ("gizmo", 19, "GizmoWorks", "gadgets")
- No (explicit) attributes (in tuple expression)

Relation as a set of functions

- Fix the set of attributes
  - A={name, price, category, manufacturer}
- A tuple = function t: A → attribute domains
- Relation = a set of tuples/functions

- E.g. t(name) = "gizmo", t(price) = 19, t(category) = "gadgets", t(manufacturer) = "GizmoWorks"

- Order in a tuple is not important
- Attribute names are important

Examples of Insert

- Positional tuples, without specifying attribute names
  - E.g., insert into Employee values (123, 'john', 35, 'los angeles)

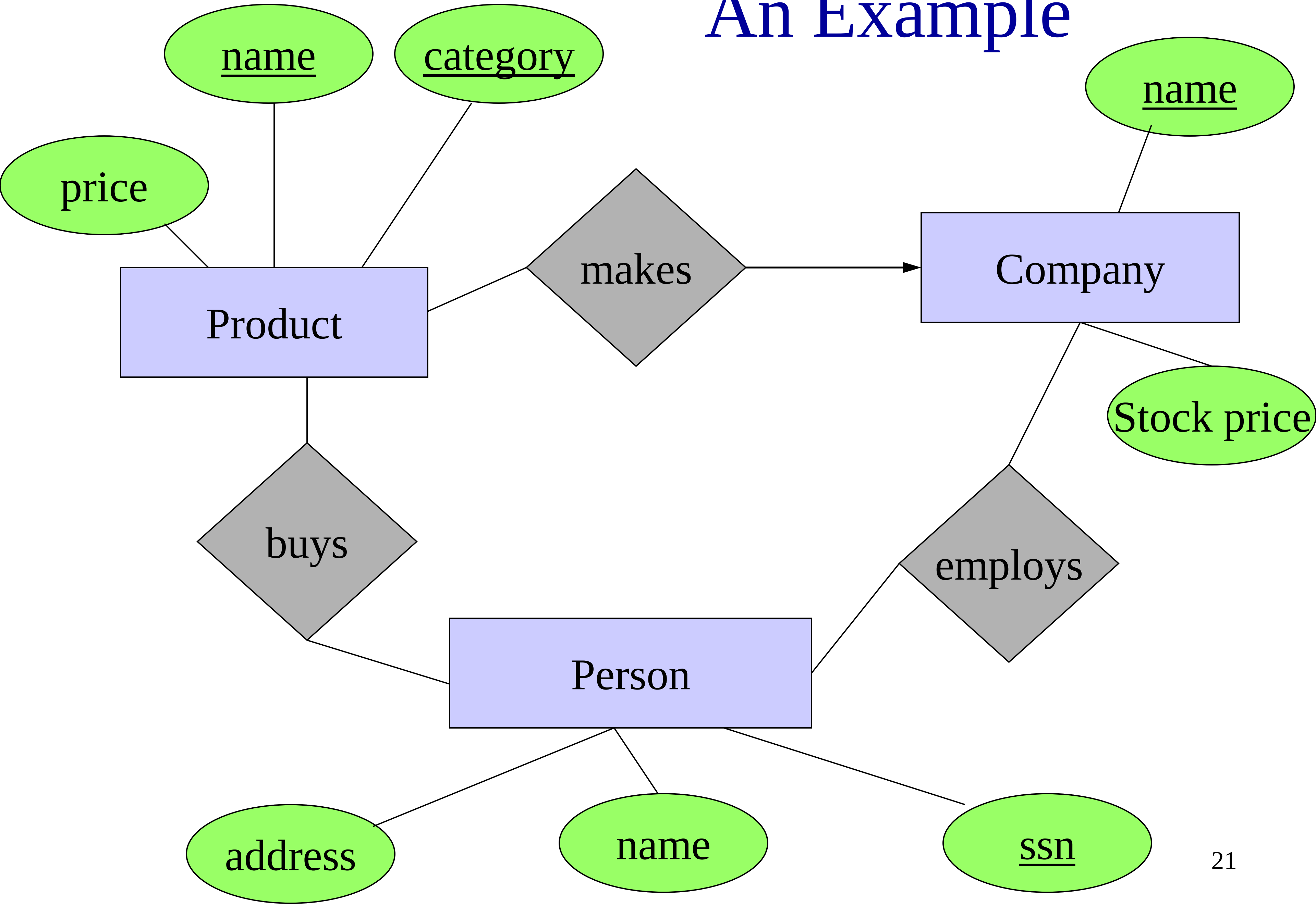
- Relational schemas with attribute names
  - E.g., insert into Employee(id, name) values (123, 'john')

