# Relational Model

Outline

- Intro to the Relational Model

- Intro to the SQL Query Language
  - SQL is the the standard language for creating, manipulating, and querying data in a relational DBMS.
  - Examine Definition Language (DDL) statements for creating relations

Relations

The relational model is very simple and elegant: a database is a collection of one or more relations, where each relation is a table with rows and columns.

A **relation** is a more concrete construction, of something we have seen before, the ER diagram.
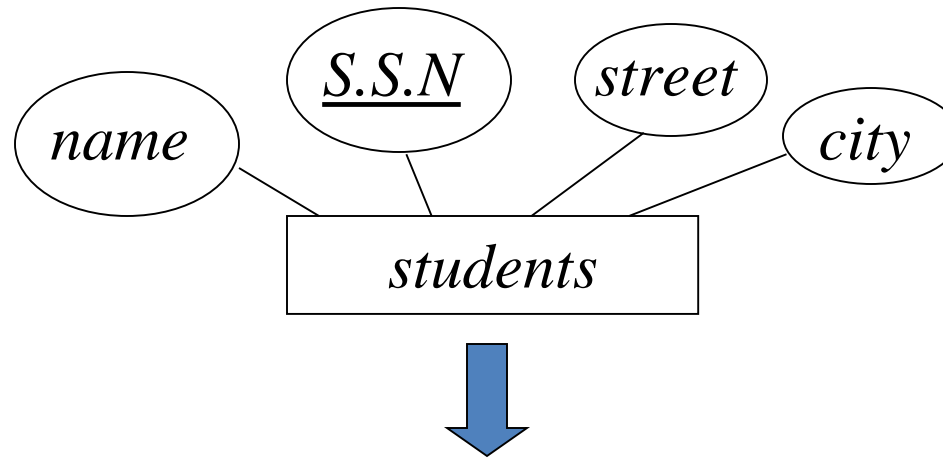
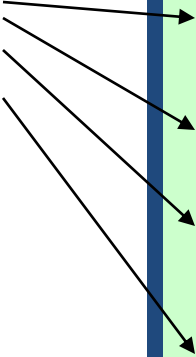| name	*S.S.N*	street	city
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside
Sue	5555	Main	Oceanside

The students relation

A **relation** consists of a **relational schema** and a **relational instance**.

A **relation schema** is essentially a list of column names with their data types. In this case…

students(*name* : string, *S.S.N* : string, *street* : string, *city* : string)

• A **relation instance** is made up of zero of more **tuples** (rows, records)

name	S.S.N	street	city
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside
Sue	5555	Main	Oceanside

A schema specifies a relation's name.

students(*name* : string, *S.S.N* : string, *street* : string, *city* : string)

A schema also specifies the name of each **field**, and its domain.

Fields are often referred to as columns, attributes, dimensions

A minor, but important point about relations, they are unordered.

name	S.S.N	street	city
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside
Sue	5555	Main	Oceanside

name	S.S.N	city	street
Lisa	1272	Riverside	Blaine
Bart	5592	Irvine	Apple
Sue	5555	Oceanside	Main
Lisa	7552	Riverside	11th

This is not a problem, since we refer to fields by name.

However sometimes, we refer to the fields by their column number, in which case the ordering becomes important. I will point this out when we get there.

Also, the tuples are unordered too!

Note that every tuple in our instance is unique. This is not a coincidence. The definition of relation demands it.

Later we will see how we can represent weak entities in relations.

name	S.S.N	street	city
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside
Sue	5592	Main	Oceanside

name	S.S.N	street	city
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside
Sue	5592	Main	Oceanside

The number of fields is called the **degree** (or **arity**, or dimensionality of the relation).

On the left we have a table of degree 4.

The number of tuples = **cardinality** of the relation

Of course, we don't count the row that has the labels!

To the right we have a table of cardinality 3.

name	S.S.N	street	city
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside

students(*name* : string, <u>*S.S.N*</u> : string, *street* : string, *city* : string)

- Note that relations have primary keys, just like ER diagrams.
- Remember that the primary key might not be one field, it may be a combination of two or more fields.
- However, we can have only 1 primary key per relation.

