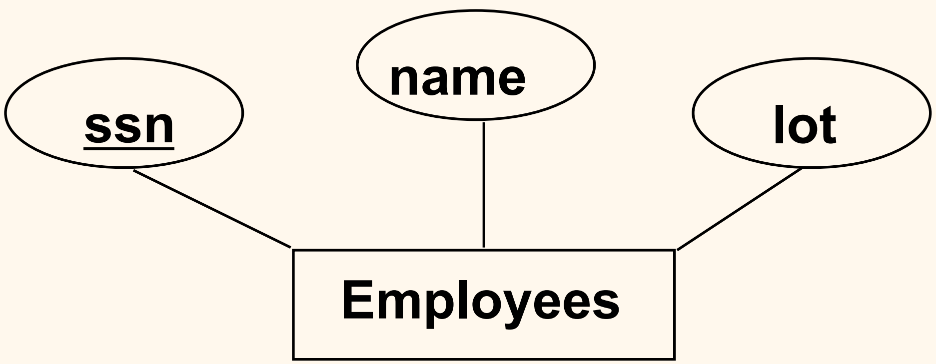
CS122A: Introduction to Data Management

Lecture #5
(E-R → Relational, Cont.)

Instructor: Chen Li

Logical DB Design: ER to Relational (Review)

- Entity sets to tables:

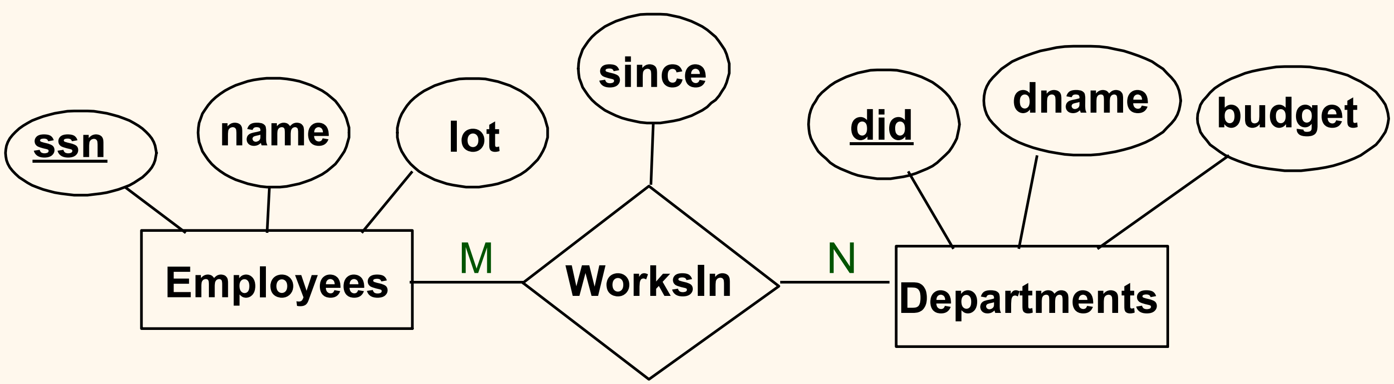



```
CREATE TABLE Employees
    (ssn CHAR(11),
    name CHAR(20),
    lot INTEGER,
    PRIMARY KEY (ssn))
```

Relationship Sets to Tables (Review)