Now the fun part: translating from ER to relational model

Translating ER Diagram to Rel. Model

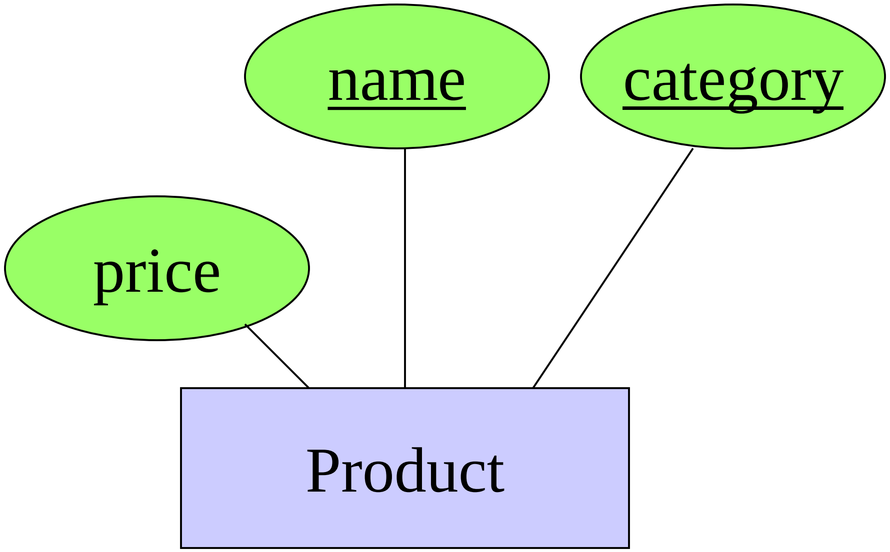
- Basic cases
  - entity set E => relation with attributes of E
  - relationship R => relation with attributes being keys of related entity sets + attributes of R
- Special cases
  - combining two relations
  - translating weak entity sets
  - translating is-a relationships and subclasses

An Example


name
category
price
Product
makes
Company
name
Stock price
buys
employs
Person
address
name
ssn

Basic cases ...

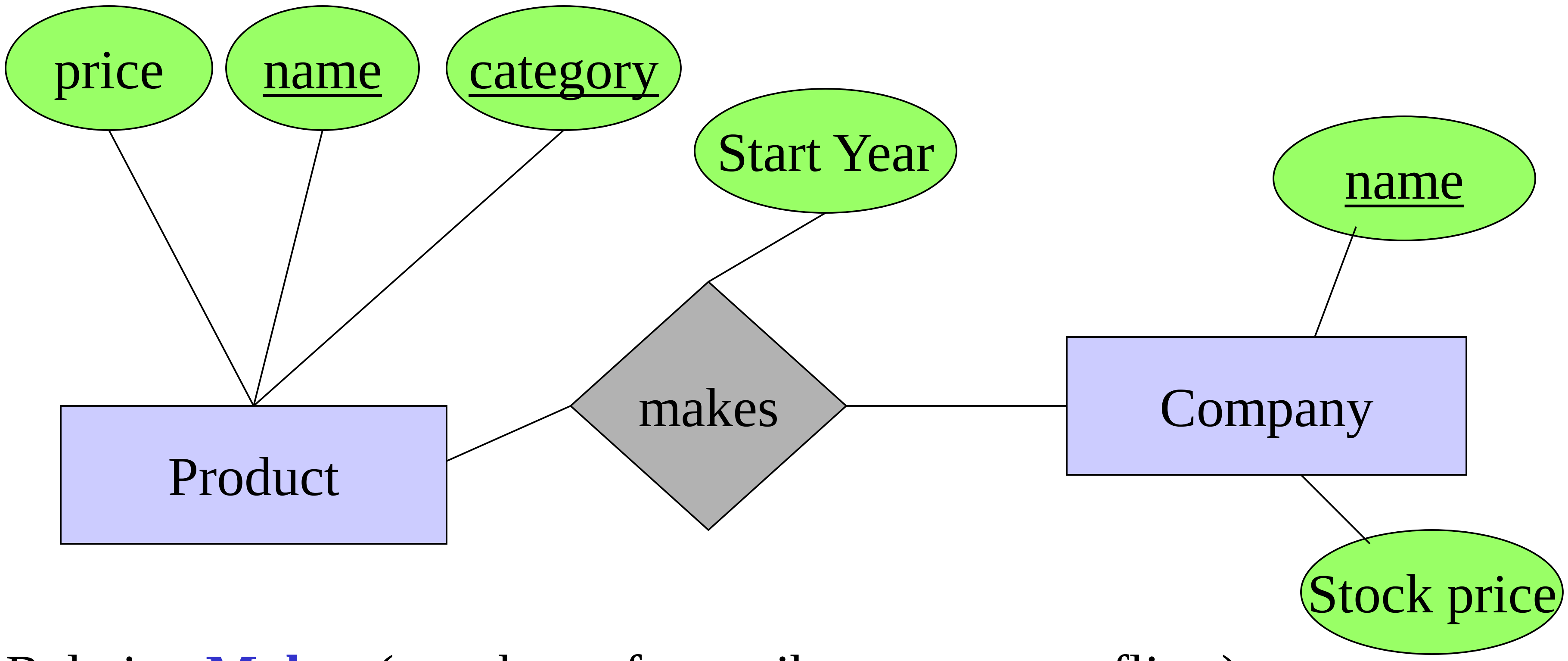
Entity Sets to Relations



Product:

Name	Category	Price
gizmo	gadgets	$19.99

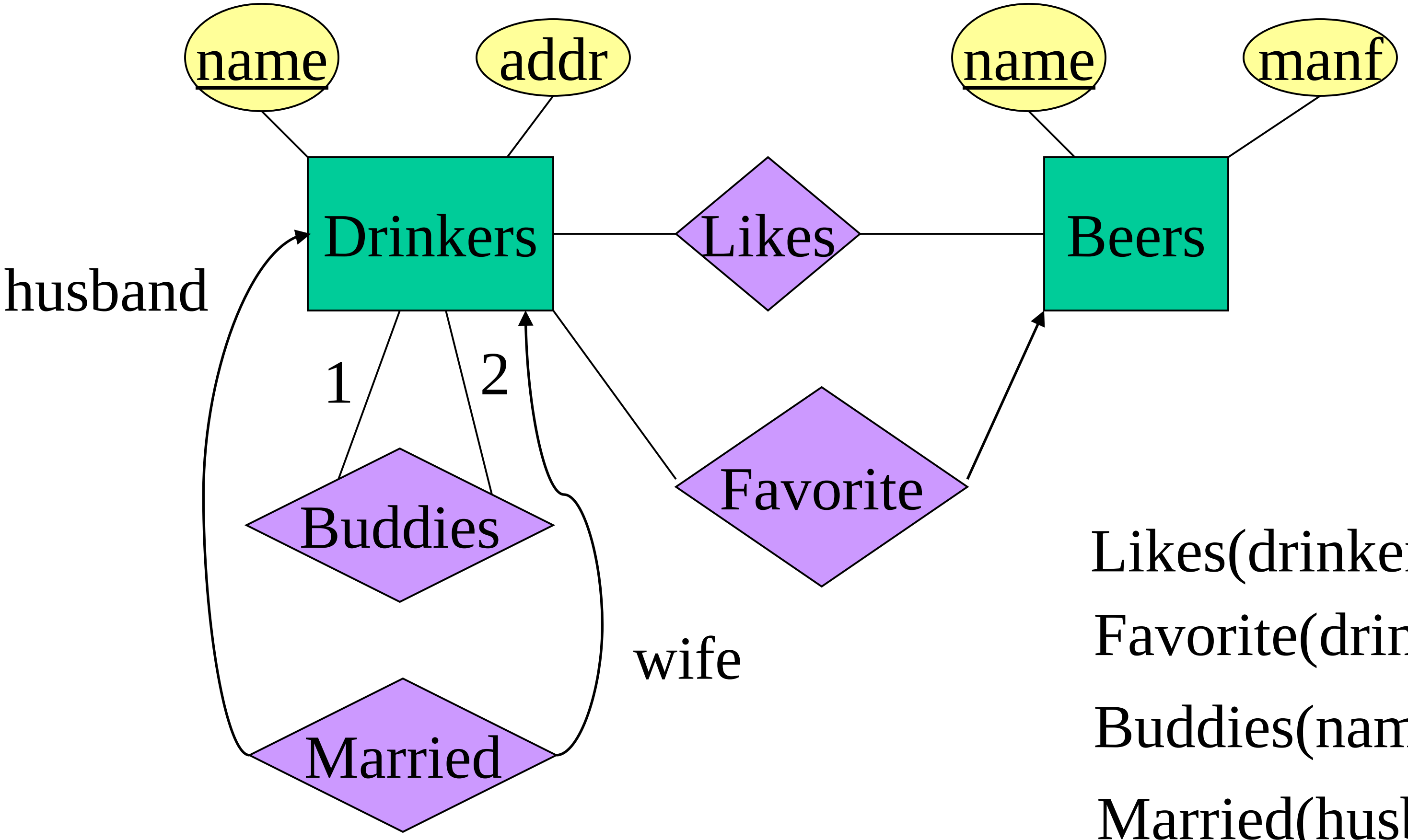
Relationships to Relations



Relation **Makes** (watch out for attribute name conflicts)

Product-name	Product-Category	Company-name	Starting-year
gizmo	gadgets	gizmoWorks	1963

Relationship to Relation: Another Example



Likes(drinker, beer)

