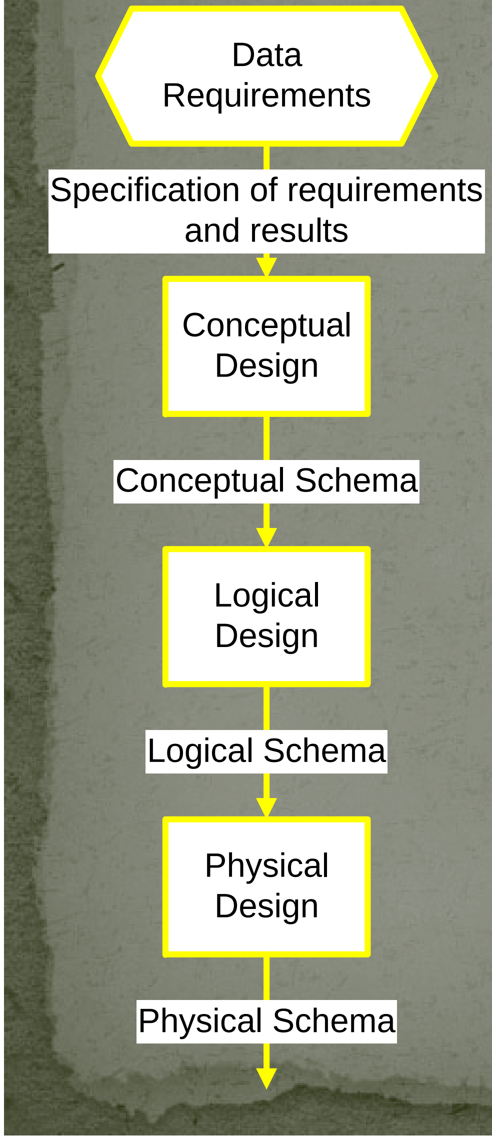
# CS 505: Intermediate Topics to Database Systems

Instructor: Jinze Liu

Fall 2008

# Phases of Database Design



- Conceptual design begins with the collection of requirements and results needed from the database (ER Diag.)
- Logical schema is a description of the structure of the database (Relational, Network, etc.)
- Physical schema is a description of the implementation (programs, tables, dictionaries, catalogs

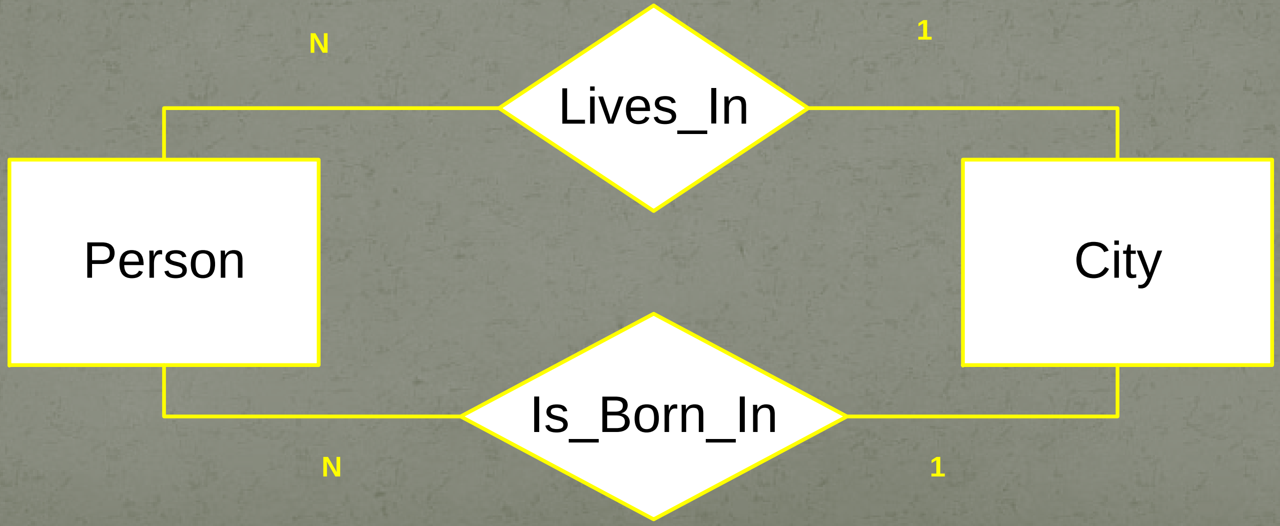
# Models

A ***data model*** is a collection of objects that can be used to represent a set of *data* and *operations* to manipulate the data

- *Conceptual models* are tools for representing reality at a very high-level of abstraction
- *Logical models* are data descriptions that can be processed by computers

