CPSC 304
Introduction to Database Systems

The Relational Model (Part 2 of 2)

Fall 2017

References:

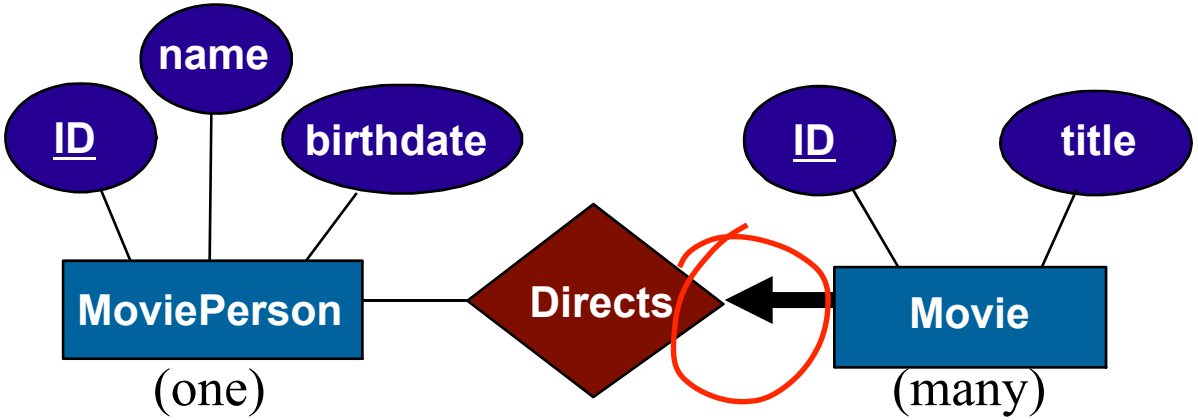
Database Management Systems, by Ramakrishnan and Gehrke (Chapter 3, Sections 3.1-3.5 of the Textbook)

Acknowledging previous work by CPSC 304 instructors over the years (alphabetically: Ed, George, Hassan, Hazra, Laks, Rachel, Raymond)

Learning Goals (same as for Part 1)

- Compare and contrast *logical* and *physical data independence*.
- Define the components (and synonyms) of the relational model: tables, rows, columns, keys, associations, etc.
- Create tables, including the attributes, keys, and field lengths, using SQL's Data Definition Language (DDL).
- Explain and differentiate the kinds of integrity constraints in a database.
- Explain the purpose of referential integrity.
- Enforce referential integrity in a database using SQL's Data Manipulation Language (DML). Determine which delete, insert, or update policy to use when coding rules/defaults for referential integrity. Analyze the impact that a poor choice has.
- Map ER diagrams to the relational model (i.e., DDL), including constraints, weak entity sets, etc.

Translating Participation Constraints



- Every movie must have exactly one director.
  - Every ID value in Movie must appear in a row of the Directs table (with a non-null MoviePerson ID value)
- How can we express that in SQL?