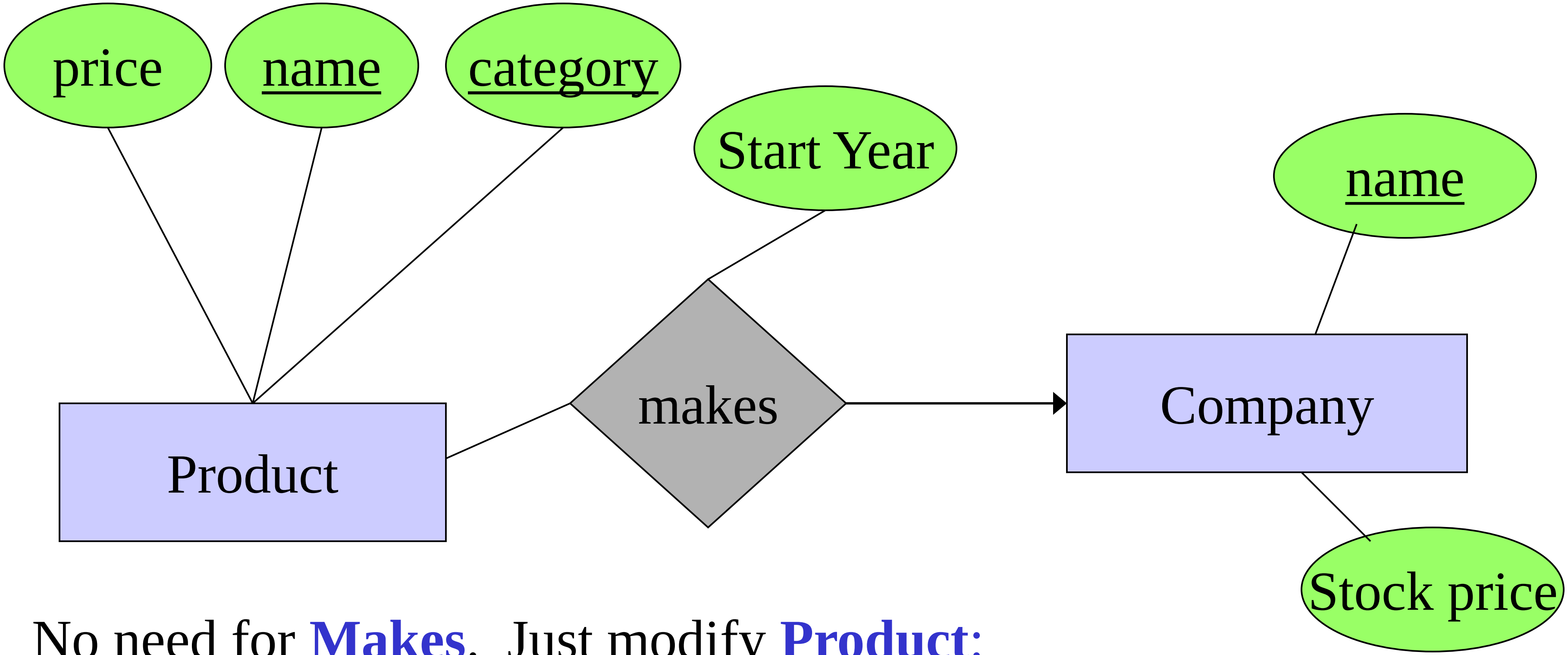
Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

Special cases:
1) many-one relations
2) weak entity sets
3) is-a cases

Combining Two Relations

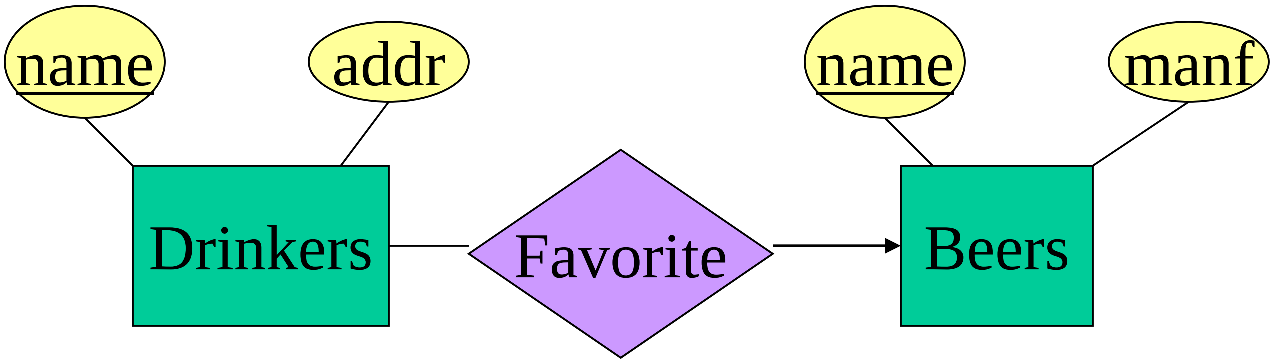


No need for **Makes**. Just modify **Product**:

name	category	price	StartYear	companyName
gizmo	gadgets	19.99	1963	gizmoWorks

Combining Relations

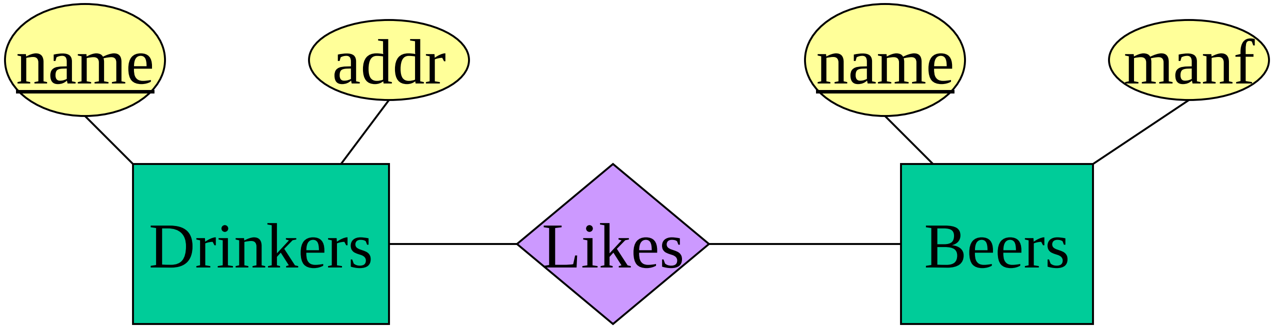
- Combine relation for an m-1 relationship R with the relation for the entity set on the many side of R