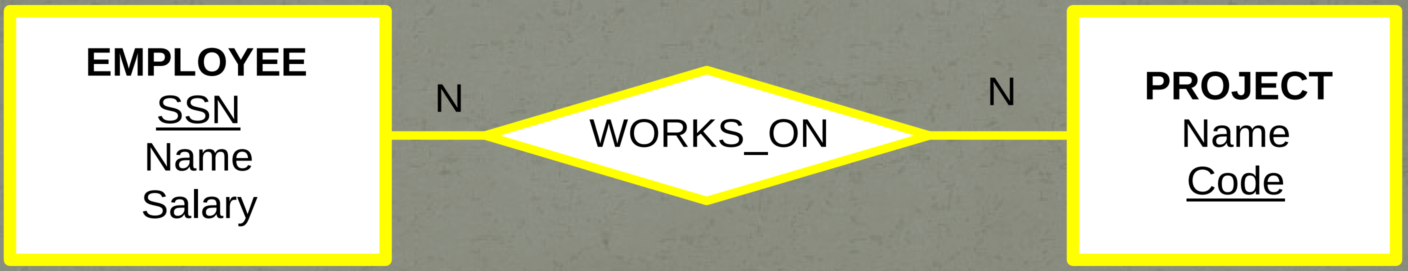
# Conceptual model:
# Entity-Relationship Diagrams

- ***Entities*** represent classes of *real-world* objects. **Person, Students, Projects, Courses** are entities of a University database
- ***Relationships*** represent interactions between two or more entities



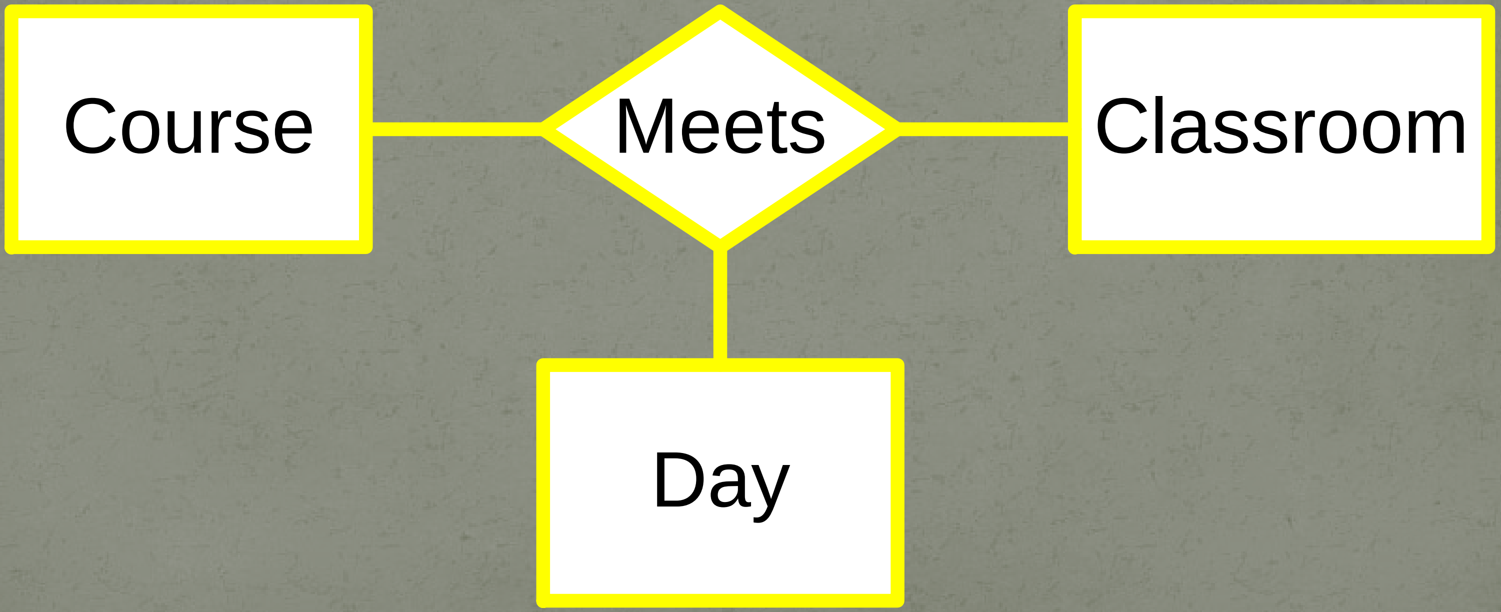
# Example:

- Every employee works in at least one project
- Every project has employees working on it.



