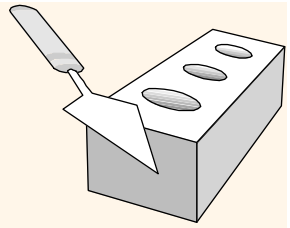
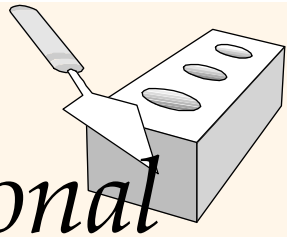


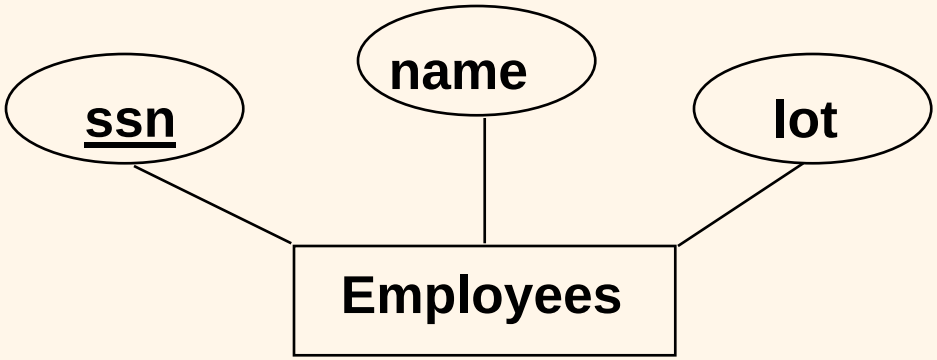
Reducing ER Schema to Relational Schema

Excerpt from
Chapter 3, "Database Management Systems" 3ed, R. Ramakrishnan and J. Gehrke

Logical DB Design: ER to Relational

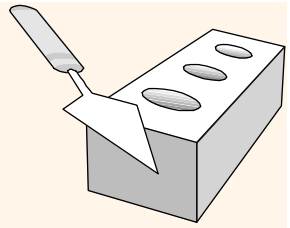


- Entity sets to tables:




```
CREATE TABLE Employees
    (ssn CHAR(11),
    name CHAR(20),
    lot  INTEGER,
    PRIMARY KEY  (ssn))
```

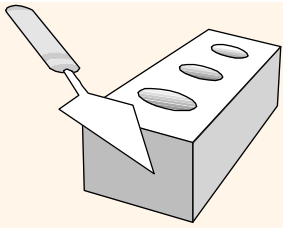
Relationship Sets to Tables



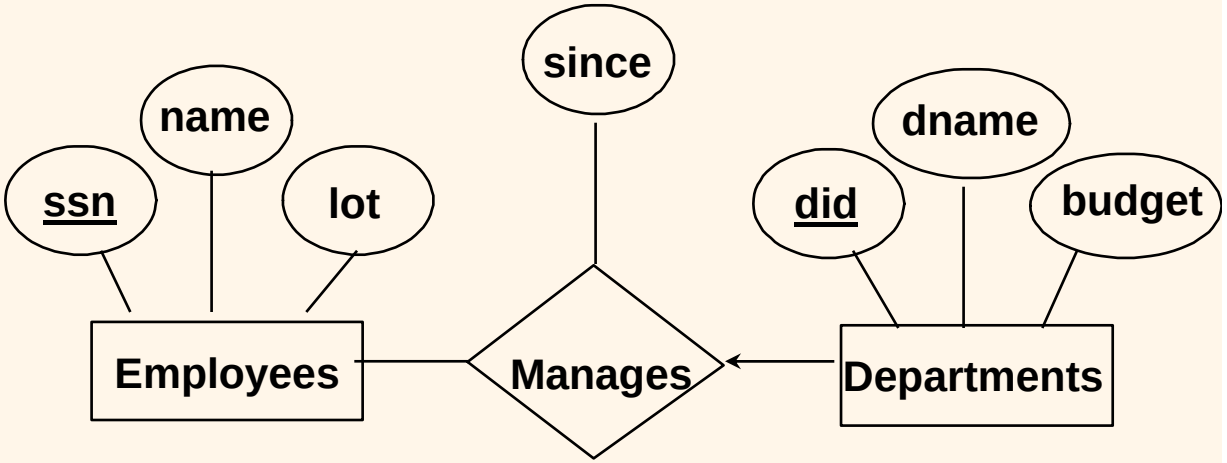
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