- Example: combine Drinkers(name, addr) and Favorite(drinker, beer) => Drinkers(name, addr, favoriteBeer).
  - But any drawback from doing this?

Risk with Many-Many Relationships

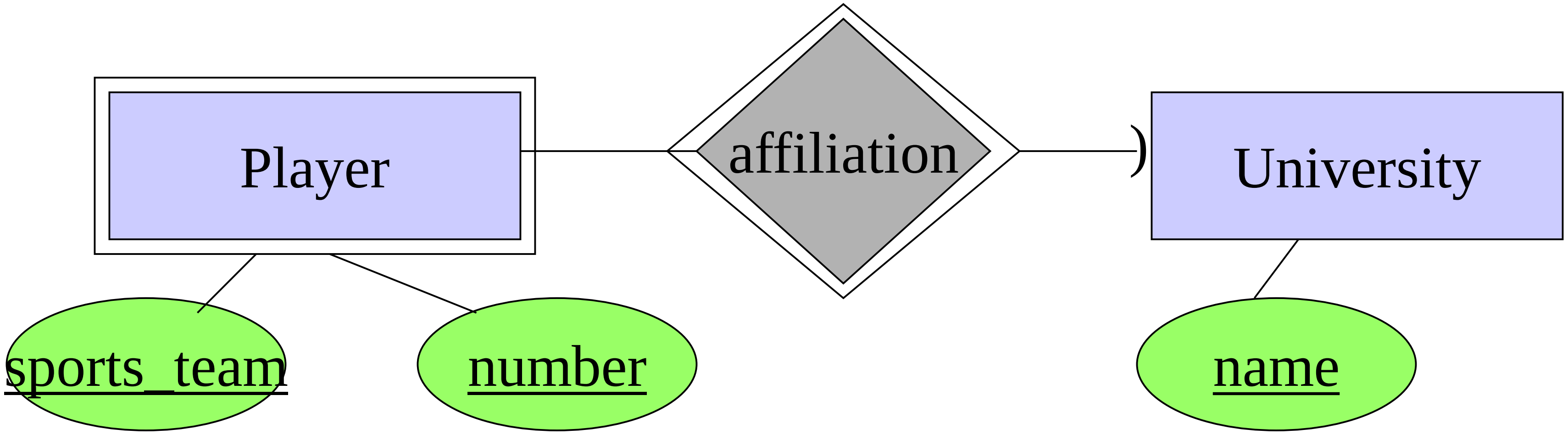
- Combining Drinkers with Likes would be a mistake. It leads to redundancy, as:



name	addr	beer
Sally	123 Maple	Bud
Sally	123 Maple	Miller

Redundancy

Handling Weak Entity Sets



Relation Player:

SportTeam	Number	Affiliated University
Trojan	15	USC

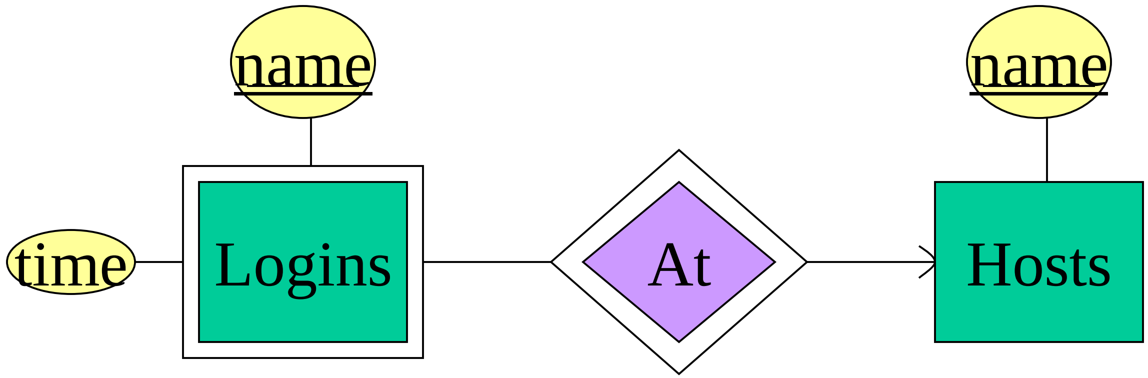
- need all the attributes that contribute to the key of Player
- don't need a separate relation for Affiliation. (why ?)

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 (double-diamond) relationship is redundant and yields no relation.