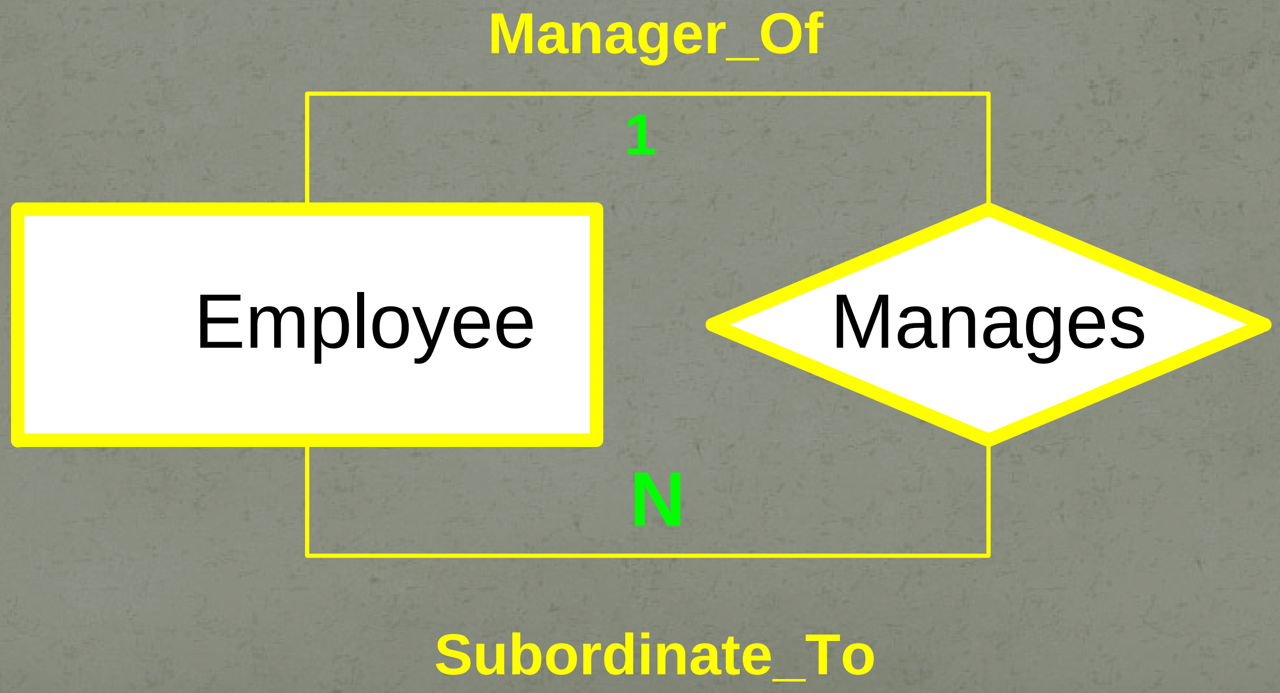
# Higher-Order Relationships

A relationship may involve more than *two* entities



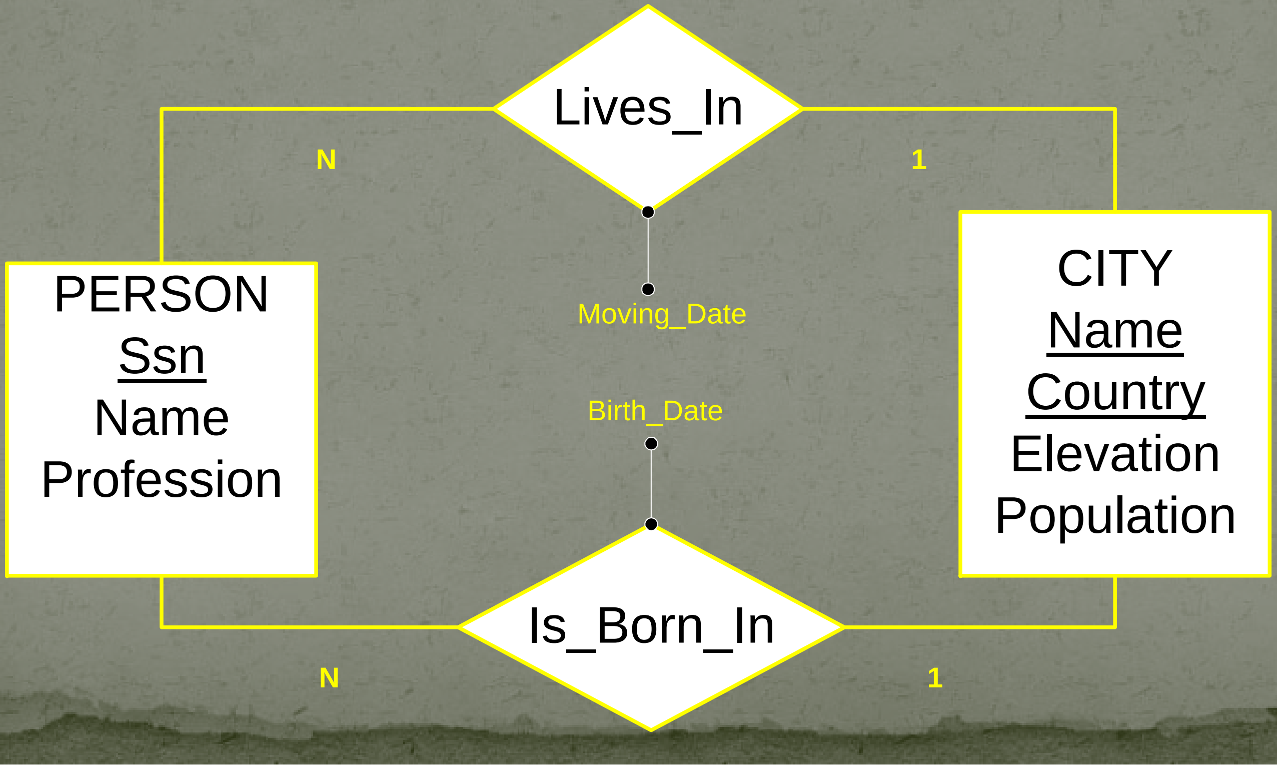
# Recursive relationships

Relationships could be mapped from one entity to itself



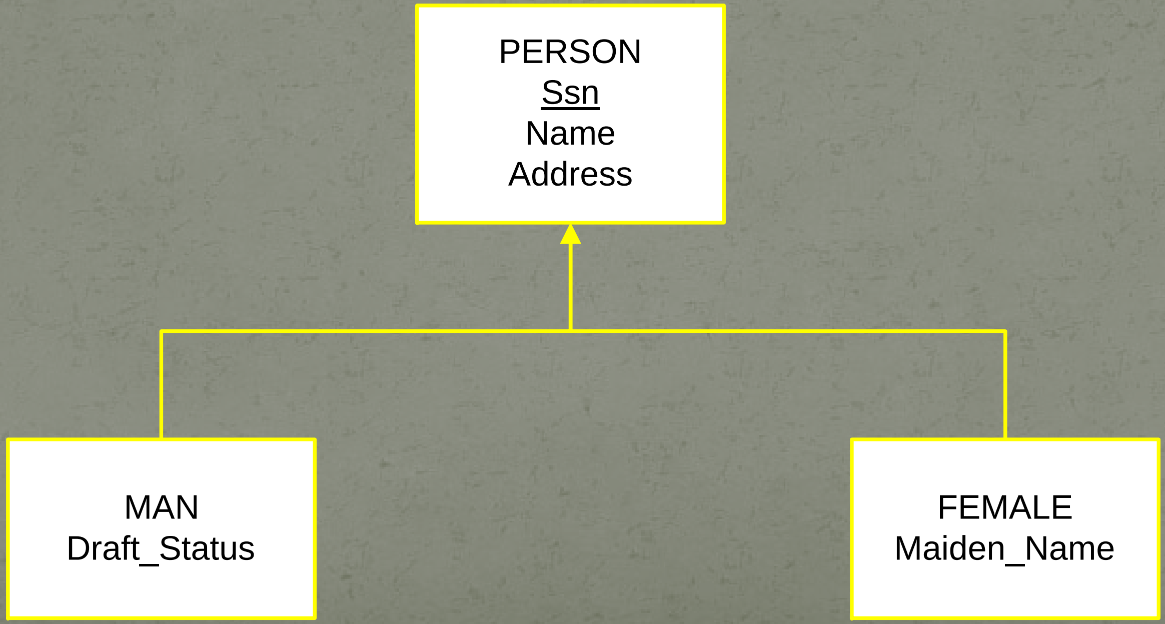