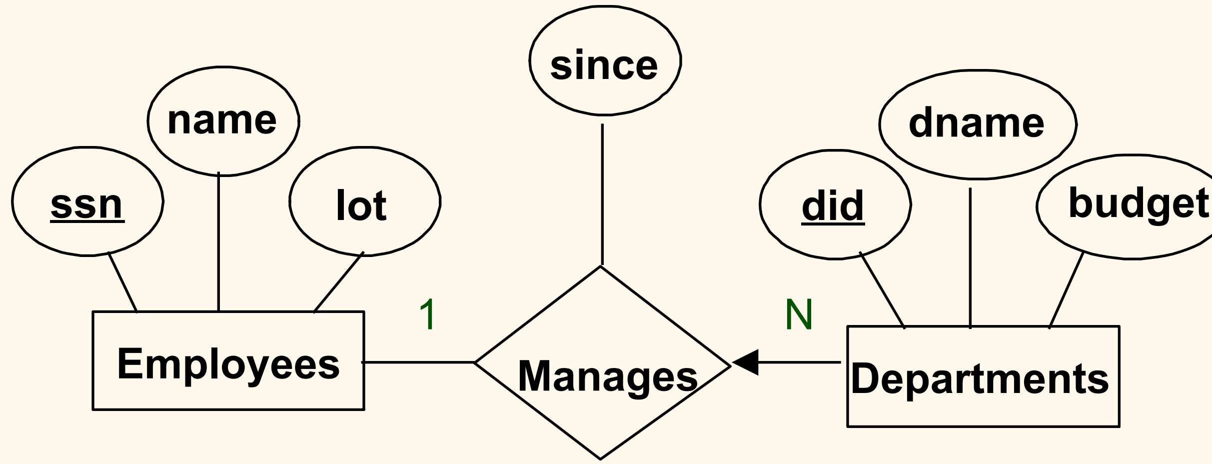
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - Note: This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

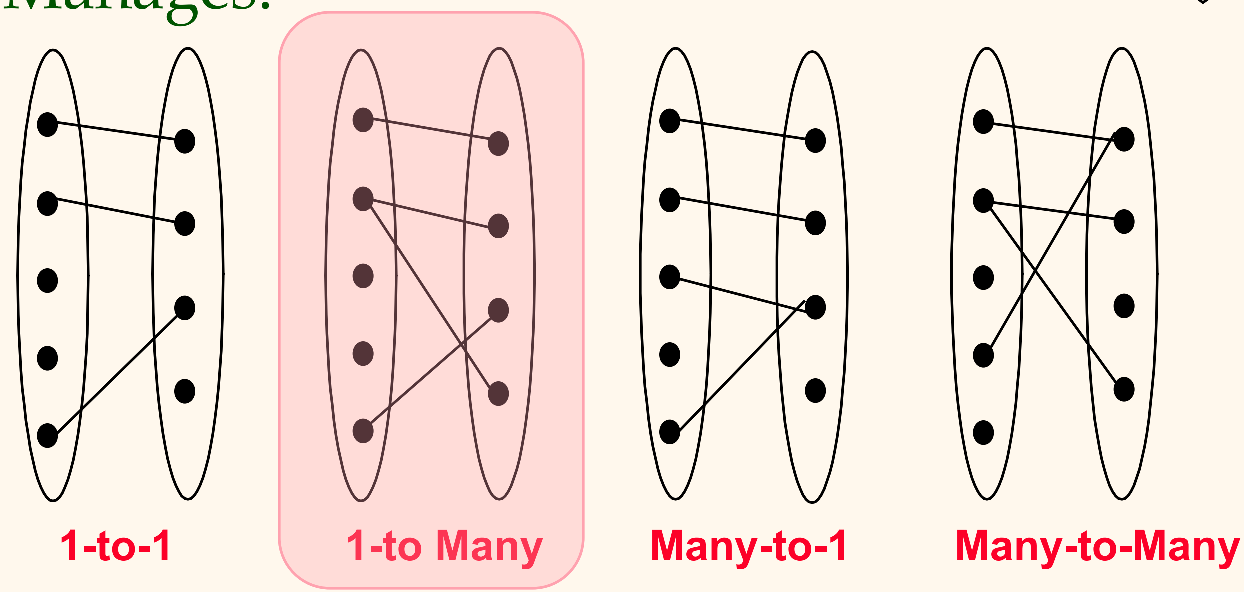
```
CREATE TABLE Works_In(
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
     REFERENCES Employees,
  FOREIGN KEY (did)
     REFERENCES Departments)
```



Key Constraints (Review)

- 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 the relationship to a table (Manages):
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- But, since *each* department has a *unique* manager, we could choose to fold Manages right into Departments.