Another Example



Hosts(hostName)

Logins(loginName, hostName, time)

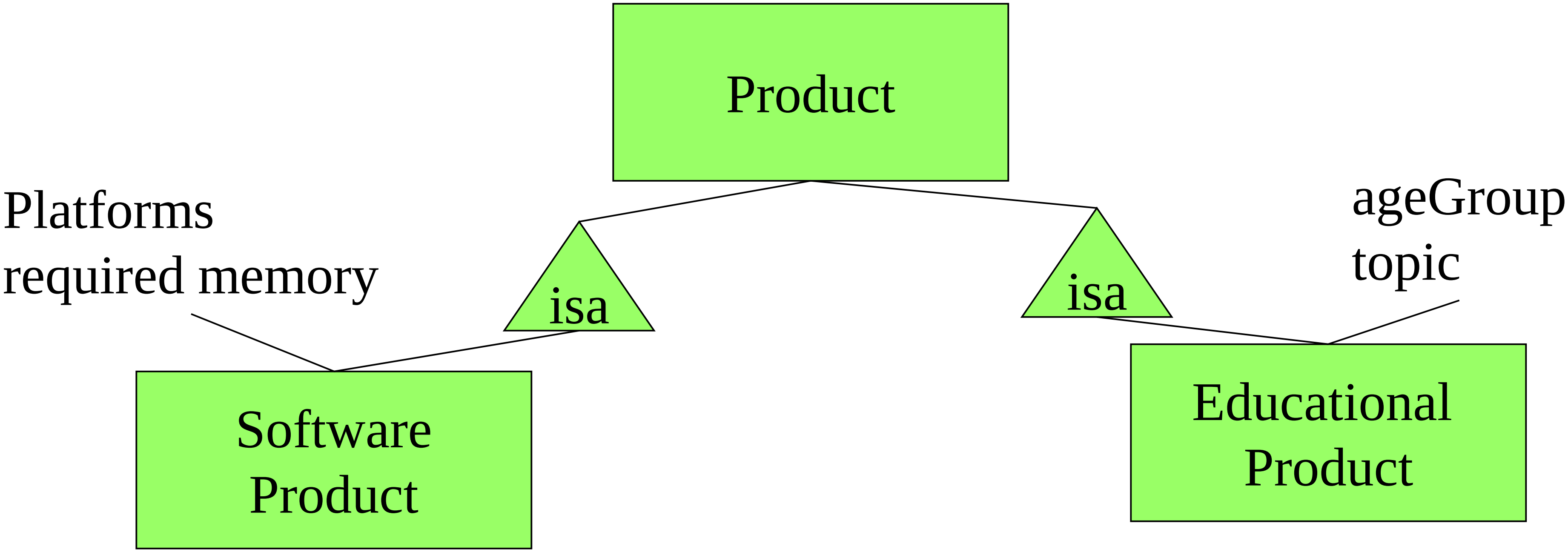
~~At(loginName, hostName, hostName2)~~

Must be the same

At becomes part of Logins

Translating Subclass Entities

Product(<u>name</u>, price, category, manufacturer)



Option #1: the OO Approach

4 tables: each object can only belong to a single table

One table for each subtree rooted at Product

Product(name, price, category, manufacturer)

EducationalProduct(name, price, category, manufacturer,
ageGroup, topic)

SoftwareProduct(name, price, category, manufacturer,
platforms, requiredMemory)

EducationalSoftwareProduct(name, price, category, manufacturer,
ageGroup, topic,
platforms, requiredMemory)

(Values of) all names in different tables are distinct

Option #2: the E/R Approach

Product(<u>name</u>, price, category, manufacturer)

EducationalProduct(<u>name</u>, ageGroup, topic)

SoftwareProduct(<u>name</u>, platforms, requiredMemory)

No need for a relation EducationalSoftwareProduct

The same name value (i.e., product) may appear in several relations

Option #3: The Null Value Approach

Has one table:

Product (name, price, category,
manufacturer, age-group, topic, platforms,
required-memory)

Some values in the table will be NULL, meaning that the attribute does not make sense for the specific product.