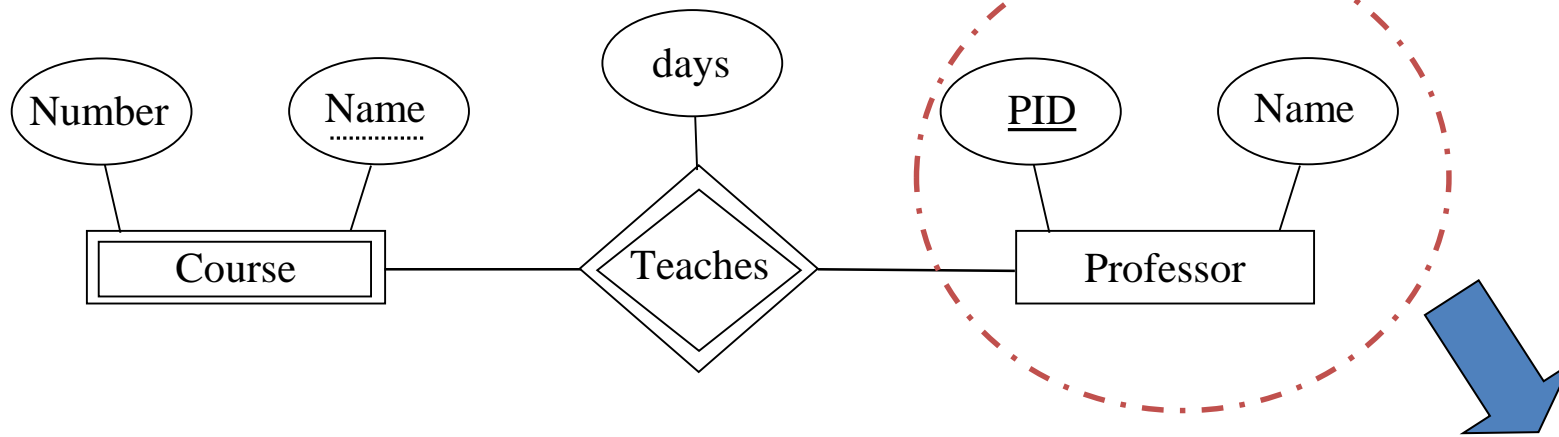| *name*	<u>S.S.N</u>	*street*	*city*
Lisa	1272	Blaine	Riverside
Bart	5592	Apple	Irvine
Lisa	7552	11th	Riverside
Sue	5555	Main	Oceanside

Translating ER diagrams into Relations

We need to figure out how to translate ER diagrams into relations.

There are only three cases to worry about.
• Strong entity sets
• Weak entity sets
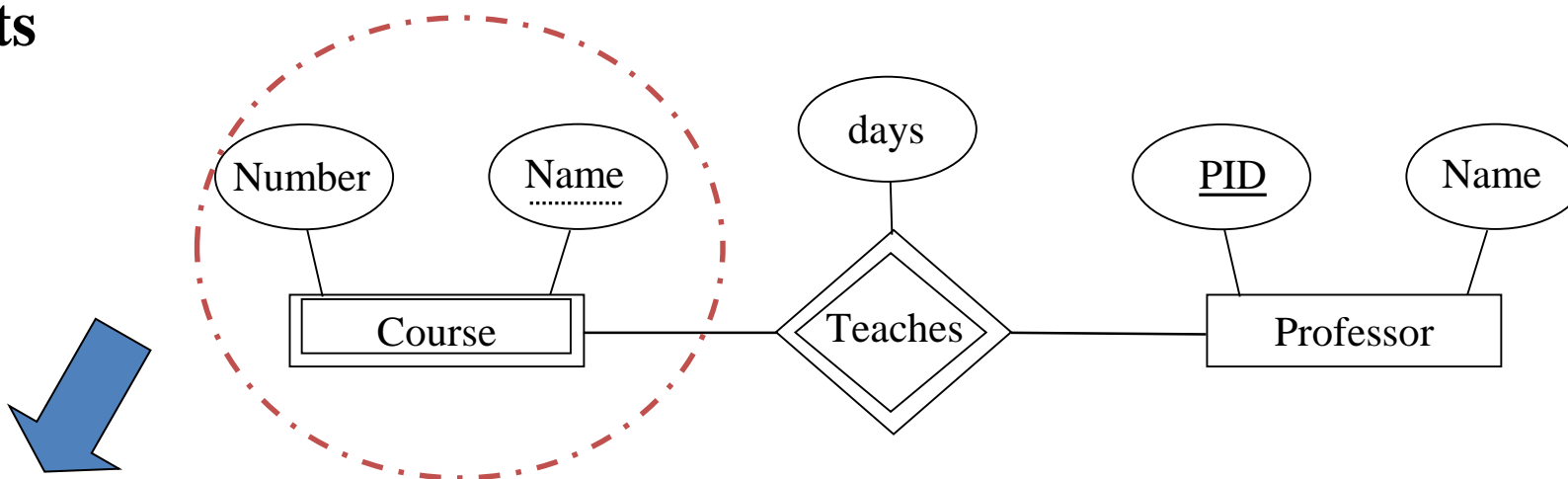• Relationship sets
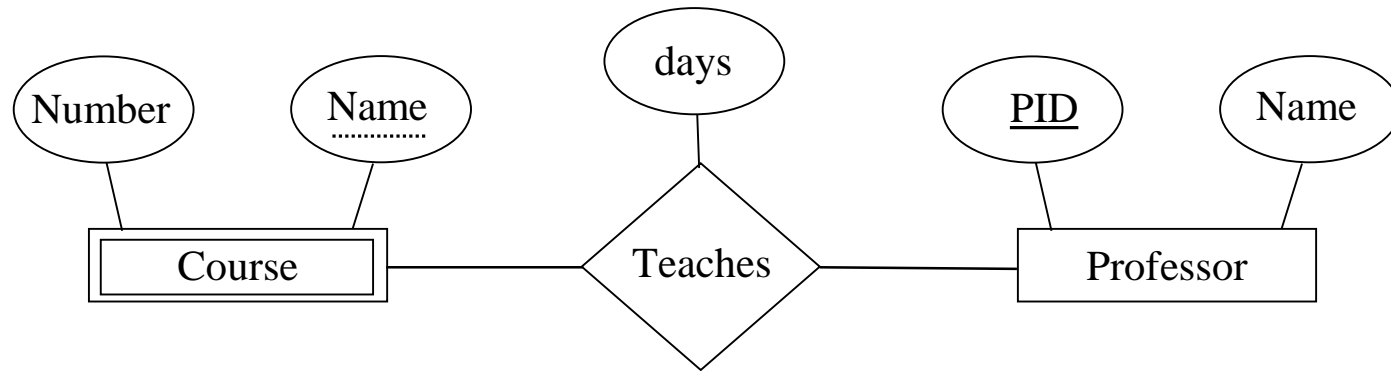
Strong entity sets

professor(*PID* : string, *name* : string)