# Attributes

*Attributes* represent elementary properties of the entities or relationships. The stored data will be kept as values of the attributes



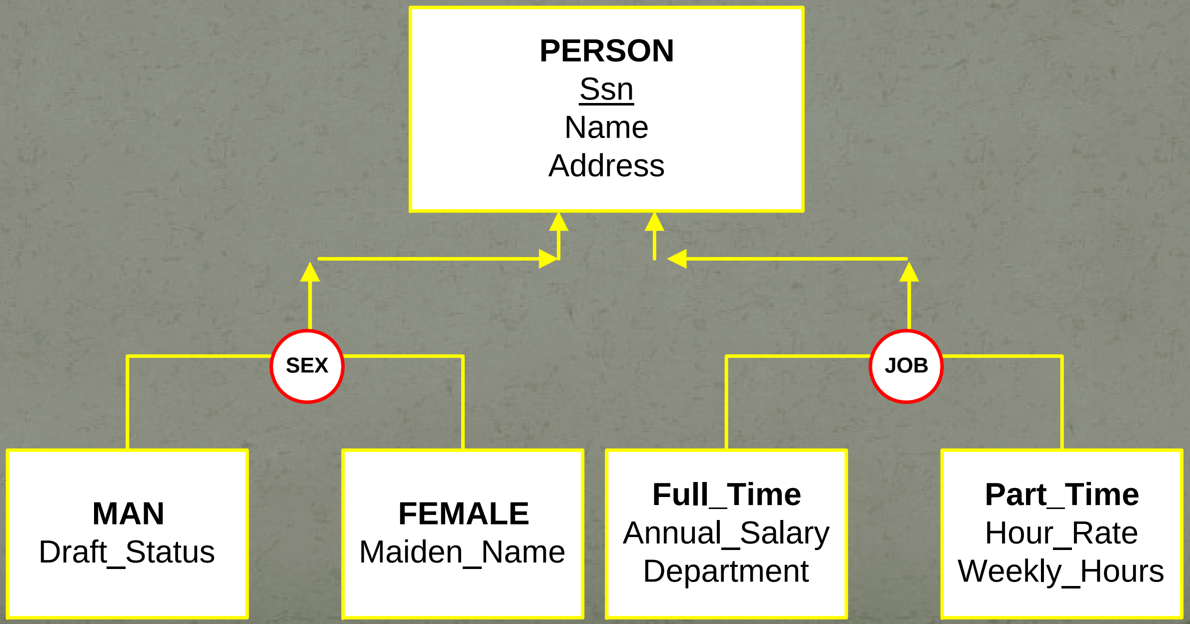
# Generalizations

- An entity could be *seen* from many different viewpoints
- Each viewpoint defines a set of *roles* in a generalization
- Example below uses *SEX* to classify the object "Person"