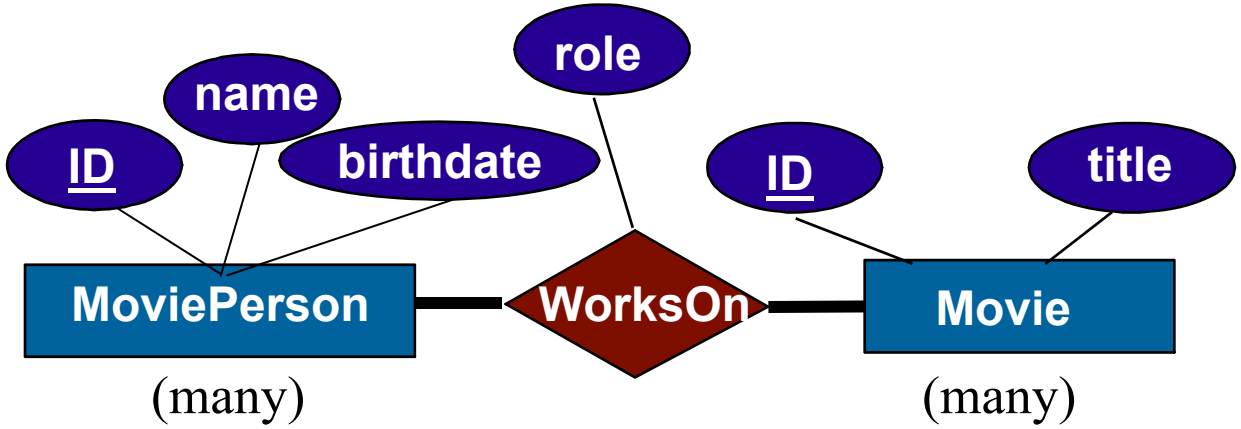
Participation Constraints in SQL

- Using Method 2 (adding the Manages relation in the Department table), we can capture participation constraints by:
  - Ensuring that each **mid** is associated with a **pid** that is not null
  - Not allowing the deletion of a director before he/she is replaced

```
CREATE TABLE Directs(                    No need Movie table
   mid          INTEGER,
   title        CHAR(20),
   pid          CHAR(11) NOT NULL,
   PRIMARY KEY  (mid),                   PK is not null by default
   FOREIGN KEY (pid) REFERENCES MoviePerson
                ON DELETE NO ACTION      rejected!
                ON UPDATE CASCADE)
```

- **Note: We cannot express this constraint if Method 1 is used for Directs.**

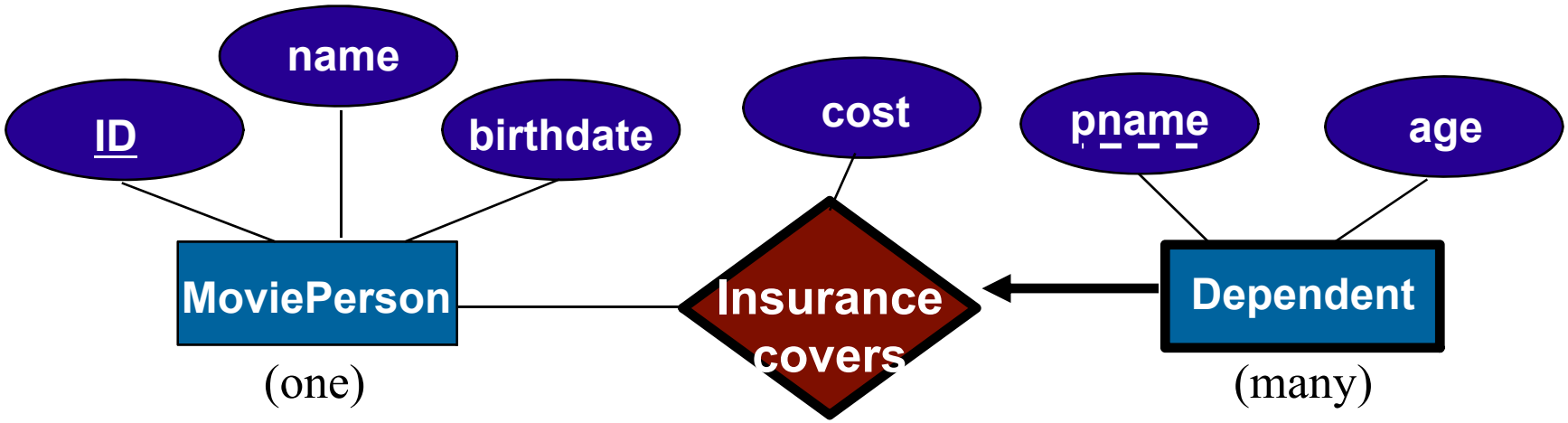
Participation Constraints in SQL (cont.)



Could be may, so we can't just add an attribute to the workson or movie table

- How can we express the requirement that: "Every Movieperson works on a movie, and every movie has some Movieperson working on it"?
- Neither the foreign key nor the not-null constraints in WorksOn can do that.
- We need *assertions* (later).

