The Relational Model

Computer Science Department
Columbia University

Why Study the Relational Model?

- By far, the most widely used model. Major vendors:
  - Oracle, 44% market share as of end 2011 (source: Gartner)
  - IBM (DB2 and Informix), 22%
  - Microsoft (SQL Server), 18%
  - SAP (bought Sybase in 2010), 4.1%
  - Teradata, 3.3%
- Other models exist
  - "Legacy systems" in older DBMSs, e.g., IBM's IMS
  - Object-oriented model (based on some OO language)
  - NoSQL
  - Application specific models (limited use); e.g., event-oriented
  - *Object-Relational model* - most relational DBMS currently employ OO concepts.

Relational Database: Definitions

- *Relational database:* a set of *relations (tables)*
- A relation (table) is a set of tuples (rows)
  - *Schema* : specifies the structure: name of relation, name and type of each column, constraints.
    - e.g. Students(*sid* string, *name* string, *login* string, *age* integer, *gpa* real).
    - #columns= *degree / arity*
  - *Instance* : the actual *table* (i.e., its rows) at a particular point in time.
- All rows in a table are distinct.

Example Instance of Students Relation

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

- All rows distinct
- Do all columns in a relation instance have to be distinct? (How about all column names?)

Relational Query Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - …
  - SQL-99 (major extensions including some OO concepts, current standard)

The SQL Query Language

Example:

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

- To find all 18 year old students, we can write:

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

Creating Relations in SQL

- Creates the Students relation. The type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa REAL)
```

- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2))
```

Destroying and Altering Relations

DROP TABLE Students

- Destroys the relation Students. The schema information *and* the tuples are deleted.

ALTER TABLE Students
ADD COLUMN firstYear: integer

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

** Powerful variants of these commands are available; more later!*

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Primary Key Constraints

- A set of fields is a ***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*.
  - *Candidate key*: any of the keys
  - *Primary key*: one of the keys chosen DBA
- e.g., *sid* is a key for Students. *name* is not. The set {*sid, gpa*} is a superkey.

SQL – Primary/Unique/Not null

- PRIMARY KEY (att1, att2, …), only one primary key
  UNIQUE (att1, att2, …)
  att NOT NULL

- "For a given student and course, there is a single grade." vs. "students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Students (
  …
  age INTEGER NOT NULL)
```

```
CREATE TABLE Enrolled (
  sid CHAR(20)
  cid  CHAR(20),
  grade CHAR(2),
  PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled (
  sid CHAR(20)
  cid  CHAR(20),
  grade CHAR(2),
  PRIMARY KEY  (sid),
  UNIQUE (cid, grade) )
```

Foreign Keys, Referential Integrity

- *Foreign key* : Set of fields in one tuple that is used to `refer' to another tuple. (Must correspond to primary key of the referenced tuple.) Like a `logical pointer'.
- *Referential integrity* is achieved if all foreign key constraints are enforced, i.e., no dangling references.
  - Can you name a data model without referential integrity?
    - Links in HTML!

SQL - Foreign Keys

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

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

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

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null*, denoting `*unknown*' or `*inapplicable*'.)
- Similar if primary key of Students tuple is updated.

SQL - Enforcing referential Integrity

- SQL/99 supports all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
        ON DELETE CASCADE
        ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.

Views

- A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungGoodStudents (sid, name, gpa) AS (
        SELECT S.sid, S.name, E.gpa
        FROM Students S
        WHERE S.age < 20 and S.gpa >= 3.0)
```

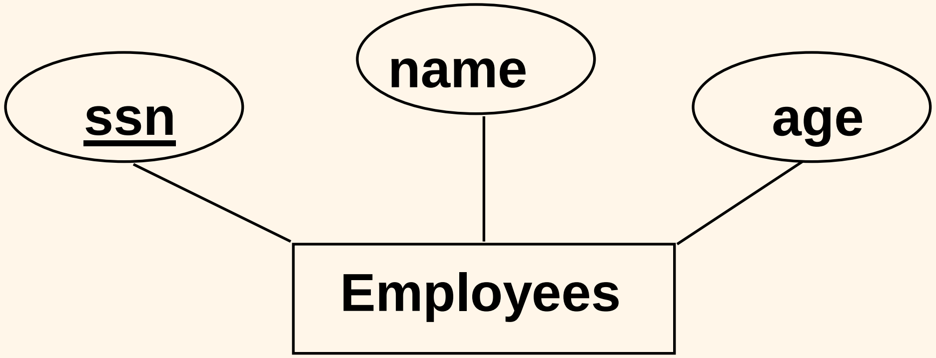
- Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students who are enrolled, but not the *cid's* of the courses they are enrolled in.