(Q: Why do that...?)

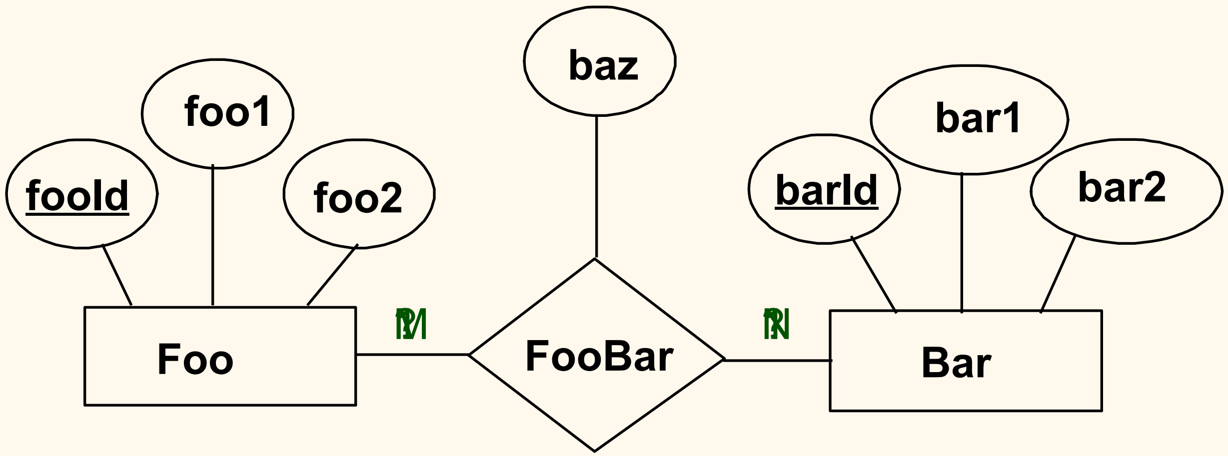
```
CREATE TABLE  Manages (
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

vs.