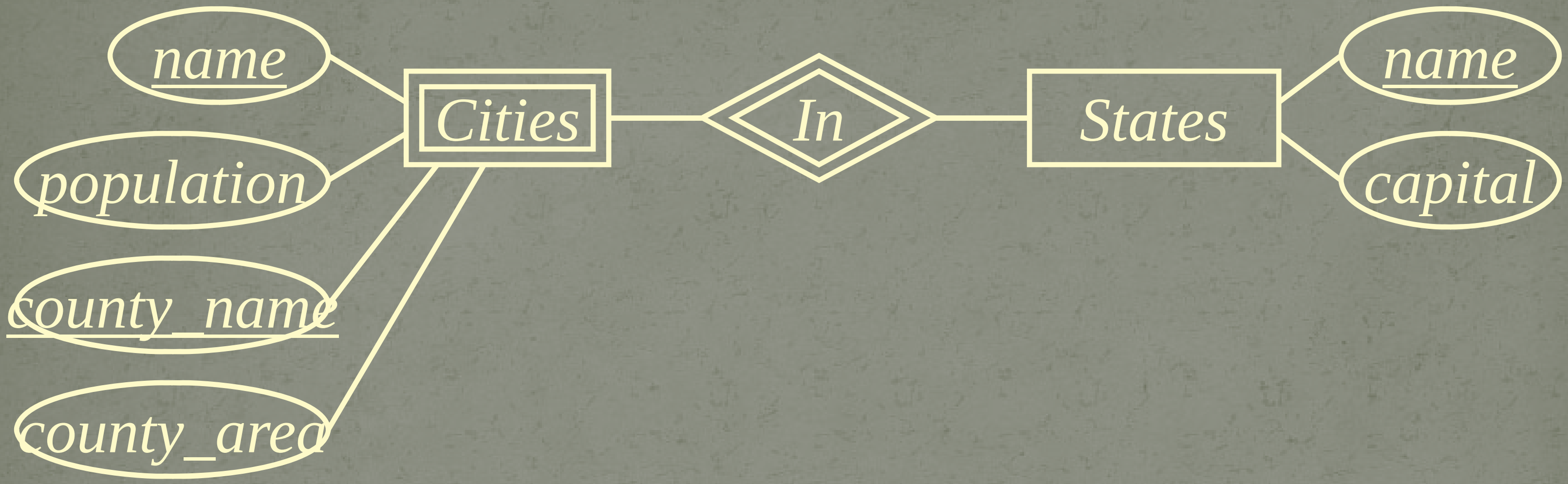


# Generalizations

- A classification could be *disjoint* or *overlapping*
- An entity could have more than one classification