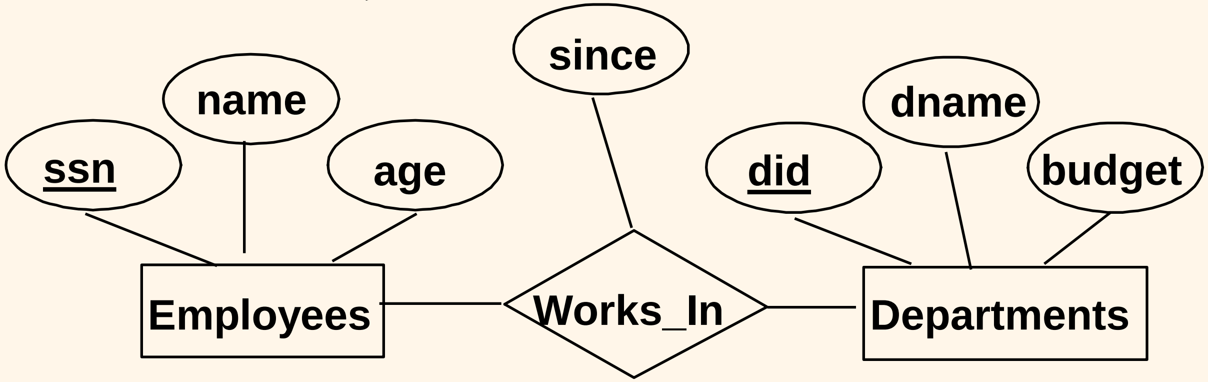
Logical DB Design: ER to Relational

- Entity sets to tables:




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

Relationship Sets to Tables

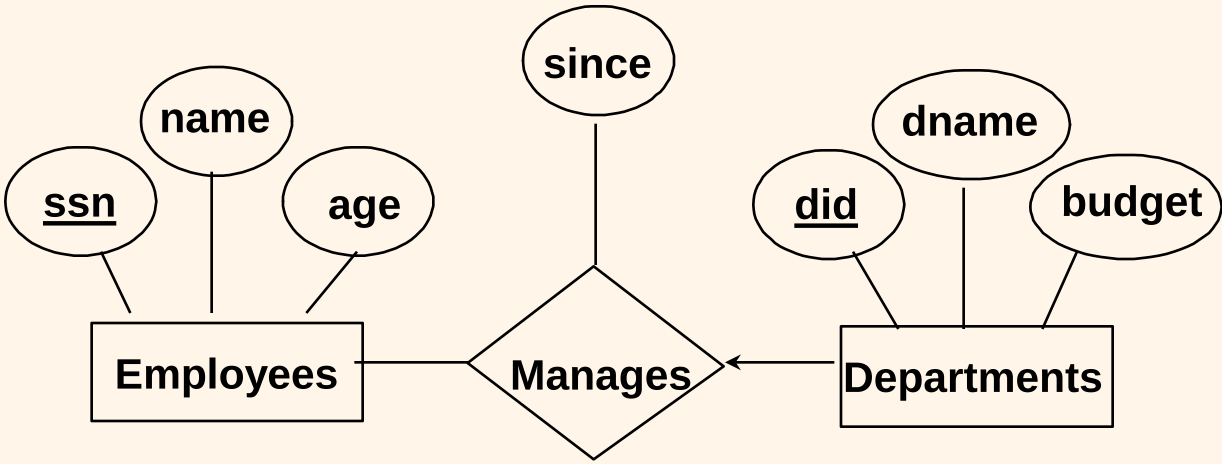


- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys
  - All descriptive attributes.

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

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



Translation to relational model?