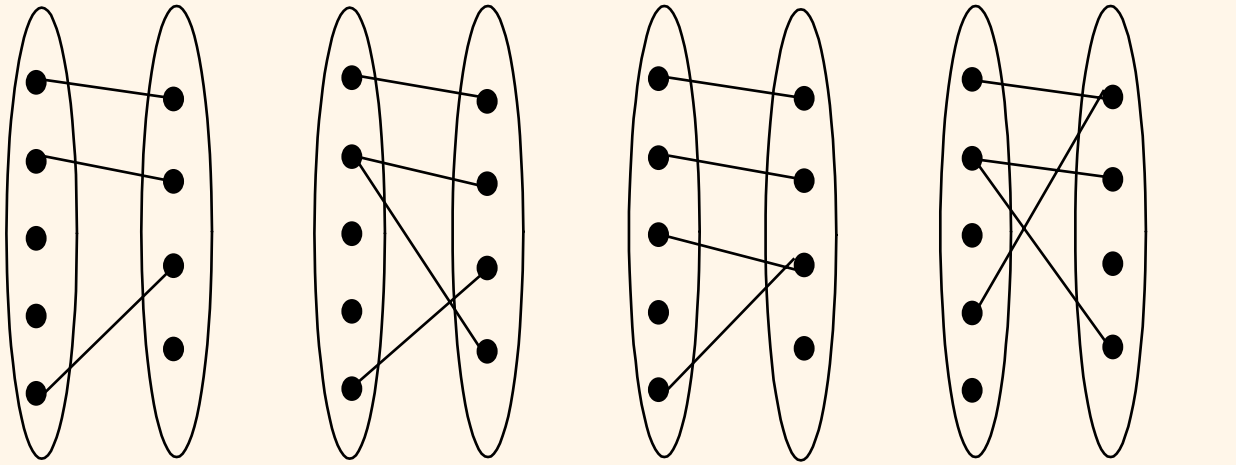
```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
      REFERENCES Employees,
 FOREIGN KEY (did)
      REFERENCES Departments)
```

Review: Key Constraints



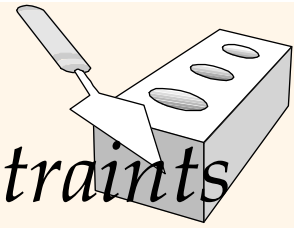
- Each dept has at most one manager, according to the *key constraint* on Manages.





Translation to relational model?