This is trivial, the primary key of the ER diagram becomes the primary key of the relation. All other fields are copied in (in any order)

PID	name
1234	John
3421	Daisy
2342	Barbara
4531	Audrey

Weak entity sets

course(*PID* : string, *number* : string, *name* : string)

PID	number	name
1234	CS12	C++
3421	CS11	Java
2342	CS12	C++
4531	CS15	LISP

The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set. The "imported" key from the strong entity set is called the **foreign key**.

All other fields are copied in (in any order)

Relationship entity sets
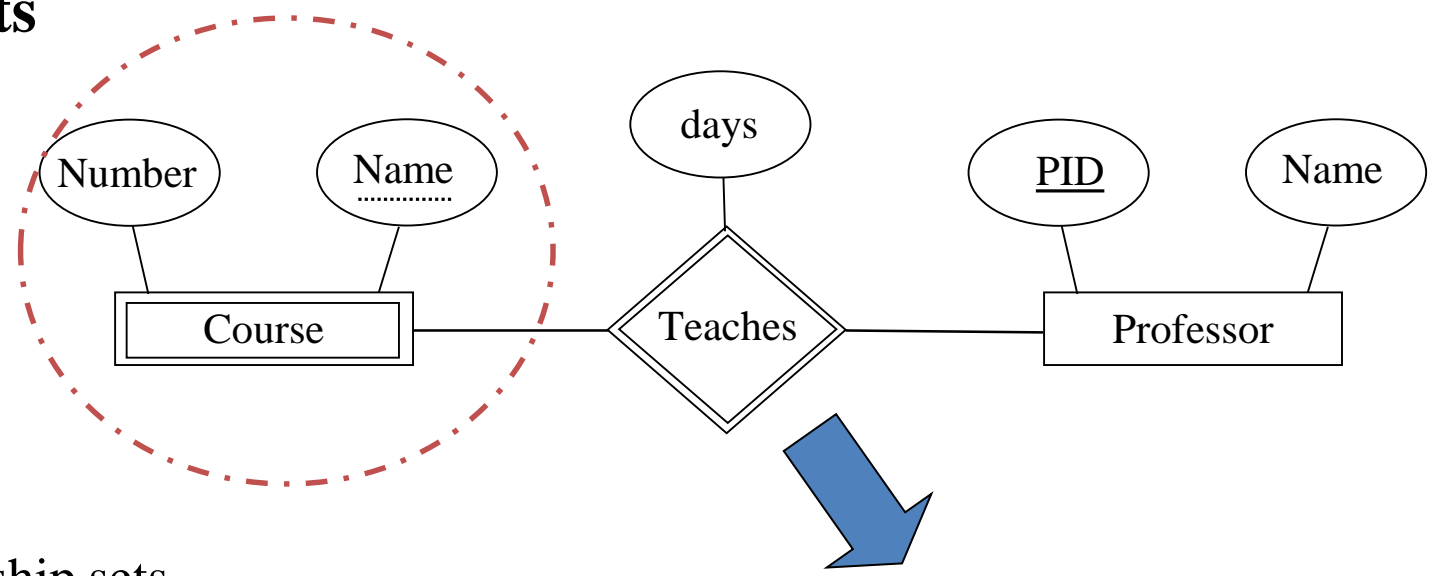
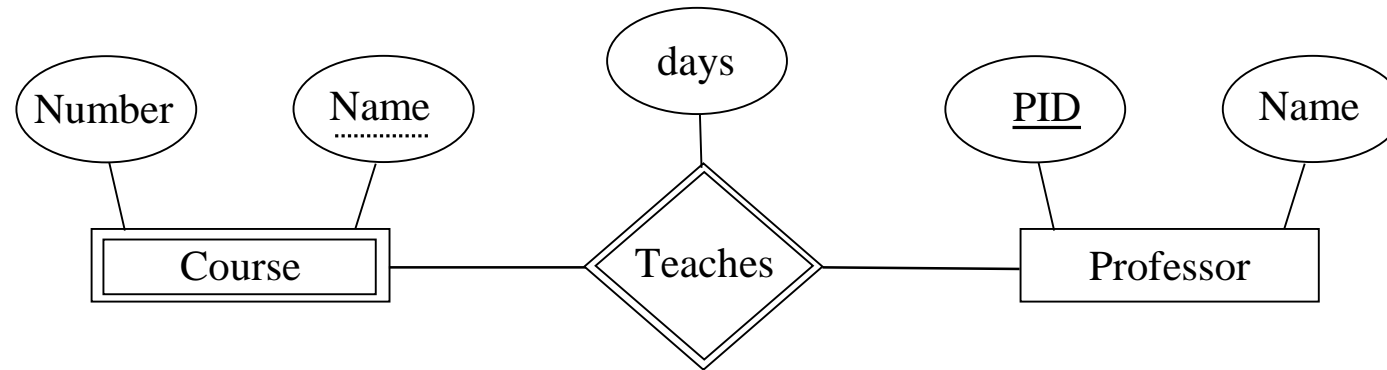For one-to-one relationship sets, the relation's primary key can be that of either entity set.