Translating Weak Entity Sets



- A **weak entity** is identified by considering the primary key of the *owner* (strong) entity.
  - The owner's entity set and the weak entity set participate in a one-to-many identifying relationship set.
  - The weak entity set has total participation.
- What is the best way to translate it?

Translating Weak Entity Sets (cont.)

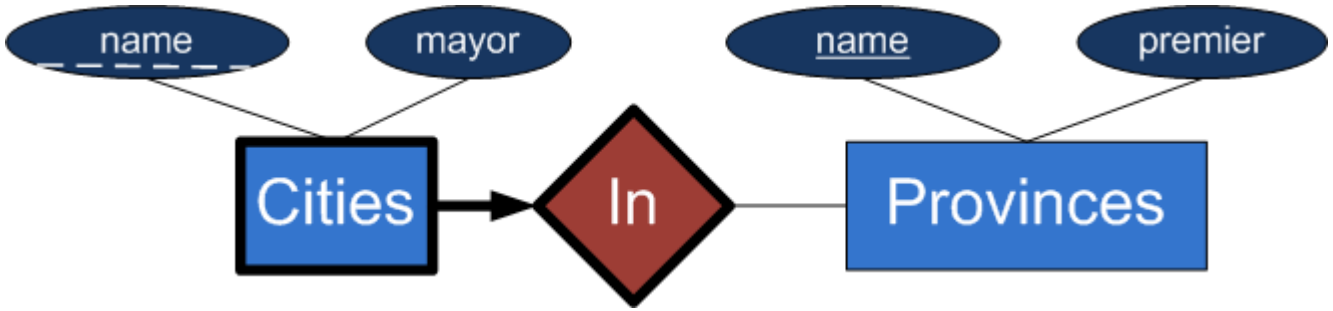
- The weak entity set and its identifying relationship set are translated into a single table.
  - The primary key consists of the owner's primary key and the weak entity's partial key.
  - When the owner entity is deleted, all of its owned weak entities must also be deleted.

```
CREATE TABLE Dep_Insurance (
    pname       CHAR(20),
    age         INTEGER,
    cost        REAL,
    ID          CHAR(11)
    PRIMARY KEY (ID, pname),
    FOREIGN KEY (ID) REFERENCES MoviePerson,
      ON DELETE CASCADE )
```

MoviePerson

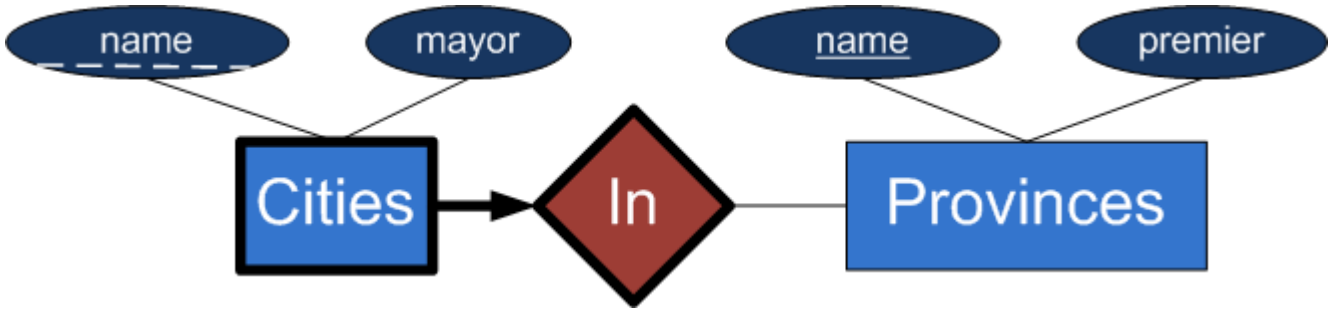
Composite key (2 or more attributes in PK)

Clicker Exercise



Convert this E/R diagram to relations, resolving the dual use of "name" in some reasonable way.