Translating ER Diagrams with Key Constraints

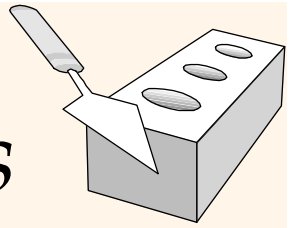


- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

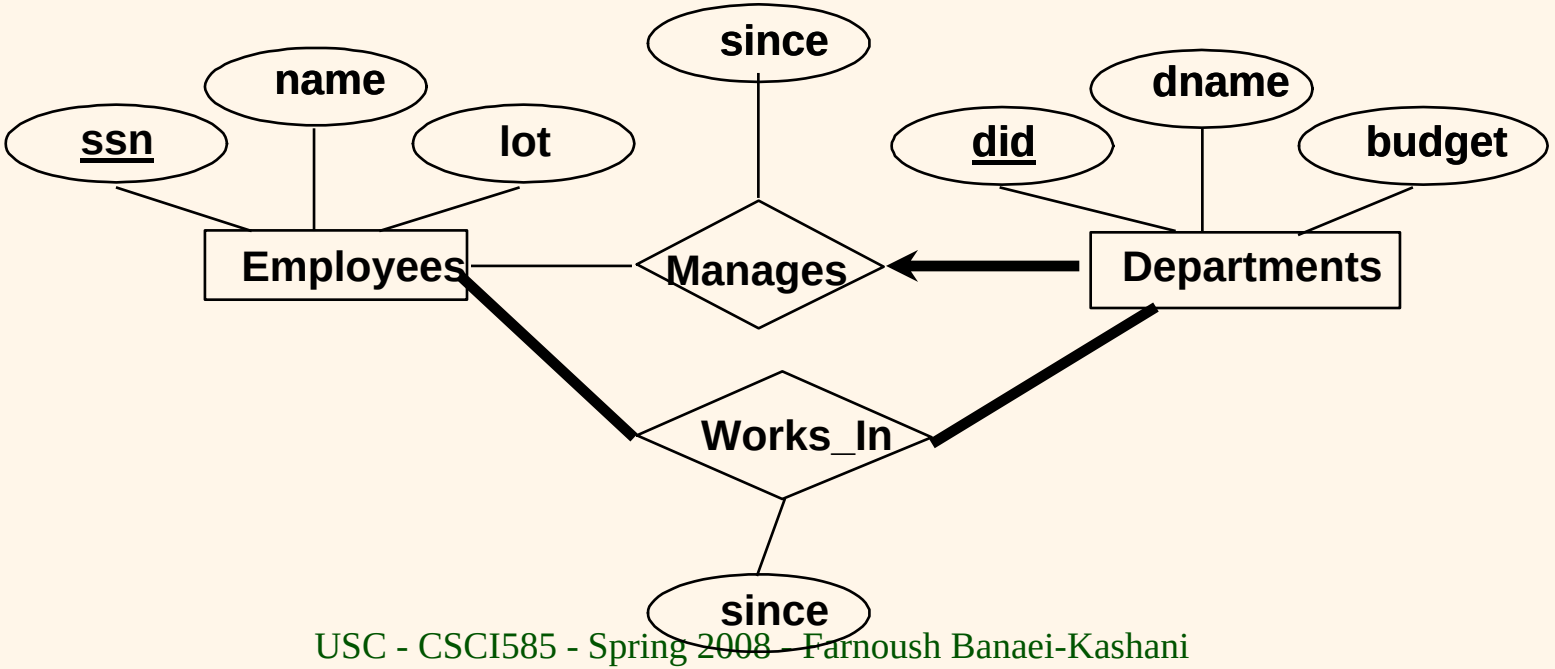
```
CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```

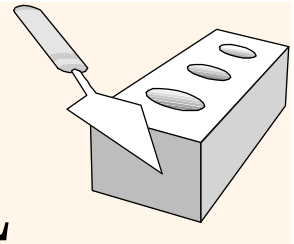
Review: Participation Constraints



- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)

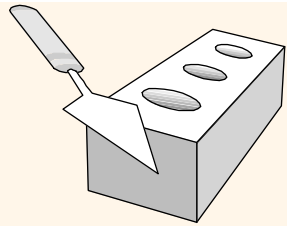


Participation Constraints in SQL



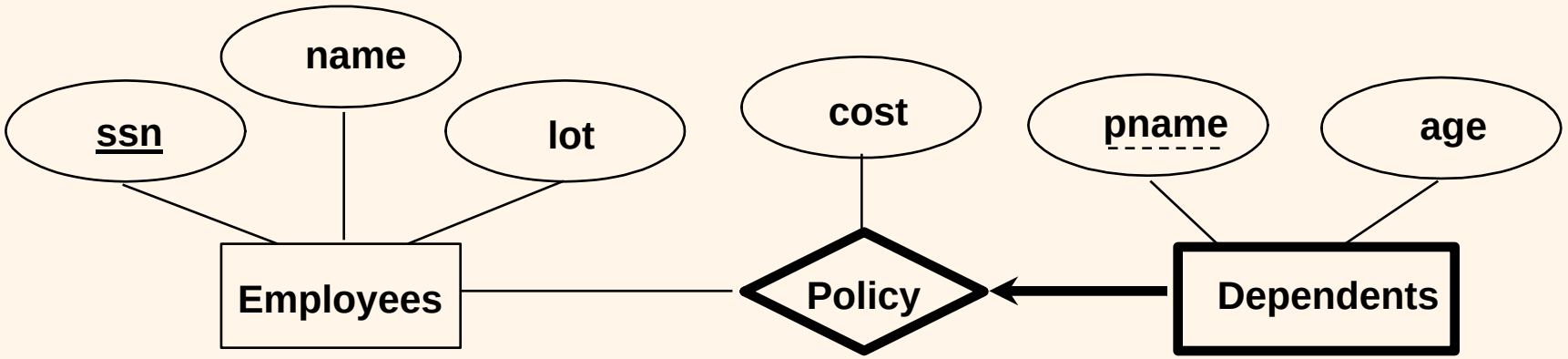
- We can capture participation constraints involving one entity set in a binary relationship

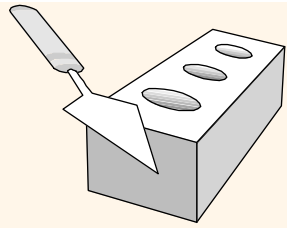
```
CREATE TABLE Dept_Mgr(
   did INTEGER,
   dname CHAR(20),
   budget REAL,