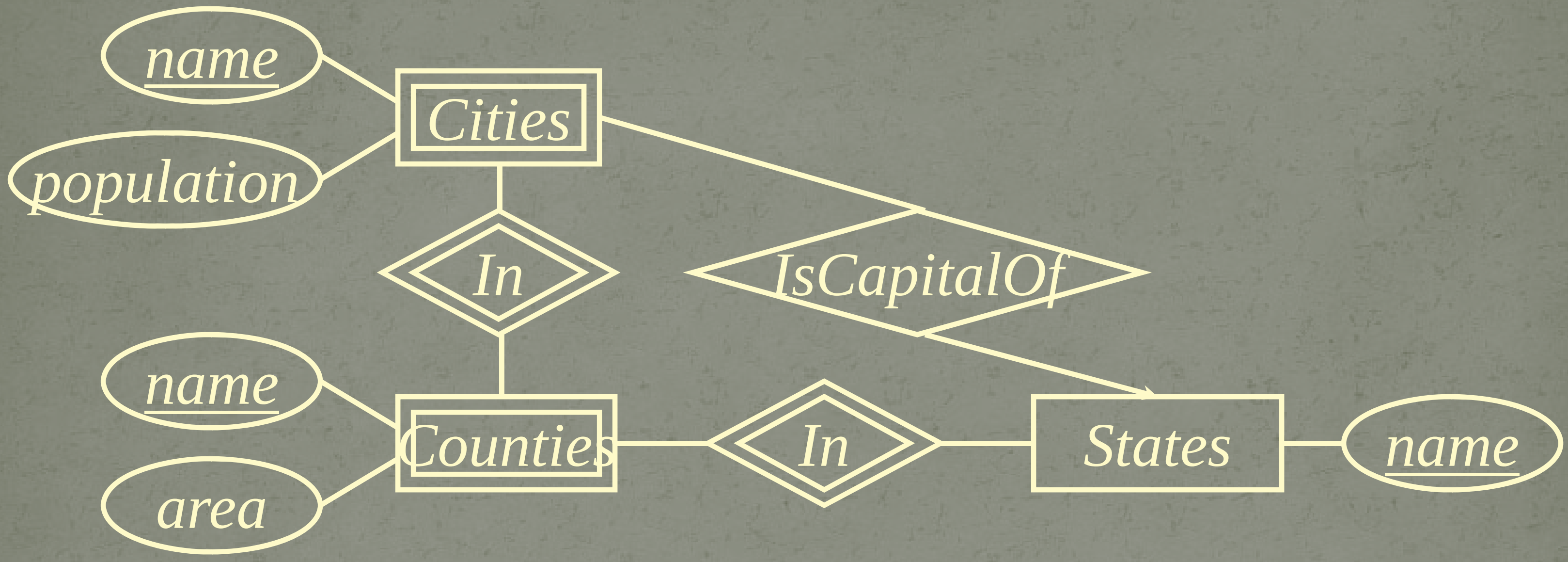


# Case study : first design



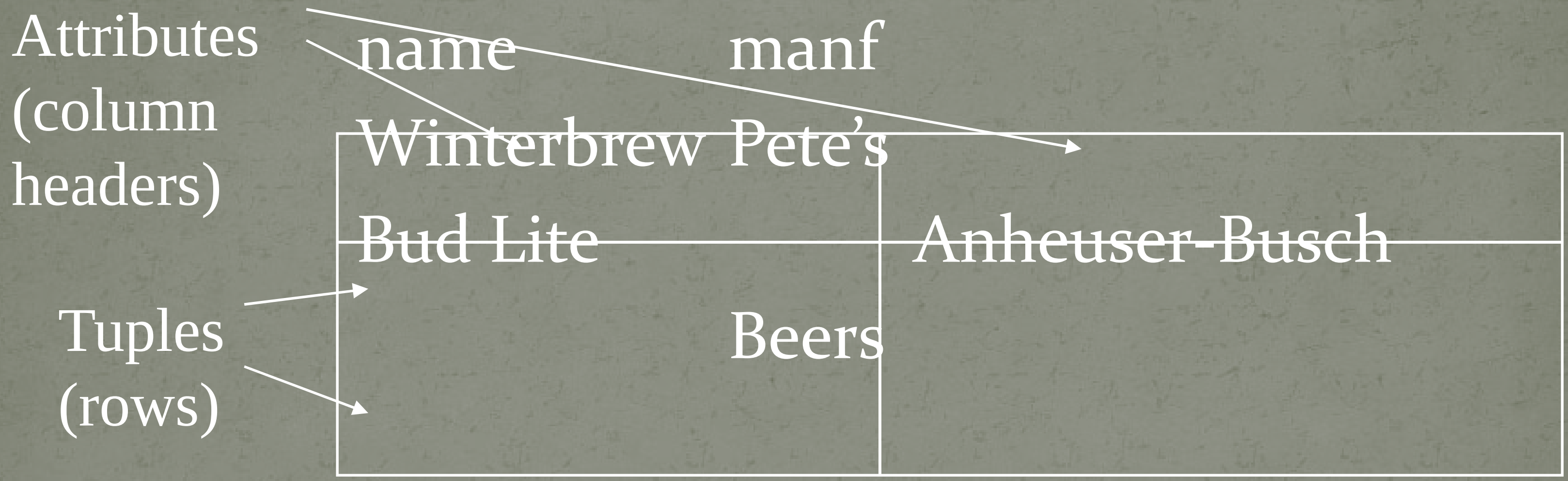
- County area information is repeated for every city in the county
  - ☞ Redundancy is bad.
  - ☞ What else?
- State capital should really be a city
  - ☞ Should “reference” entities through explicit relationships

# Case study : second design



- Technically, nothing in this design could prevent a city in state *X* from being the capital of another state *Y*, but oh well…

# A Relation is a Table



| name | manf |
|---|---|
| Winterbrew | Pete's |
| Bud Lite | Anheuser-Busch |

Beers

# Schemas

- *Relation schema* = relation name + attributes, in order (+ types of attributes).
  - Example: Beers(name, manf) or Beers(name: string, manf: string)
- *Database* = collection of relations.
- *Database schema* = set of all relation schemas in the database.

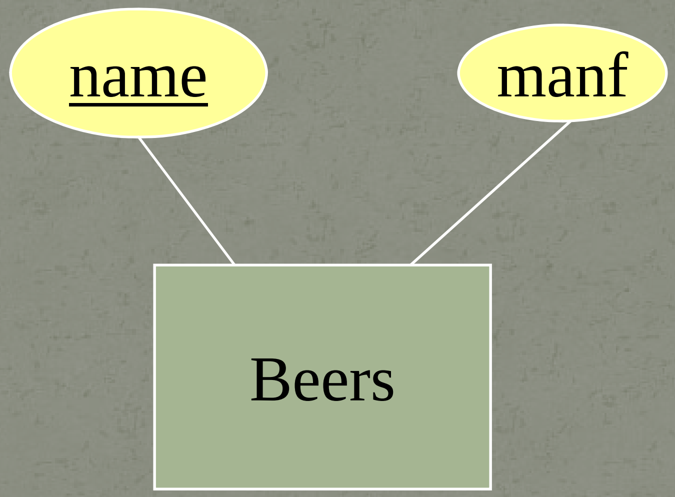
# Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.
  - But SQL uses bags, while the relational model is a set-based model.