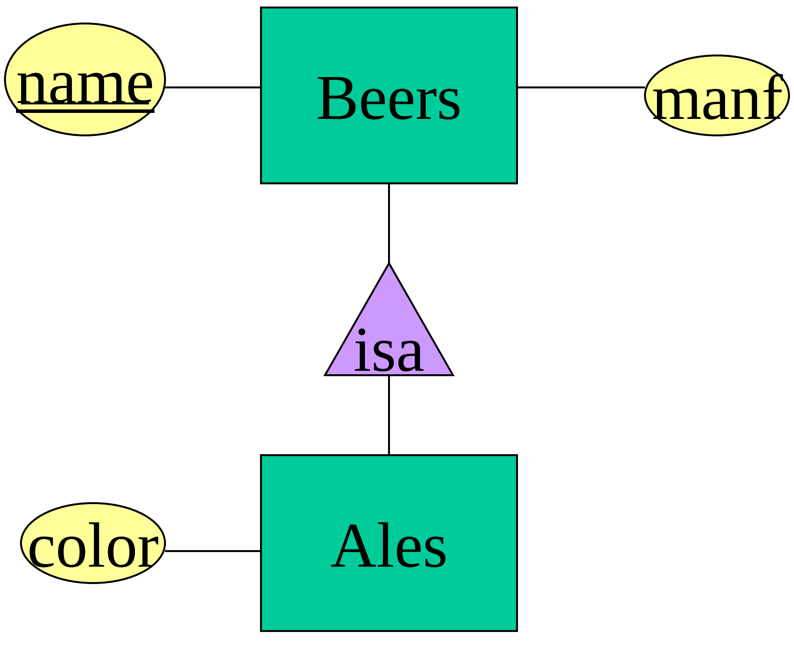
Problem: too many NULLs

Translating Subclass Entities: The Rules

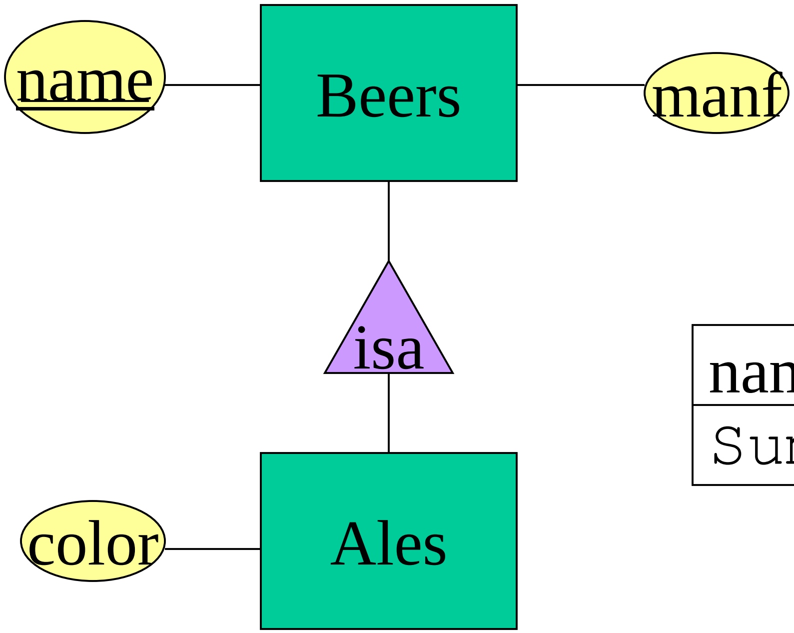
Three approaches:

1. *Object-oriented* : each entity belongs to exactly one class; create a relation for each possible subtree including the root, 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 entity set.
3. *Use nulls* : create one relation; entities have null in attributes that don't belong to them.