   ssn CHAR(11) NOT NULL,
   since DATE,
   PRIMARY KEY (did),
   FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE NO ACTION)
```



Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.



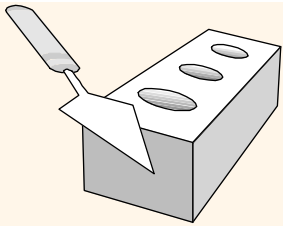


Translating Weak Entity Sets

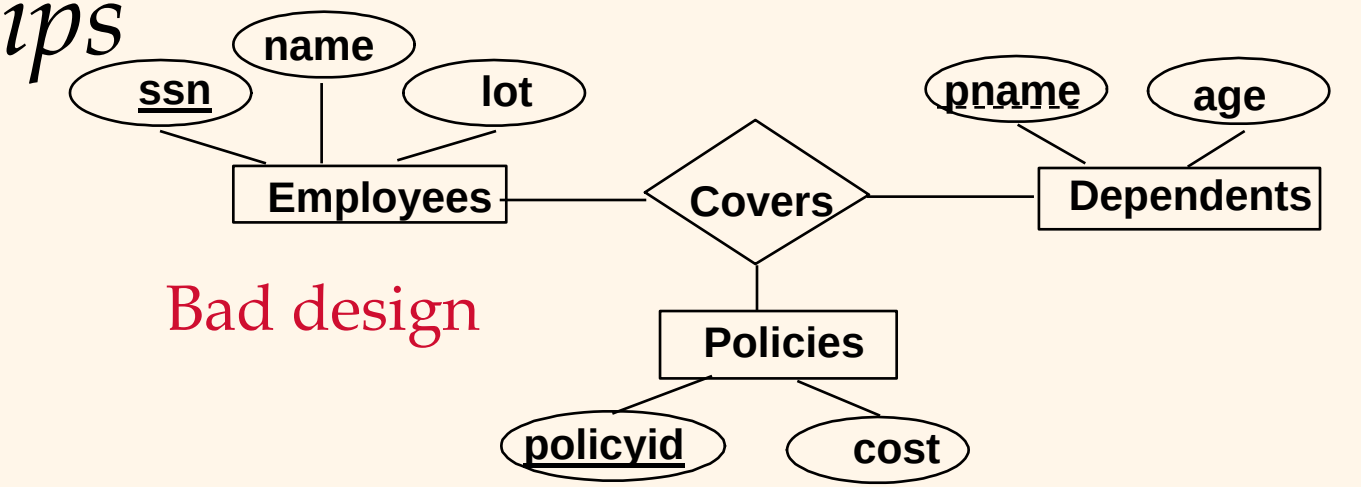
- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

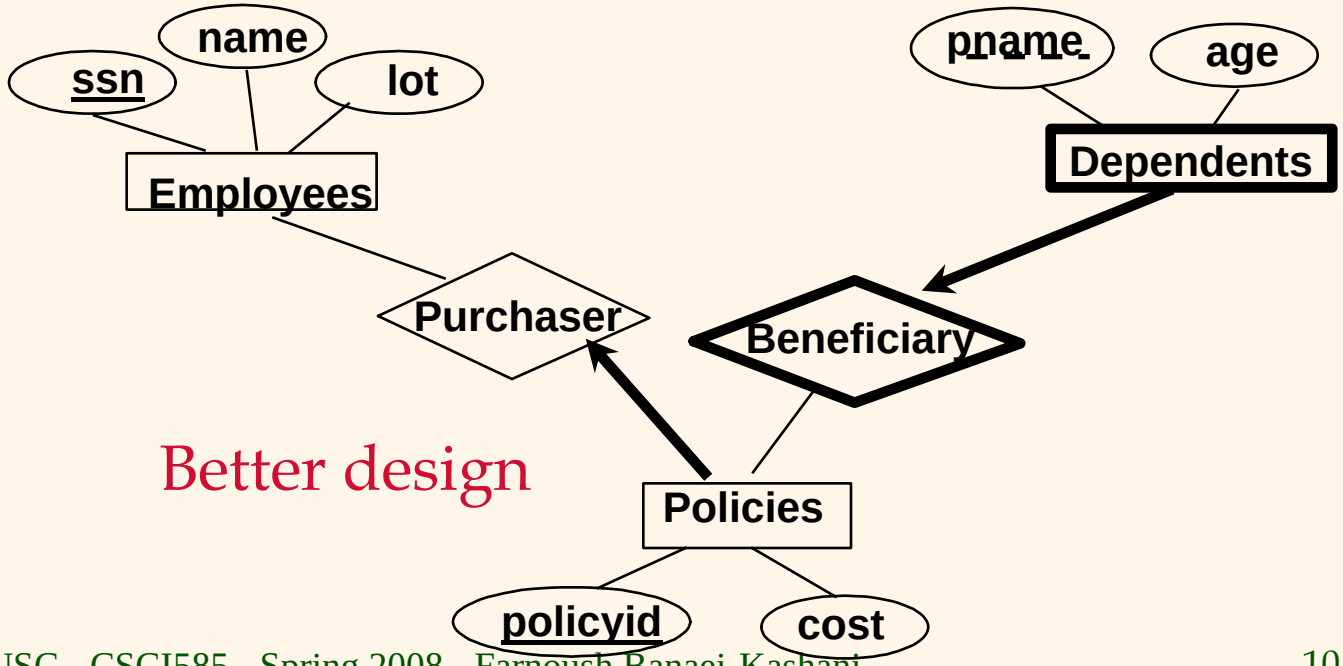
Review: Binary vs. Ternary Relationships



- Constraints:
  - each policy is owned by just 1 employee
  - every policy must be owned by some employee
  - each dependent is tied to the covering policy

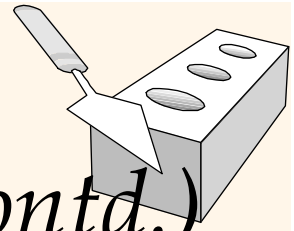


Bad design



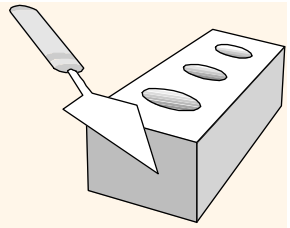
Better design