# From E/R Diagrams to Relations

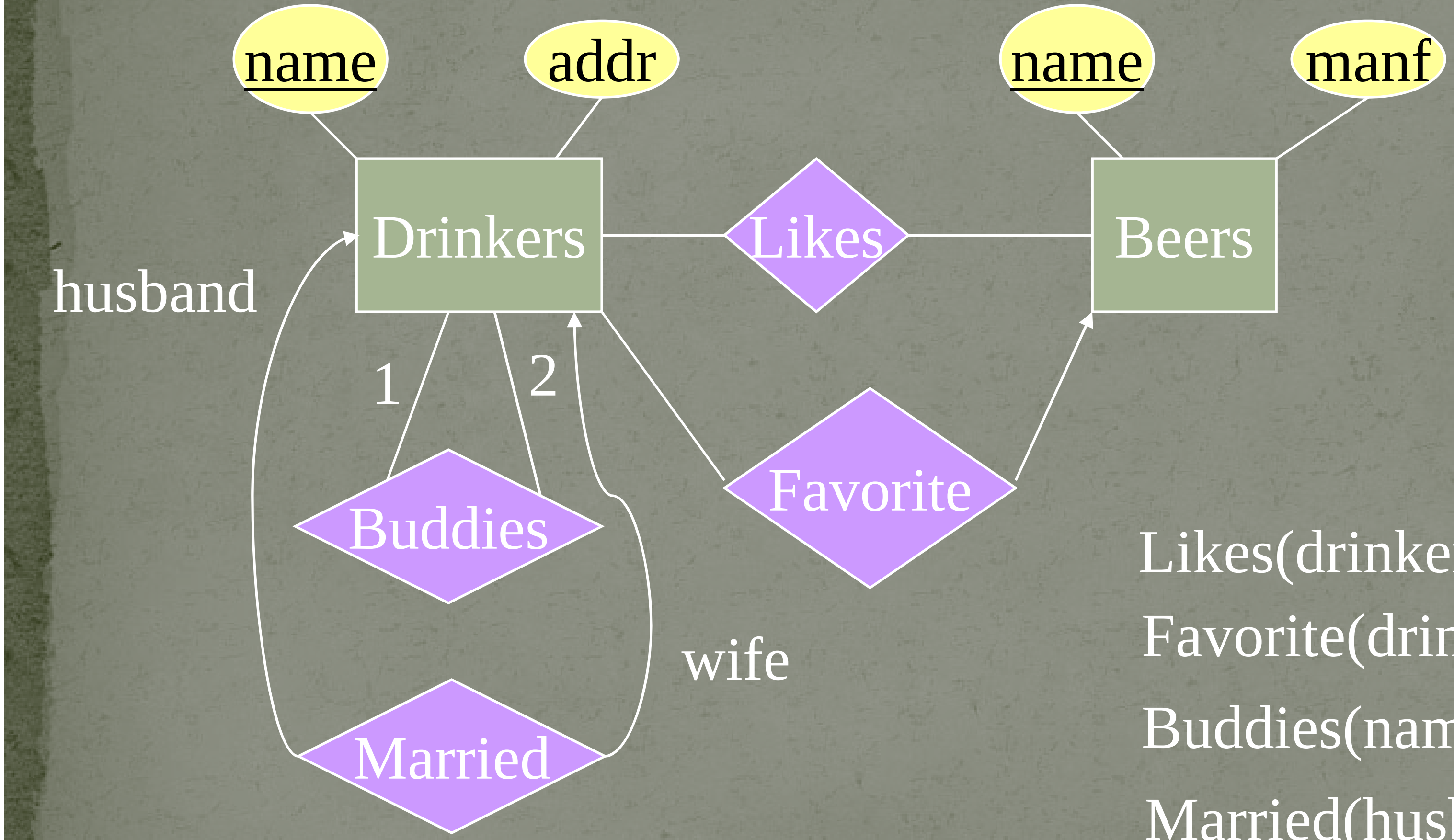
- Entity sets become relations with the same set of attributes.
- Relationships become relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.

# Entity Set -> Relation



Relation:  Beers(name, manf)

# Relationship -> Relation



Likes(drinker, beer)

Favorite(drinker, beer)

Buddies(name1, name2)

Married(husband, wife)

# Combining Relations

- It is OK to combine the relation for an entity-set *E* with the relation *R* for a many-one relationship from *E* to another entity set.
- Example: Drinkers(name, addr) and Favorite(drinker, beer) combine to make Drinker1(name, addr, favBeer).

# Risk with Many-Many Relationships

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

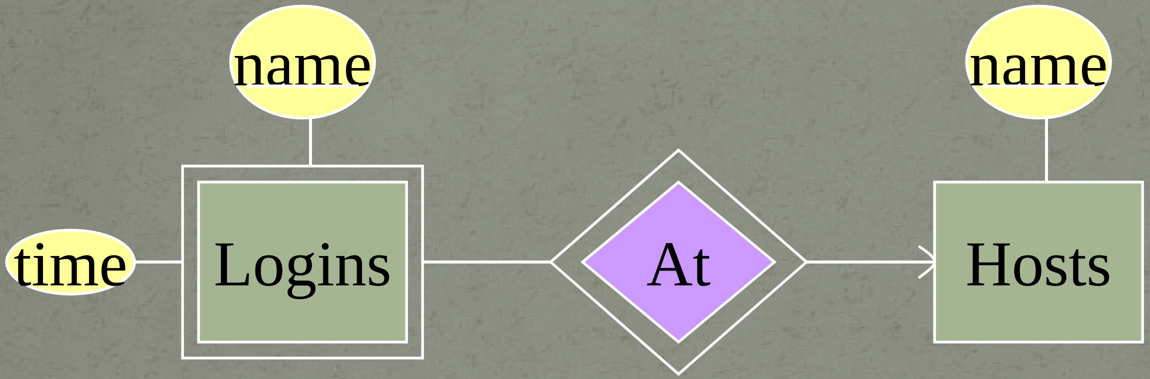
| name | addr | beer |
|---|---|---|
| Sally | 123 Maple | Bud |
| Sally | 123 Maple | Miller |

Redundancy

# Handling Weak Entity Sets

- Relation 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.