```
CREATE TABLE  Departments2 (
  did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  mgr_ssn  CHAR(11),
  mgr_since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY (mgr_ssn) REFERENCES Employees)
```

Note*: The relationship info has been pushed to the N-side's entity table!*

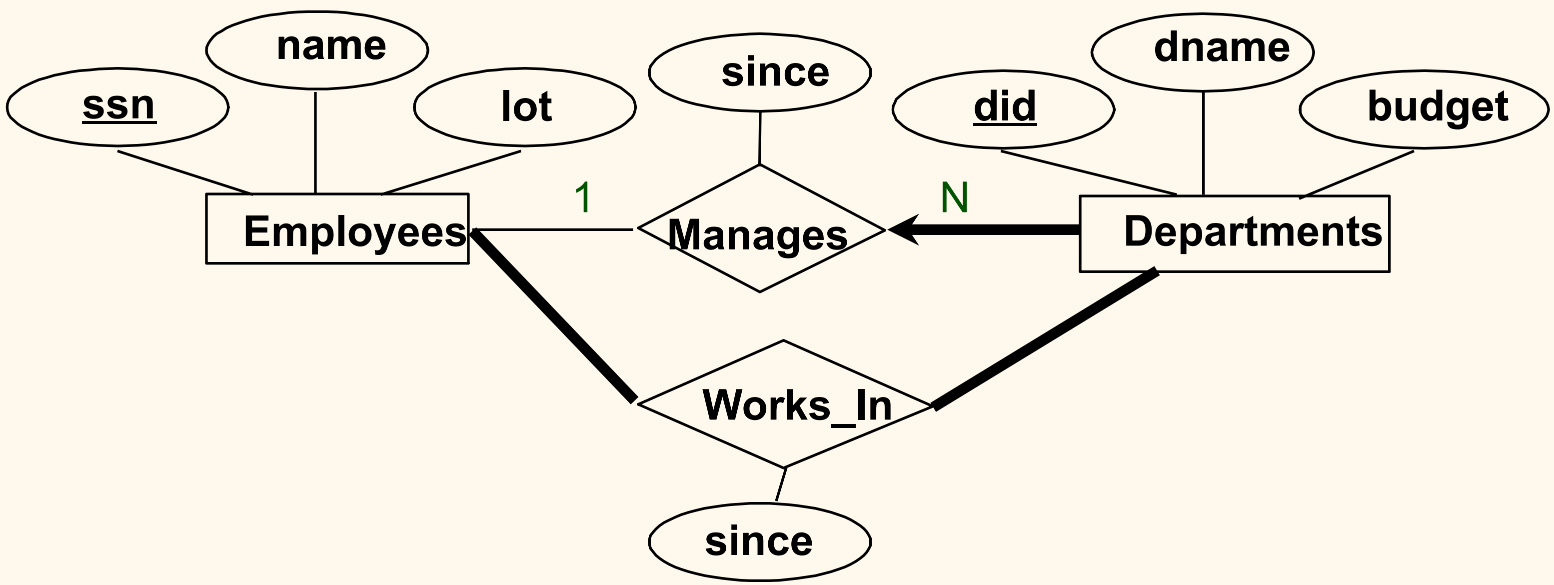
Properly Reflecting Key Constraints



- So what are the translated relationship table keys (*etc.*) when…
  - FooBar is M:N? → FooBar(fooId, barId, baz)
  - FooBar is N:1? → FooBar(fooId, barId, baz)
  - Foobar is 1:N? → FooBar(fooId, barId, baz)
  - Foobar is 1:1? → FooBar(fooId, barId, baz) *(Note: unique)*

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!!)



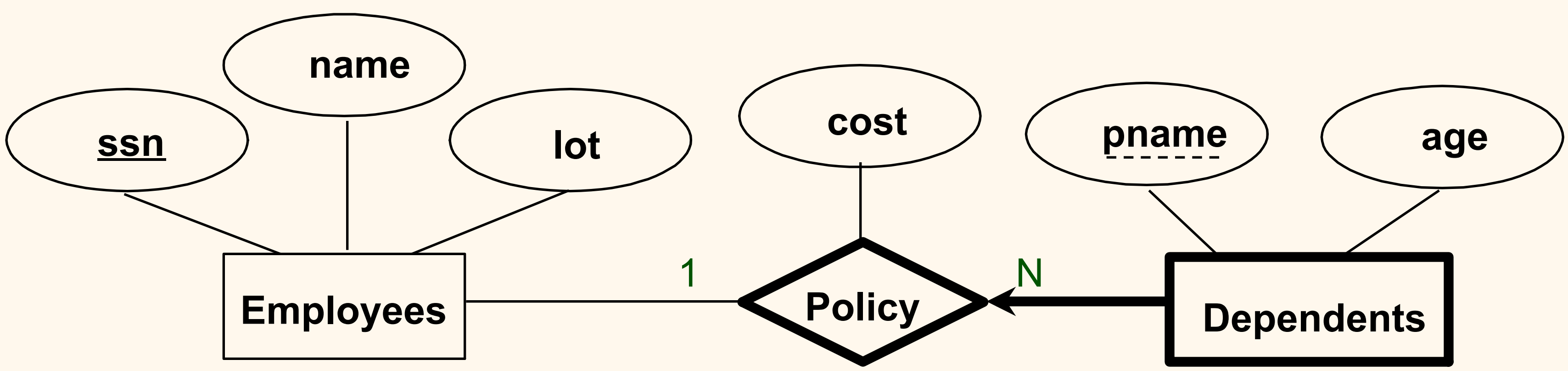
Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to the use of *triggers*).

```
CREATE TABLE  Department2 (
  did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  mgr_ssn  CHAR(11) NOT NULL,
  mgr_since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY  (mgr_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.