Translating ER Diagrams with Key Constraints

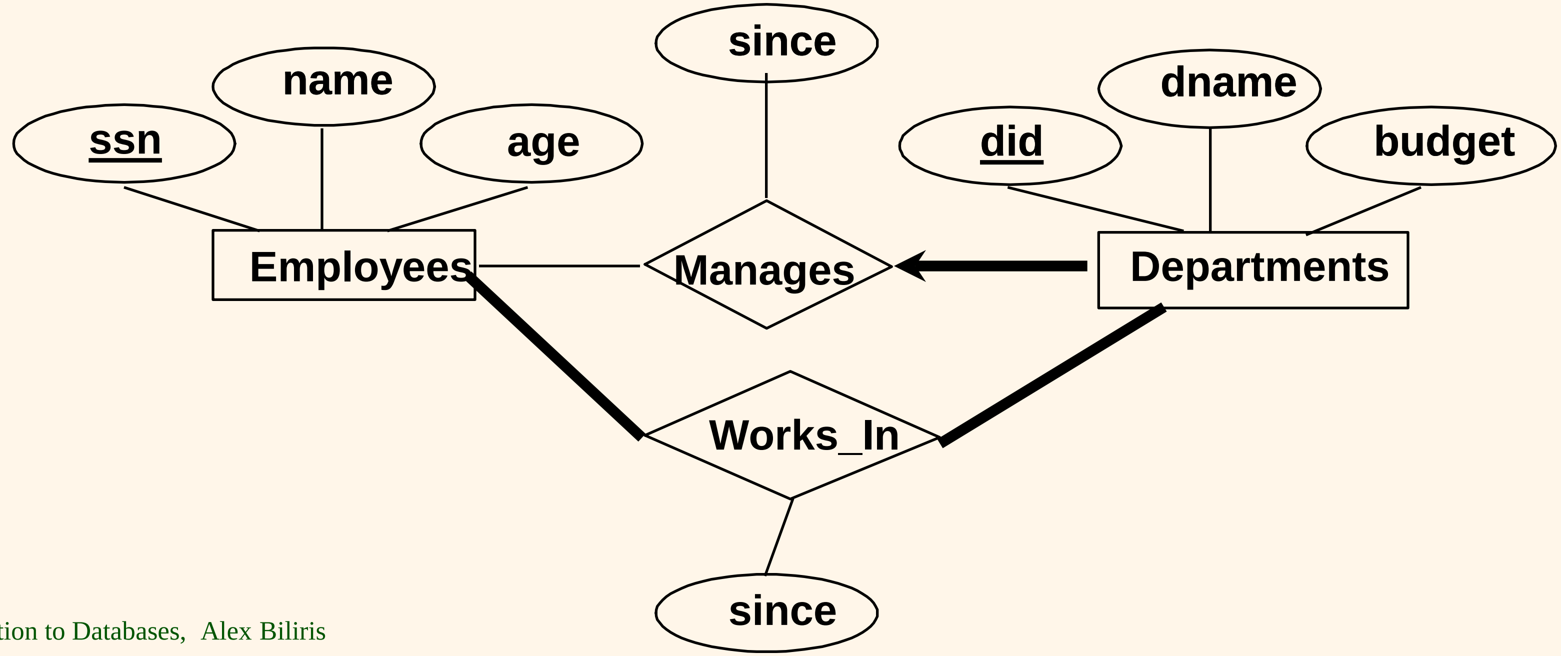
1. Map relationship to a table:
   - Note that did is the key now!
   - Separate tables for Employees and Departments.
2. Since each department has a unique manager, we could instead combine Manages and Departments.

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

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

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 CHECK constraints).

```
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)
```