Binary vs. Ternary Relationships (Contd.)



- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.

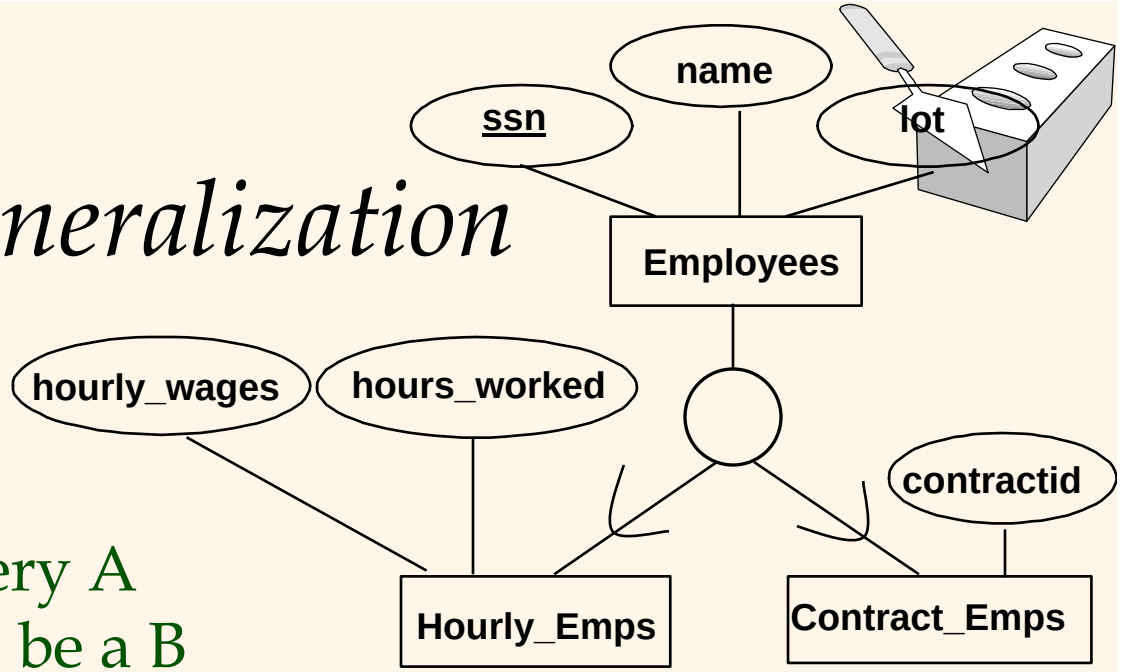
```
CREATE TABLE Policies (
    policyid  INTEGER,
    cost  REAL,
    ssn  CHAR(11)  NOT NULL,
    PRIMARY KEY (policyid).
    FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE CASCADE)

CREATE TABLE Dependents (
    pname  CHAR(20),
    age  INTEGER,
    policyid  INTEGER,
    PRIMARY KEY (pname, policyid).
    FOREIGN KEY (policyid) REFERENCES Policies,
      ON DELETE CASCADE)
```



Extended ER (EER)

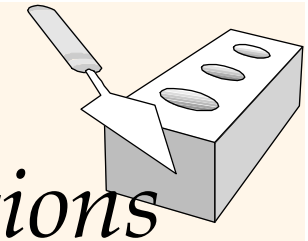
Excerpt from
Chapters 2 & 3, "Database Management Systems" 3ed, R. Ramakrishnan and J. Gehrke

Specialization/Generalization



- As in C++, or other PLs, attributes are inherited.
- If we declare A **ISA** B, every A entity is also considered to be a B entity.

- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Disjoint/Overlapping*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Total/Partial)*
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.



Translating Class Hierarchies to Relations

- ***General approach:***
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn*); must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
- Alternative: Just Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: *ssn*, *name*, *lot*, *hourly_wages*, *hours_worked*.
  - Each employee must be in one of these two subclasses.