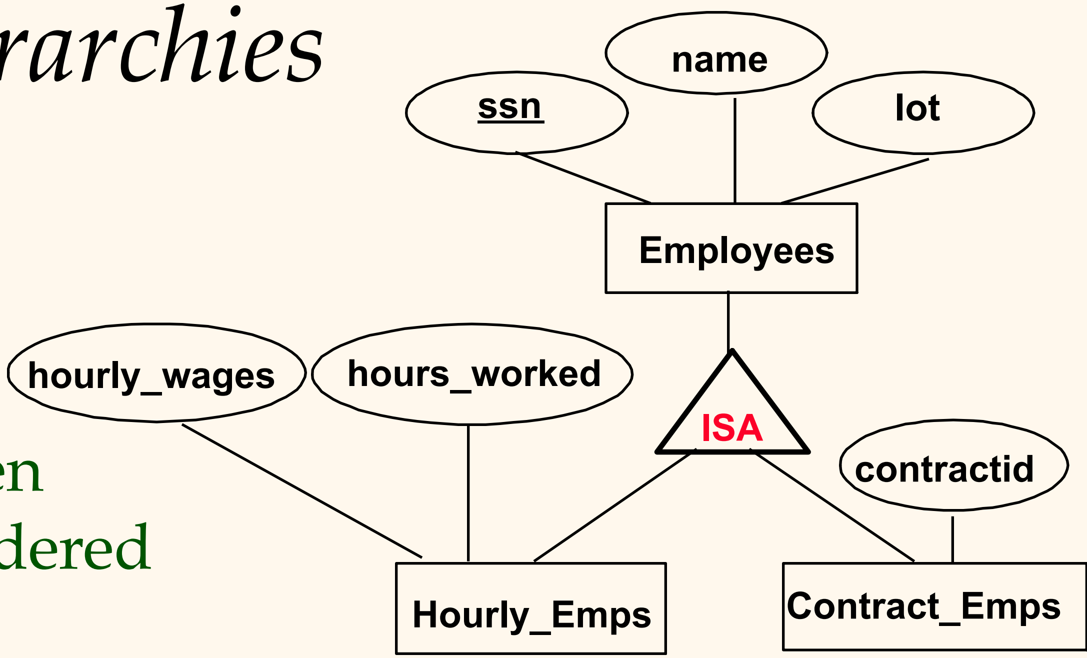


Translating Weak Entity Sets

- 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 Dependents2 (
  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: ISA Hierarchies



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

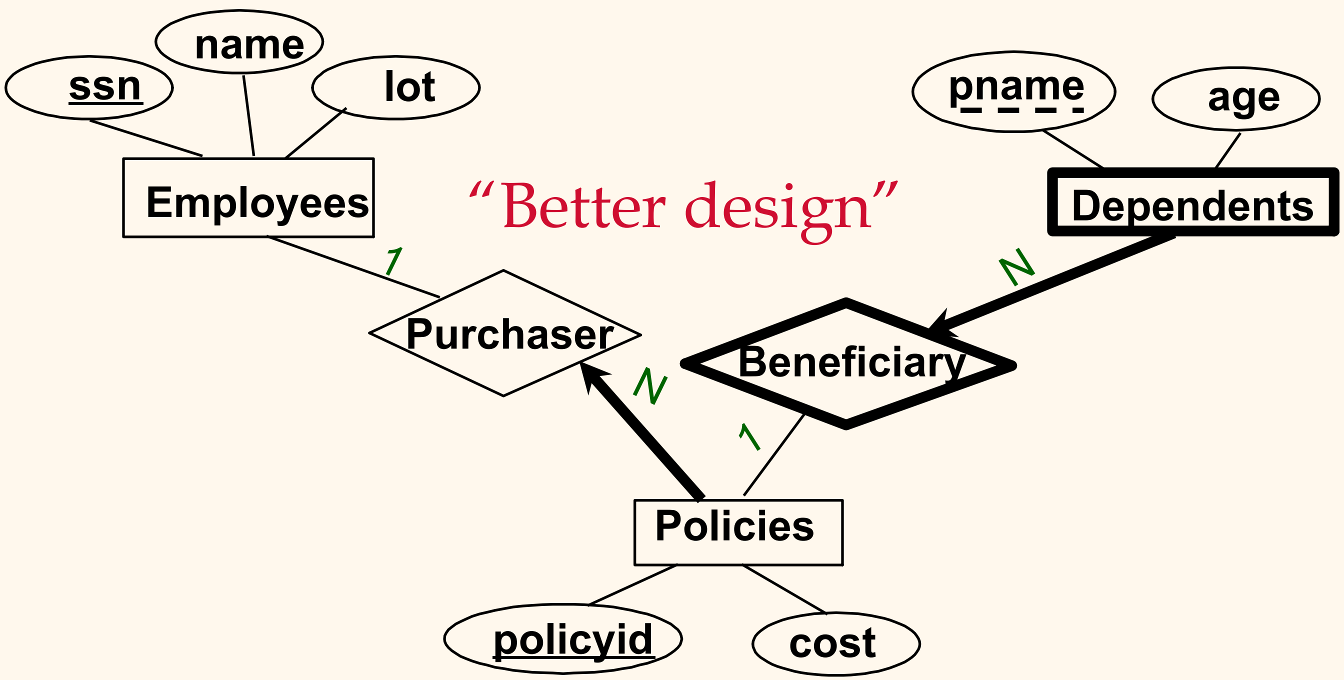
- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