For many-to-many relationship sets, the union of the primary keys becomes the relation's primary key

For other cases, the the relation's primary key is taken from the strong entity set.

$teaches(\underline{PID} : \text{string}, days : \text{string})$

PID	days
1234	mwf
3421	wed
2342	tue
4531	sat

So, this ER Model…

… maps to this **database schema**

professor(<u>PID</u> : string, *name* : string)
course(<u>PID</u> : string, *number* : string, <u>name</u> : string)
teaches(<u>PID</u> : string, *days* : string)

We have seen how to create a database schema, how do we create an actual database on our computers?
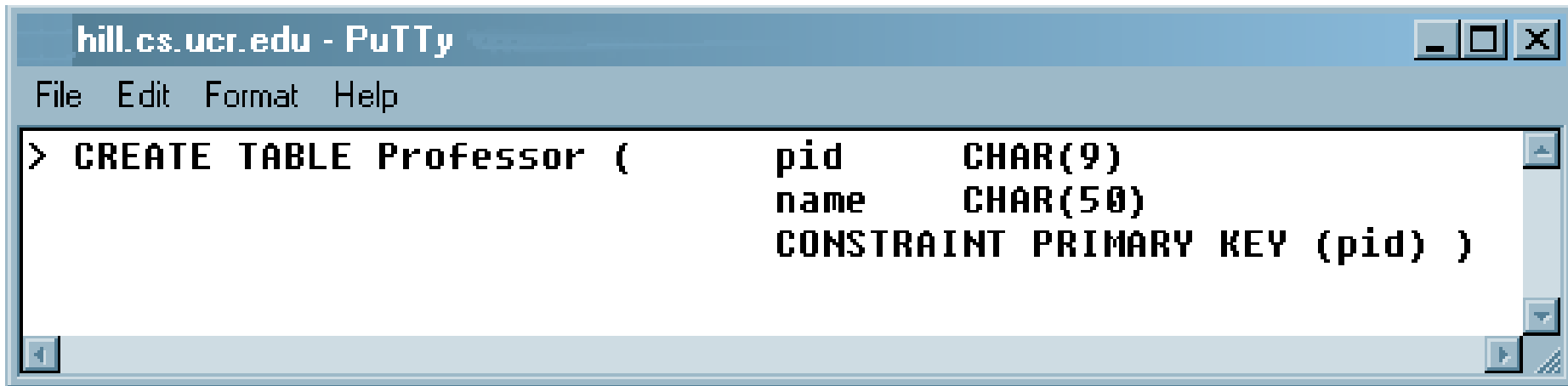
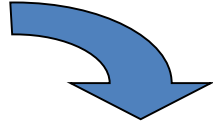*professor*(<u>*PID*</u> : string, *name* : string)
course(<u>*PID*</u> : string, *number* : string, <u>*name*</u> : string)
teaches(<u>*PID*</u> : string, *days* : string)

…how do we create an actual database on our computers?

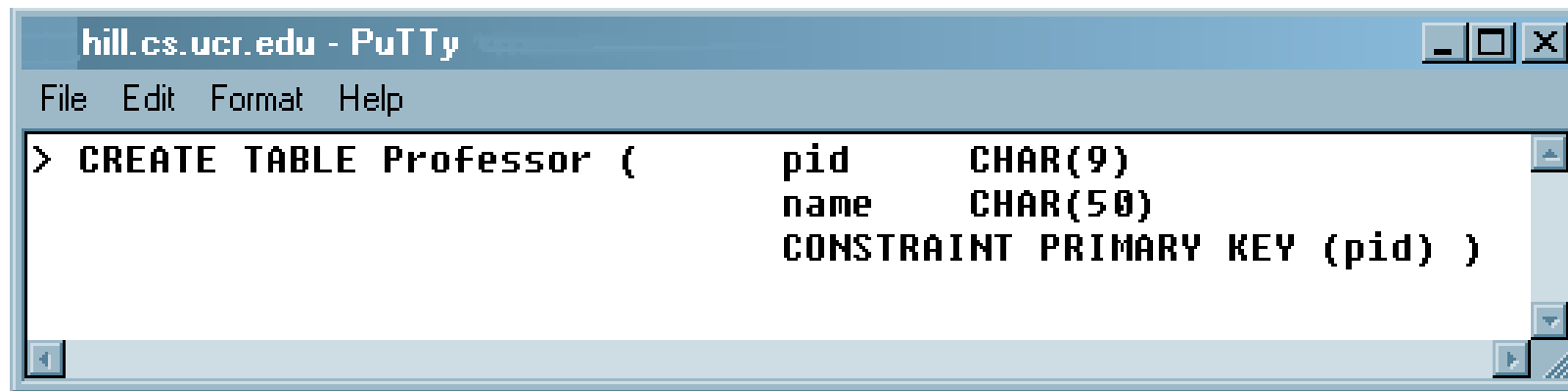We use SQL, a language that allows us to build, modify and query databases.

professor(*PID* : string, *name* : string)

```
hill.cs.ucr.edu - PuTTy                              _ □ x
File  Edit  Format  Help
> CREATE TABLE Professor (       pid      CHAR(9)
                                 name     CHAR(50)
                                 CONSTRAINT PRIMARY KEY (pid) )
```

SQL (Structured Query Language)

• SQL is a language that allows us to build, modify and query databases.