# Example



Hosts(hostName)
Logins(loginName, hostName, time)
~~At(loginName, hostName, hostName2)~~

Must be the same

At becomes part of Logins

# A (Slightly) Formal Definition

- A *database* is a collection of *relations* (or tables)
- Each *relation* is identified by a name and a list of *attributes* (or columns)
- Each *attribute* has a name and a *domain* (or type)
  - Set-valued attributes not allowed

# Schema versus instance

- Schema (metadata)
  - Specification of how data is to be structured logically
  - Defined at set-up
  - Rarely changes
- Instance
  - Content
  - Changes rapidly, but always conforms to the schema

☞ Compare to type and objects of type in a programming language

# Example

- Schema
  - *Student* (*SID* integer, *name* string, *age* integer, *GPA* float)
  - *Course* (*CID* string, *title* string)
  - *Enroll* (*SID* integer, *CID* integer)
- Instance
  - { h142, Bart, 10, 2.3i, h123, Milhouse, 10, 3.1i, ...}
  - { hCPS116, Intro. to Database Systemsi, ...}
  - { h142, CPS116i, h142, CPS114i, ...}

# Relational Integrity Constraints

- Constraints are *conditions* that must hold on *all* valid relation instances. There are four main types of constraints:

1. Domain constraints
    1. The value of a attribute must come from its domain
2. Key constraints
3. Entity integrity constraints
4. Referential integrity constraints

# Primary Key Constraints

- A set of fields is a *candidate key* for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., given a schema Student(sid: string, name: string, gpa: float) we have:
  - *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

# Key Example

- CAR (licence_num: string, Engine_serial_num: string, make: string, model: string, year: integer)
  - What is the candidate key(s)
  - Which one you may use as a primary key
  - What are the super keys

# Entity Integrity

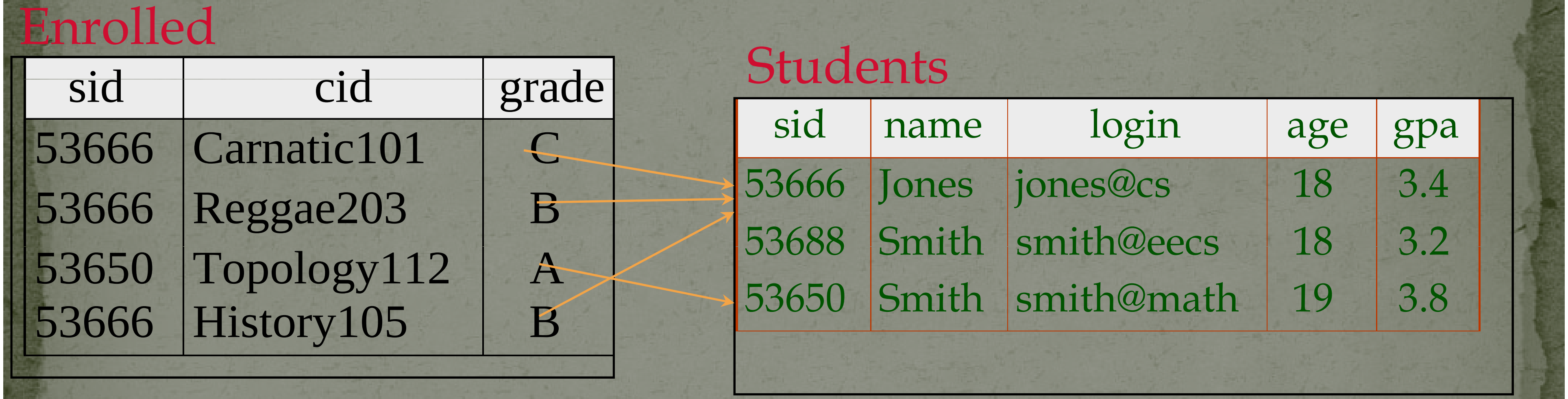
- Entity Integrity: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).
  - Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

# Foreign Keys, Referential Integrity

- *Foreign key* : Set of fields in one relation that is used to `refer’ to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer’.
- E.g. *sid* is a foreign key referring to *Students*:
  - Student(sid: string, name: string, gpa: float)
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
    - Links in HTML!

# Foreign Keys

- Only students listed in the Students relation should be allowed to enroll for courses.



Enrolled

| sid | cid | grade |
|---|---|---|
| 53666 | Carnatic101 | C |
| 53666 | Reggae203 | B |
| 53650 | Topology112 | A |
| 53666 | History105 | B |

Students

| sid | name | login | age | gpa |
|---|---|---|---|---|
| 53666 | Jones | jones@cs | 18 | 3.4 |
| 53688 | Smith | smith@eecs | 18 | 3.2 |
| 53650 | Smith | smith@math | 19 | 3.8 |

- Or, use NULL as the value for the foreign key in the referencing tuple when the referenced tuple does not exist

# Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - e.g., “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
  - A *constraint specification language* may have to be used to express these
  - SQL-99 allows triggers and ASSERTIONS to allow for some of these