Translating ISA Hierarchies to Relations

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

  (Q: Can we always do this, then? A: Not w/o redundancy!)

Review: Binary vs. Ternary Relationships



Binary vs. Ternary Relationships (Contd.)

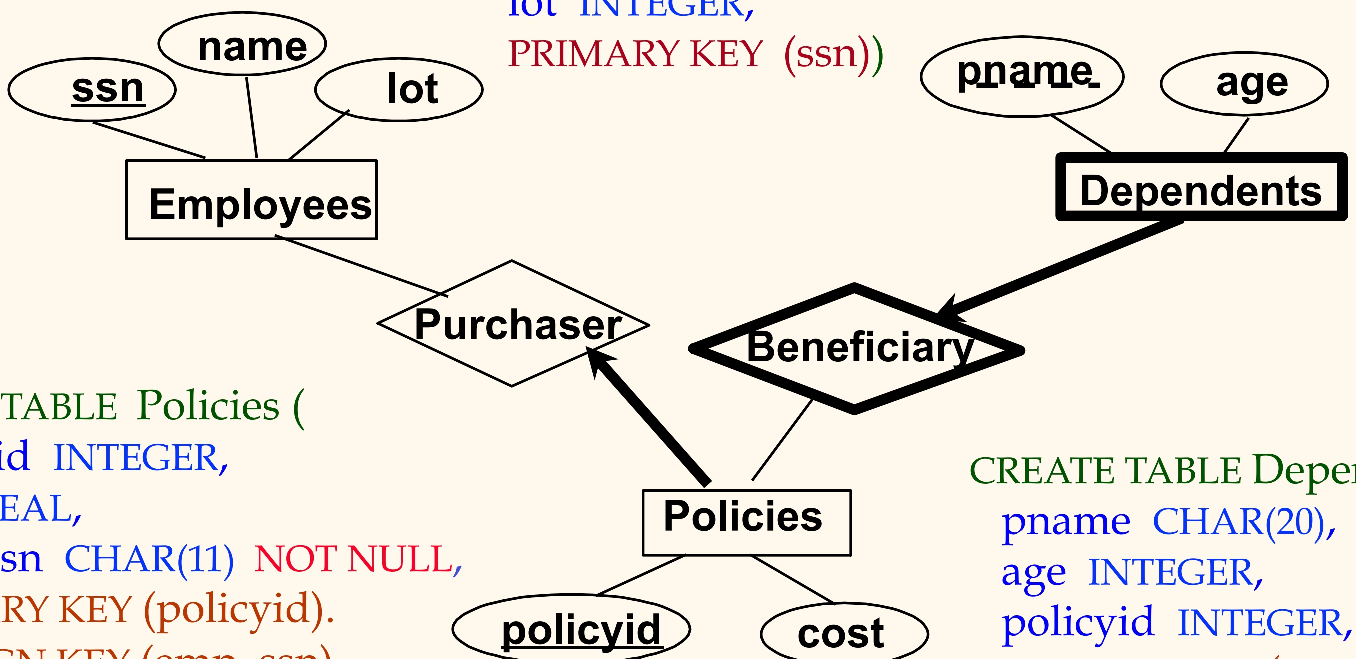
- The key constraints let us combine *Purchaser with Policies* and *Beneficiary with Dependents*.
- Participation constraints lead to NOT NULL constraints.
  (*Note:* Primary key attributes are NOT NULL as well – check documentation to see if that's implicit or explicit!

```
CREATE TABLE  Policies (
  policyid  INTEGER,
  cost  REAL,
  emp_ssn  CHAR(11)  NOT NULL,
  PRIMARY KEY (policyid).
  FOREIGN KEY (emp_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)
```