• SQL is an ANSI standard language. American National Standards Institute

• SQL is the "engine" behind Oracle, Sybase, Microsoft SQL Server, Informix, Access, Ingres, etc.

• Most of these systems have built GUIs on top of the command line interface, so you don't normally write statements directly in SQL (although you can).

```
hill.cs.ucr.edu - PuTTy

File  Edit  Format  Help

> CREATE TABLE Professor (        pid     CHAR(9)
                                  name    CHAR(50)
                                  CONSTRAINT PRIMARY KEY (pid) )
```

Creating Relations in SQL

- Creates the Students relation.

- Observe that the type (domain) of each field is specified and enforced by the DBMS whenever tuples are added or modified.

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

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


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

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)

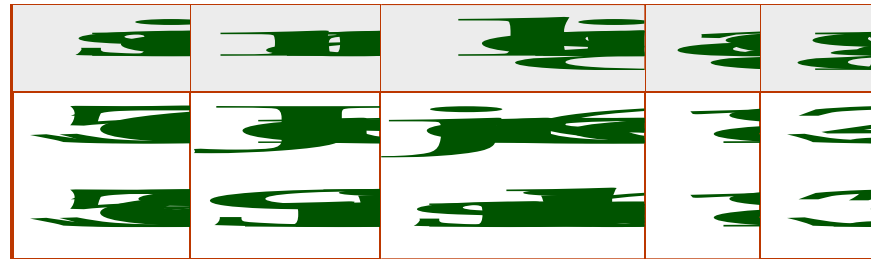- Can delete all tuples satisfying some condition (e.g., name = Smith):

  DELETE
  FROM Students
  WHERE name = 'Smith'

 Powerful variants of these commands are available; more later!

The SQL Query Language

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

SELECT *
FROM Students S
WHERE S.age=18

- To show just names and logins columns, replace the first line with:

SELECT S.name, S.login

Querying Multiple Relations

- What does this query compute?

SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A'

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

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

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.

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,

    2. no subset of the key is also a key.

    - 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., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

- "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."

- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

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 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:
  - 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 in SQL

- 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

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.

Referential Integrity in SQL

- SQL/92 and SQL:1999 support 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.

Logical DB Design: ER to Relational

- Entity sets to tables:
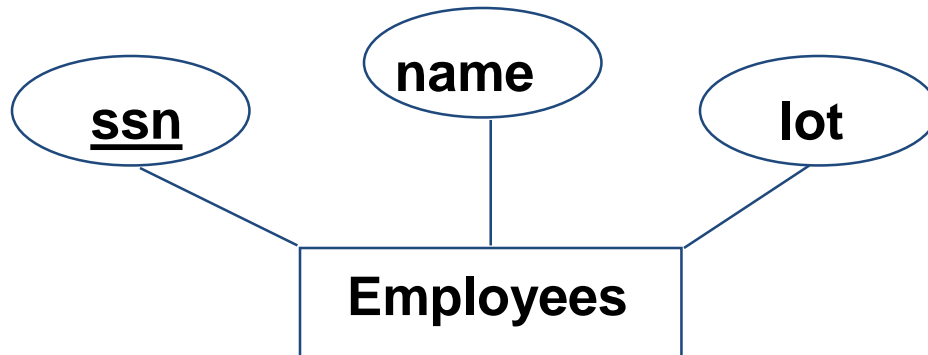

```
CREATE TABLE Employees
    (ssn CHAR(11),
    name CHAR(20),
    lot  INTEGER,
    PRIMARY KEY  (ssn))
```

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.
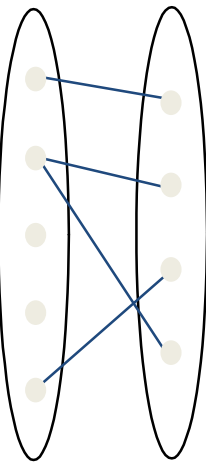
  - All descriptive attributes.

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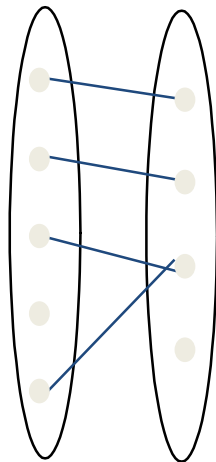   REFERENCES Departments
)

Review: Key Constraints

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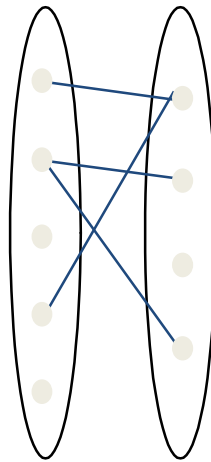