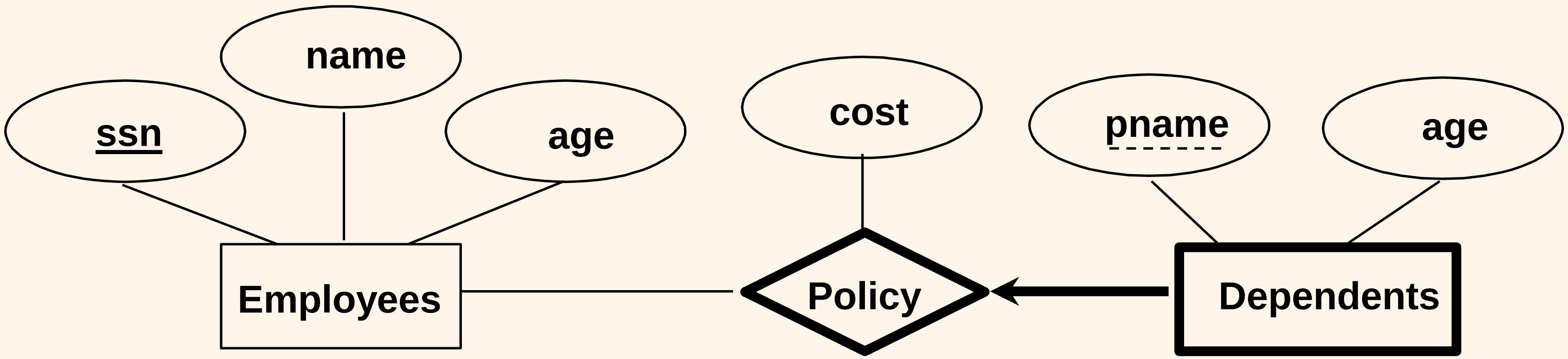
Would this work???

```
CREATE TABLE Manages(
  ssn CHAR(11) NOT NULL,
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

Not exactly the same – Why?

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),
  PRIMARY KEY (pname, ssn),
  FOREIGN KEY (ssn) REFERENCES Employees
    ON DELETE CASCADE)
```

A quick note

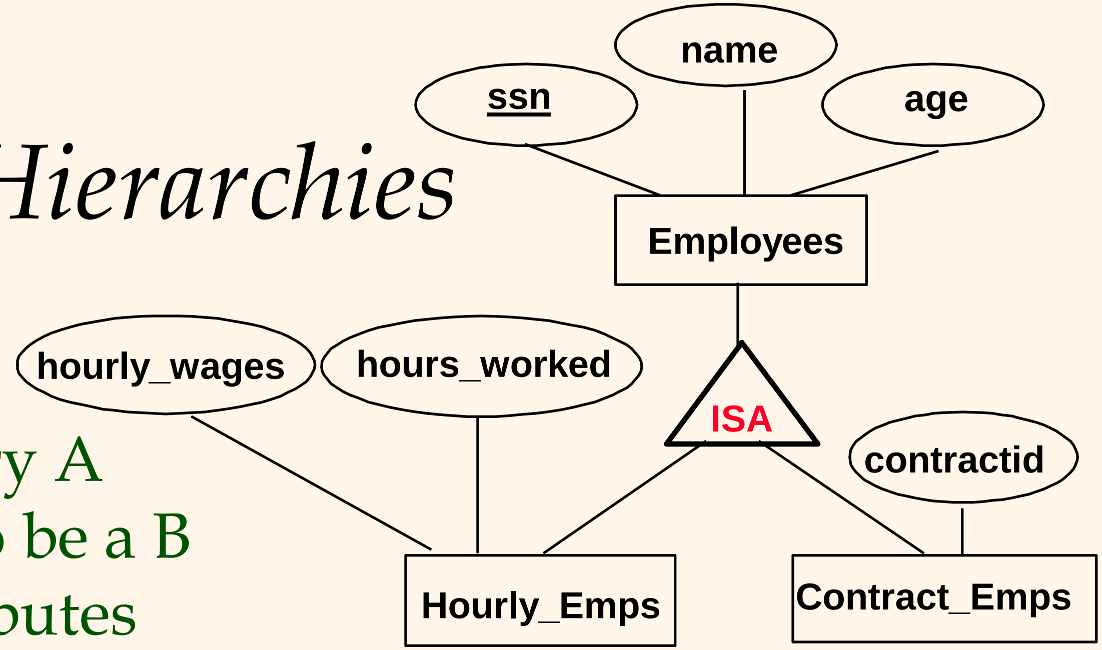
During the initial phases of the DB design process if the attribute types are not that important in clarifying the design, we may write our table definitions in a more compact form by omitting the types; e.g., we may write the previous example as follows:

Dept_Policy (pname, ssn, age, cost, PK(pname, ssn),
FK(ssn) → Employees on delete cascade)

Dept_Policy (<u>pname, ssn</u>, age, cost,
FK(ssn) → Employees on delete cascade)

[But without omitting the always crucial PK, UNIQUE, NOT NULL and FK constraints]

Review: ISA Hierarchies



If we declare A **ISA** B, every A entity is also considered to be a B entity. As in OO PLs, attributes are inherited.