Review: Binary vs. Ternary Relationships

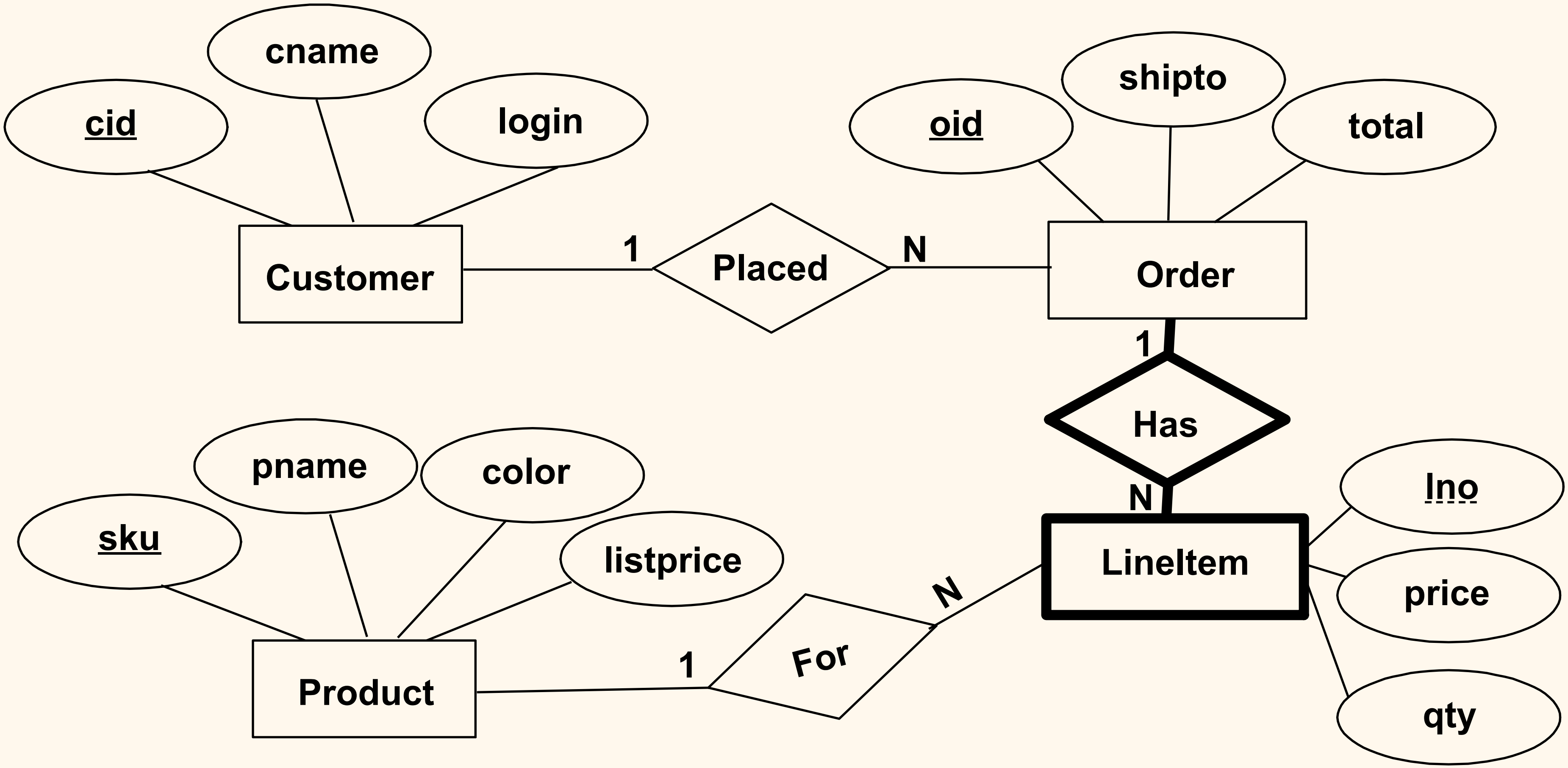
```
CREATE TABLE Employees (
  ssn CHAR(11),
  name CHAR(20),
  lot  INTEGER,
  PRIMARY KEY  (ssn))
```




```
CREATE TABLE  Policies (
  policyid  INTEGER,
  cost  REAL,
  emp_ssn  CHAR(11)  NOT NULL,
  PRIMARY KEY (policyid).
  FOREIGN KEY (emp_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)
```