Translation to relational model?
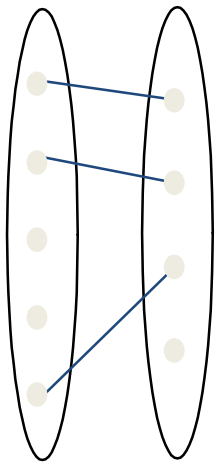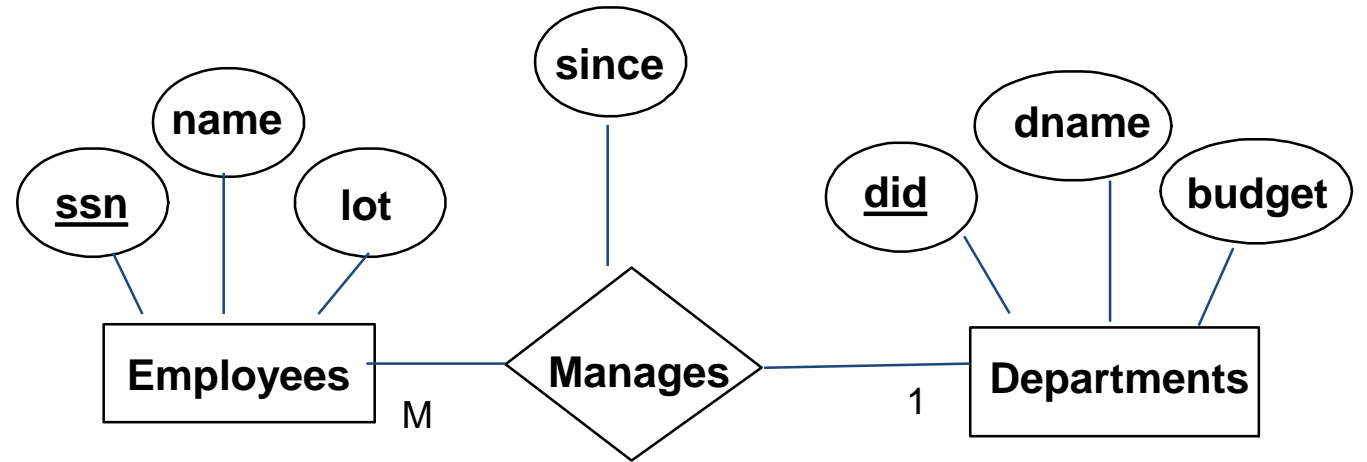
1-to-1   **1-to Many**   **Many-to-1**   **Many-to-Many**

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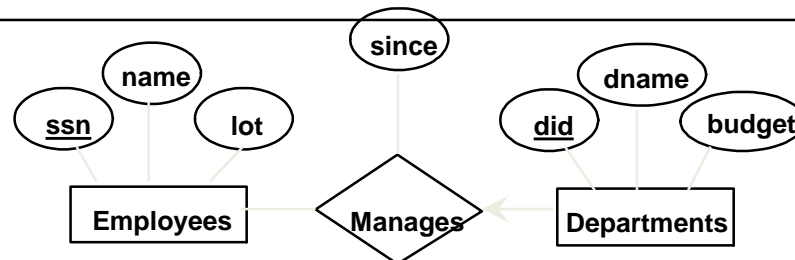
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)

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!)
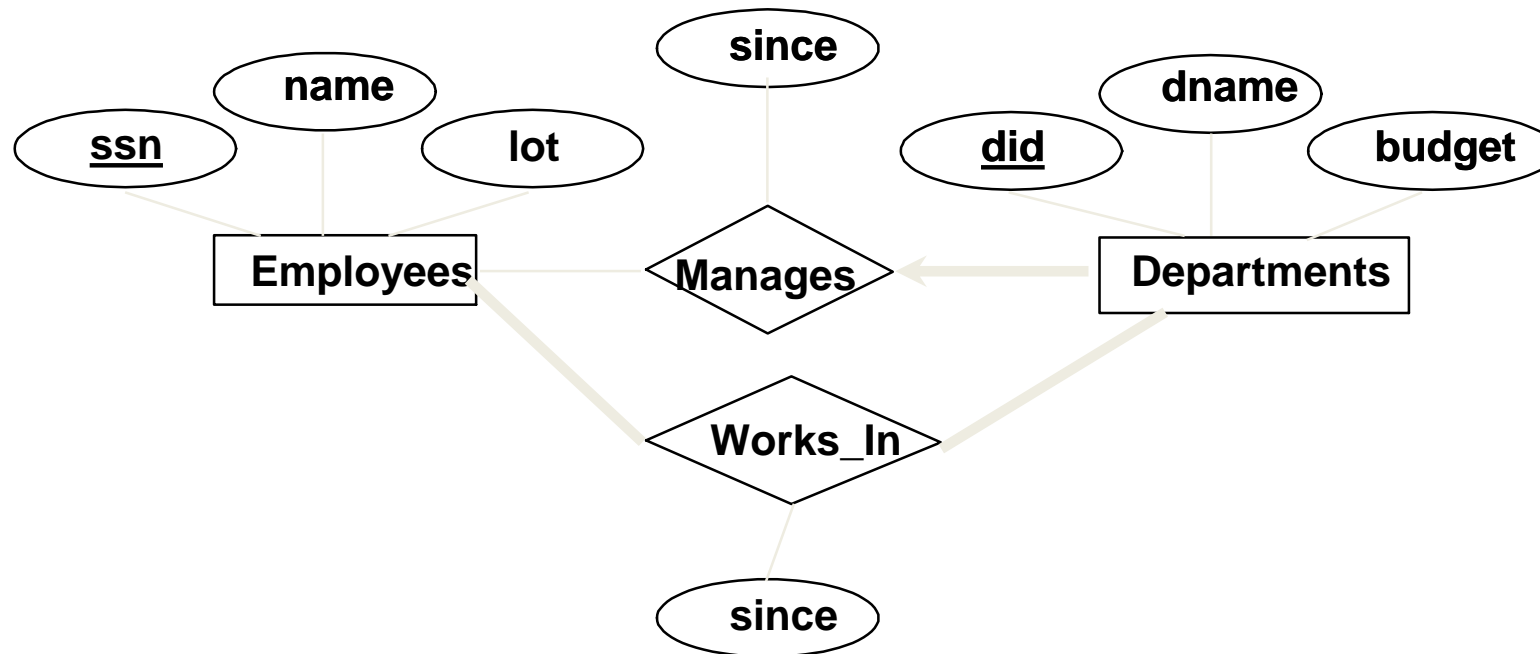
Participation Constraints in SQL