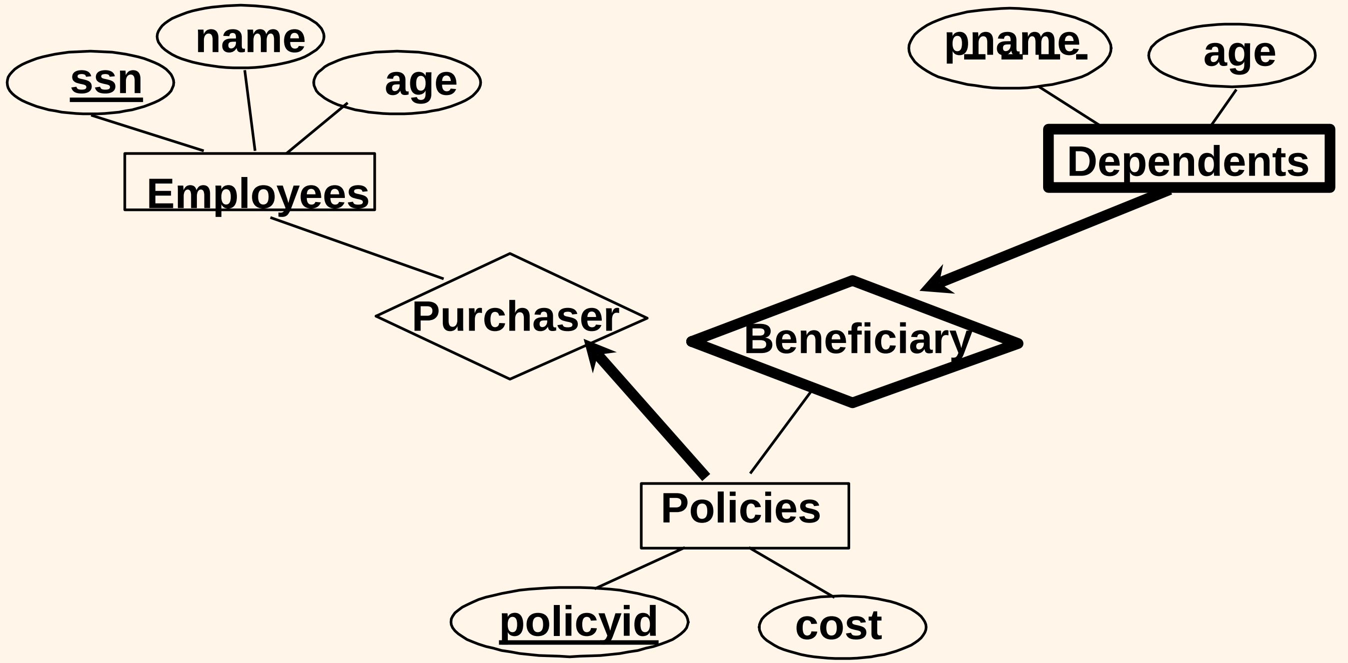
- *Overlap constraints*: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

Translating ISA Hierarchies to Relations

1. General approach (one table/set):
   - Emps(ssn, name, age, PK(ssn))
   - Hourly_Emps(h_wages, h_worked, ssn,
     PK(ssn), FK(ssn) → Emps on delete cascade)
   - Contract_Emps(c_id, ssn, PK(ssn), FK(ssn) → Emps on delete cascade)
   - Queries involving all employees easy; those involving just Hourly_Emps require a join to get some attributes (who does the join?)
2. Or, may be:
   - Hourly_Emps(<u>ssn</u>, name, age, h_wages, h_worked)
   - Contract_Emps(<u>ssn</u>, name, age, c_id)
   - Queries involving just Hourly_ or Contract_Emps easy; those involving all Emps require a join (by whom?)

- Effect of overlap/covering constraints?