An Example: Putting It Together



Putting It Together (Cont'd.)

```
CREATE TABLE Customer (
  cid INTEGER,
  cname VARCHAR(50),
  login VARCHAR(20)
    NOT NULL,
  PRIMARY KEY (cid),
  UNIQUE (login))

CREATE TABLE Product (
  sku INTEGER,
  pname VARCHAR(100),
  color VARCHAR(20),
  listprice DECIMAL(8,2),
  PRIMARY KEY (sku))

CREATE TABLE Order (
  oid INTEGER,
  custid INTEGER,
  shipto VARCHAR(200),
  total DECIMAL(8,2),
  PRIMARY KEY (oid),
  FOREIGN KEY (custid) REFERENCES Customer))

CREATE TABLE LineItem (
  oid INTEGER,
  lno INTEGER,
  price DECIMAL(8,2),
  qty INTEGER,
  sku INTEGER,
  PRIMARY KEY (oid, lno),
  FOREIGN KEY (oid) REFERENCES Order
     ON DELETE CASCADE),
  FOREIGN KEY (sku) REFERENCES Product))
```

Putting It Together (Cont'd.)

Customer

cid	cname	login
1	Smith, James	jsmith@aol.com
2	White, Susan	suzie@gmail.com
3	Smith, James	js@hotmail.com

Product

sku	pname	color	listprice
123	Frozen DVD	null	24.95
456	Graco Twin Stroller	green	199.99
789	Moen Kitchen Sink	black	350.00

Order

oid	custid	shipto	total
1	3	J. Smith, 1 Main St., USA	199.95
2	1	Mrs. Smith, 3 State St., USA	300.00

LineItem

oid	lno	price	qty	item
1	1	169.95	1	456
1	2	15.00	2	123
2	1	300.00	1	789

SQL Views

- A *view* is just a relation, but we store its *definition* rather than storing the (materialized) set of tuples.

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age < 21
```

- Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

Views and Security

- Views can be used to present necessary information (or a summary) while hiding some details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students *S* who have are enrolled, but not the *cid's* of the courses they are enrolled in.
- Other view uses in our ER translation context might include:
  - Derived attributes, e.g., age (vs. birthdate)
  - Simplifying/eliminating join paths (for SQL)

Relational Model and E-R Schema Translation: Summary

- A tabular representation of data.
- Simple and intuitive, also widely used.
- Integrity constraints can be specified by the DBA based on application semantics. DBMS then checks for violations.
  - Two important ICs: Primary and foreign keys (PKs, FKs).
  - In addition, we *always* have domain constraints.
- Powerful and natural query languages exist (soon!)
- Rules to translate E-R to relational model
  - Can be done by a human, or automatically (using a tool)