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

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.
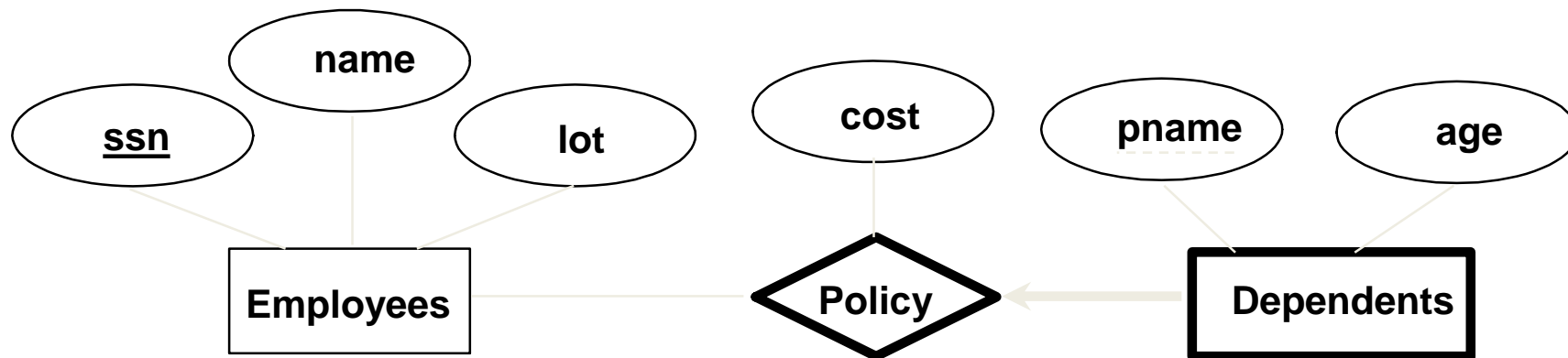
Translating Weak Entity Sets