Multiple relationships - ER diagram



Multiple relationships – SQL mapping

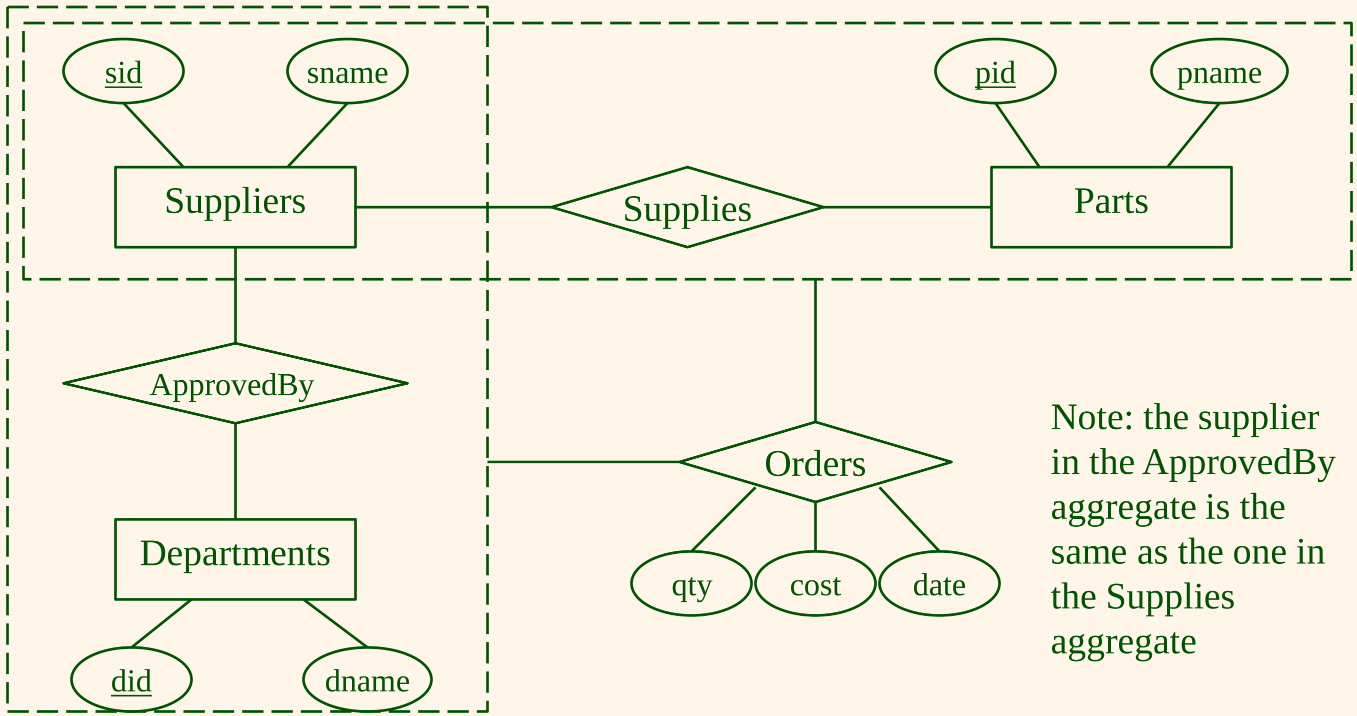
- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.

```
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)
```

Orders example - ER diagram

- Depts, suppliers, parts, orders
- A department maintains a list of approved suppliers who are the only ones that can supply parts to the department
- Each supplier maintains a list of parts that can supply



Note: the supplier in the ApprovedBy aggregate is the same as the one in the Supplies aggregate

Orders example – SQL mapping

Suppliers (<u>sid</u>, sname)

Parts (<u>pid</u>, pname)

Departments (<u>did</u>, dname)

ApprovedBy (<u>sid, did</u>, FK(sid) → Suppliers, FK(did) → Departments)

Supplies (<u>sid, pid</u>, FK(sid) → Suppliers, FK(pid) → Parts)

Orders (<u>sid, did, pid, qty, cost, date</u>,
FK(sid, did) → ApprovedBy,
FK(sid, pid) → Supplies)

[Choice of PK in Orders has significant implications on what is allowed]

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we *always* have domain constraints.
- Powerful and natural query languages exist.
- Rules to translate ER to relational model