Example



Object-Oriented



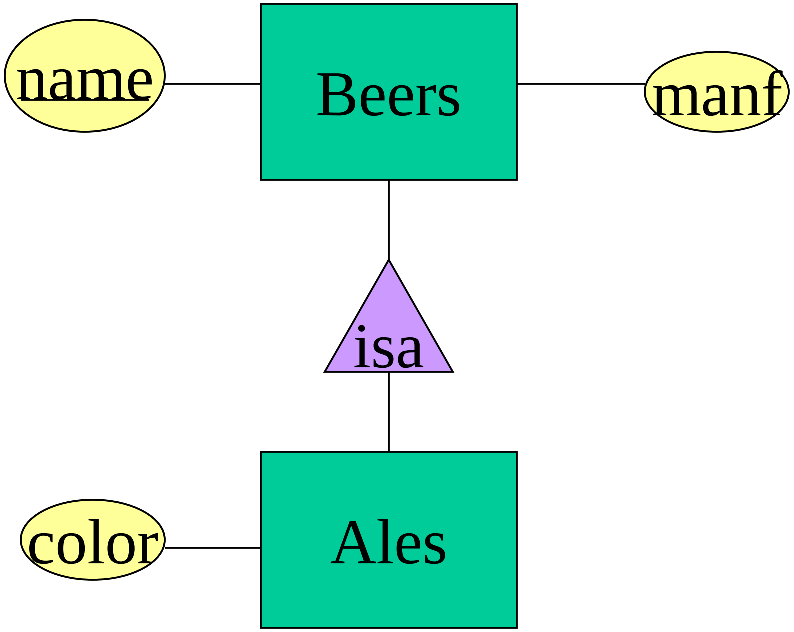
name	manf
Bud	Anheuser-Busch

Beers

name	manf	color
Summerbrew	Pete's	dark

Ales

E/R Style



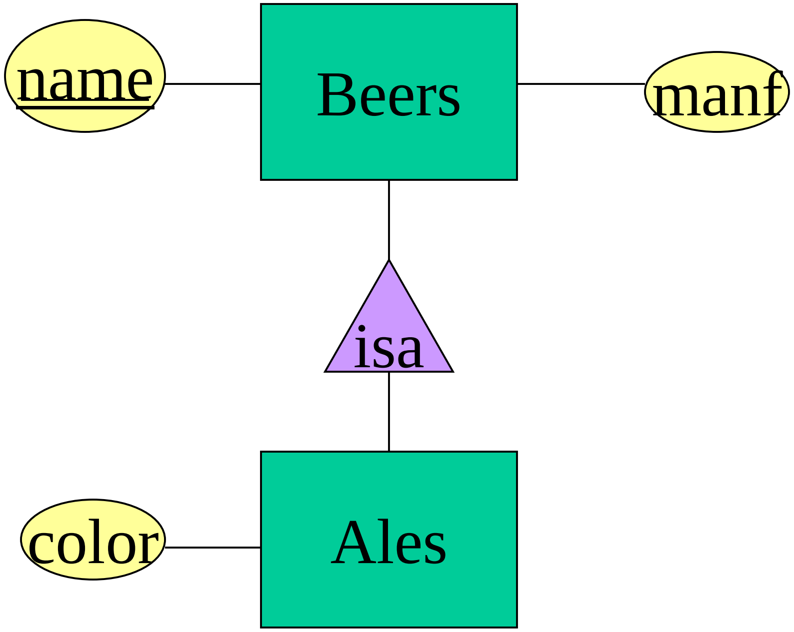
name	manf
Bud	Anheuser-Busch
Summerbrew	Pete's

Beers

name	color
Summerbrew	dark

Ales

Using Nulls



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

Beers

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 might waste space if there are *lots* of attributes that are usually null.

Mixed-Type Inheritance in PostgreSQL

- CREATE TABLE cities

  (<u>name</u> text, population real, altitude int);
- CREATE TABLE capitals

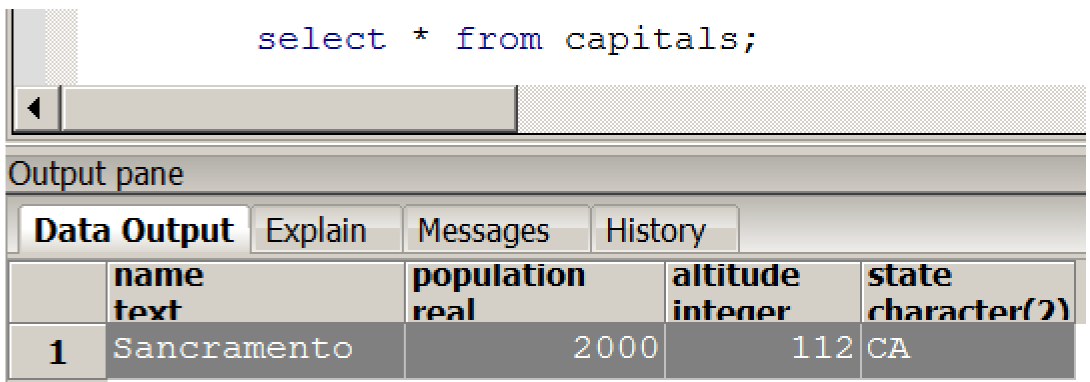
  (state char(2)) INHERITS (cities);

```
-- DROP TABLE public.capitals;

CREATE TABLE public.capitals
(
-- Inherited from table cities:  name text,
-- Inherited from table cities:  population real,
-- Inherited from table cities:  altitude integer,
  state character(2)
)
INHERITS (public.cities)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.capitals
  OWNER TO postgres;
```

Example

- insert into capitals(name, population, altitude, state) values('Sancramento', 2000, 112, 'CA');

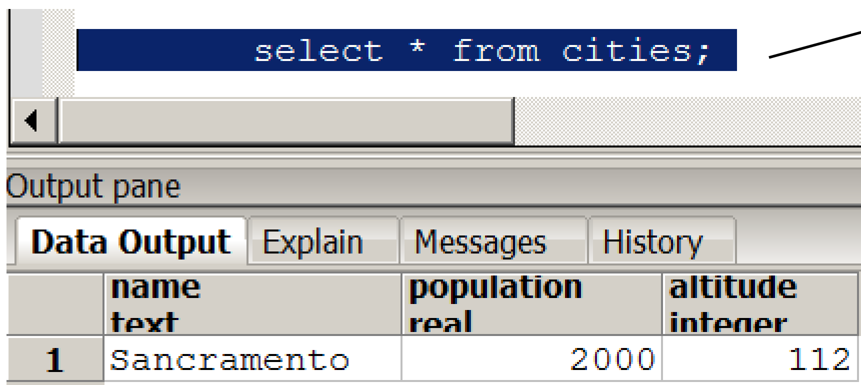


This is more like OO-approach

Caveat

- PostgreSQL *logically" adds a tuple into cities
  - Cities may be regarded as view
  - View: (select * from "non-capital cities") union

    (select name, population, altitude from capitals)



This is more of ER approach

- delete * from capitals
  - Will remove "logical" tuple from cities as well

Translation Review

- Basic cases
  - entity to table, relationship to table
  - selecting attributes based on keys
- Special cases
  - many-one relation can be merged
  - merging many-many is dangerous
  - translating weak entity sets
  - translating isa hierarchy
    - 3 choices, with trade-offs