- 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)
```
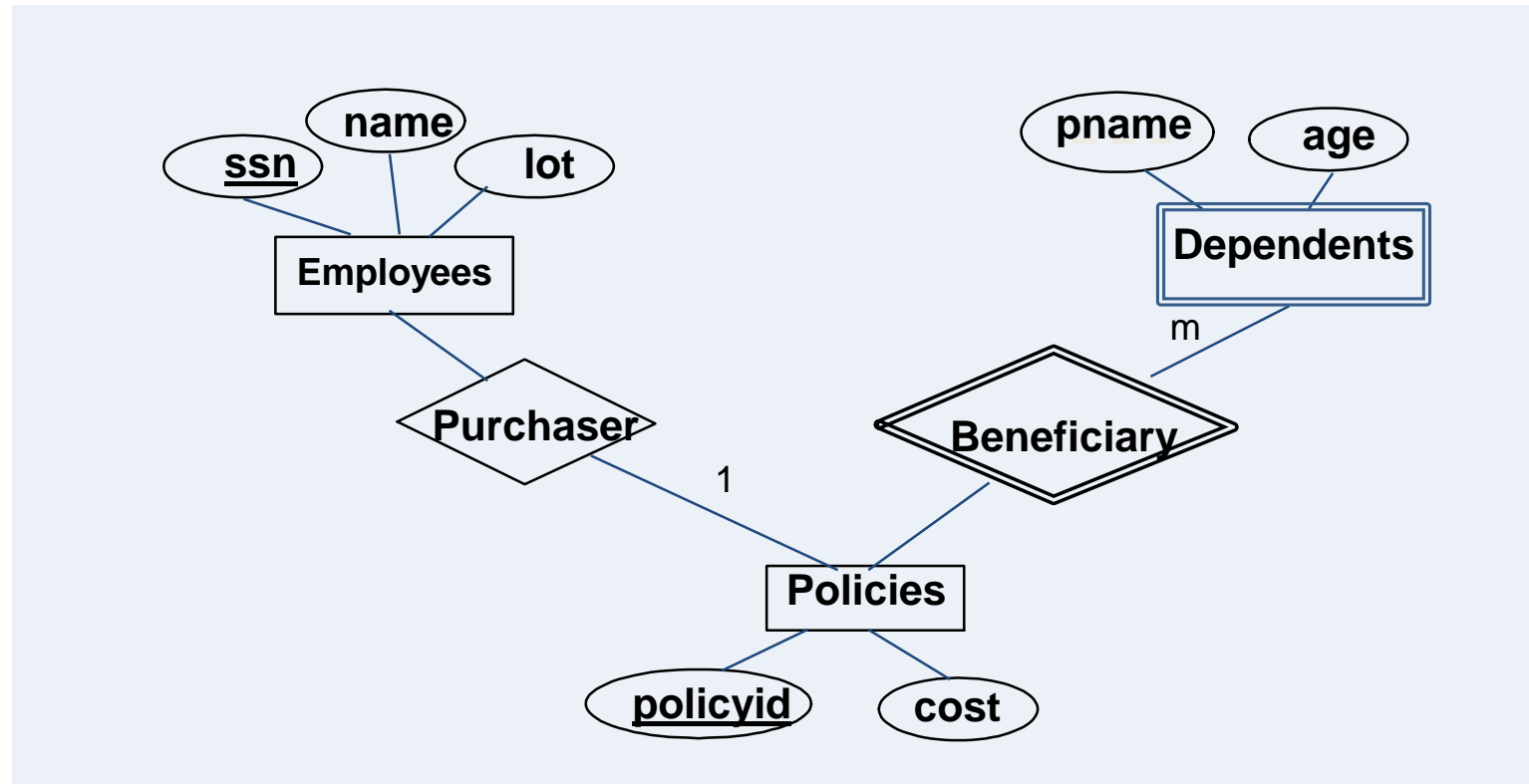
Review: Binary vs. Ternary Relationships

Bad design
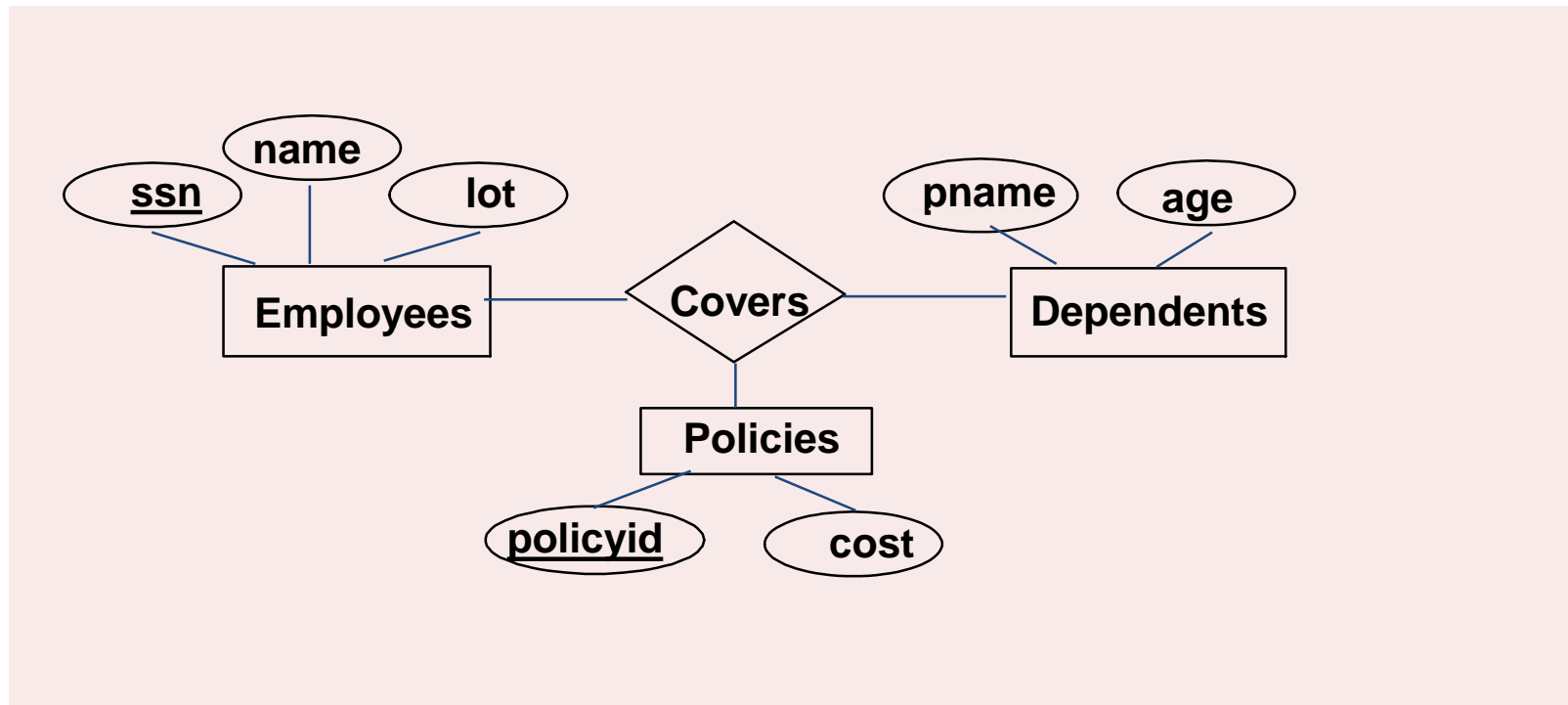
Review: Binary vs. Ternary Relationships

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)

Views

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

CREATE VIEW YoungStudents (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.

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 young students who are enrolled, but not the cid's of the courses they are enrolled in.

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