Clicker Exercise



Convert this E/R diagram to relations, resolving the dual use of "name" in some reasonable way. Which schema below is the most reasonable translation from ER to relations? (underline = PK, **bold** = FK)

A. Cities(name, mayor), Provinces(name, premier)

B. Cities(**cname**, **pname**, mayor), Provinces(pname, premier)

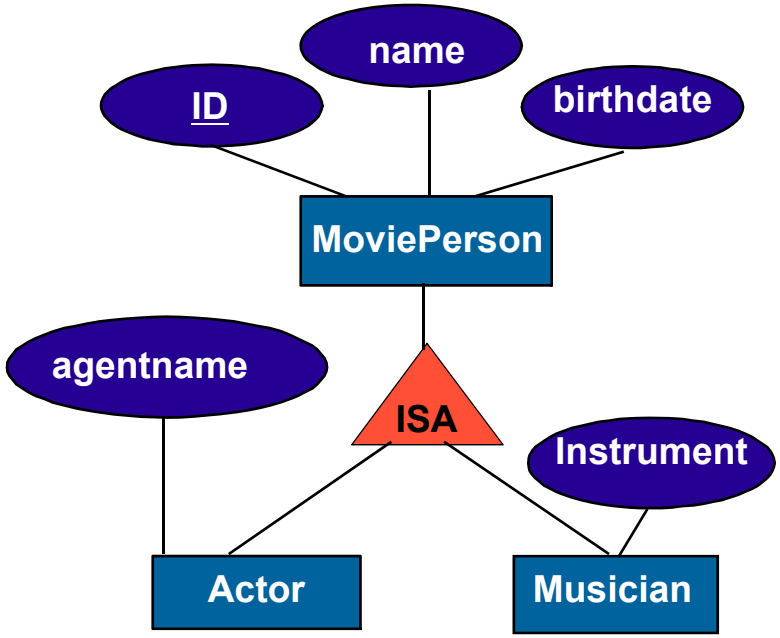
C. Cities(cname, **pname**, mayor), Provinces(pname, premier)

D. Cities(cname, **pname**, mayor), In(cname, pname), Provinces(name, premier)

E. None of the above

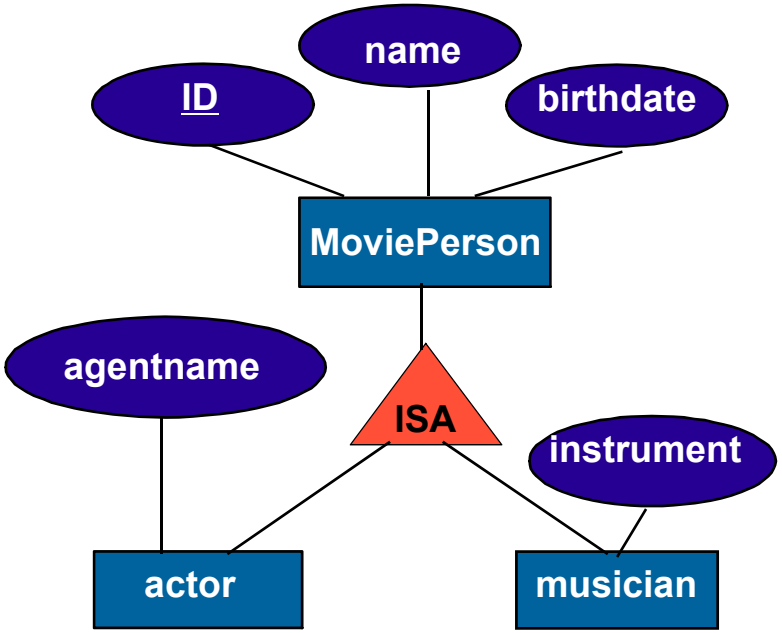
A Cities week ent., needs pname
B cname is not an FK
C Correct
D Don't need a separete In

Translating ISA Hierarchies to Relations



What is the best way to translate this into tables?

Unsatisfactory Attempt: "Safest", but with Lots of Duplication (not in Book)



One table per entity. Each has *all* attributes:

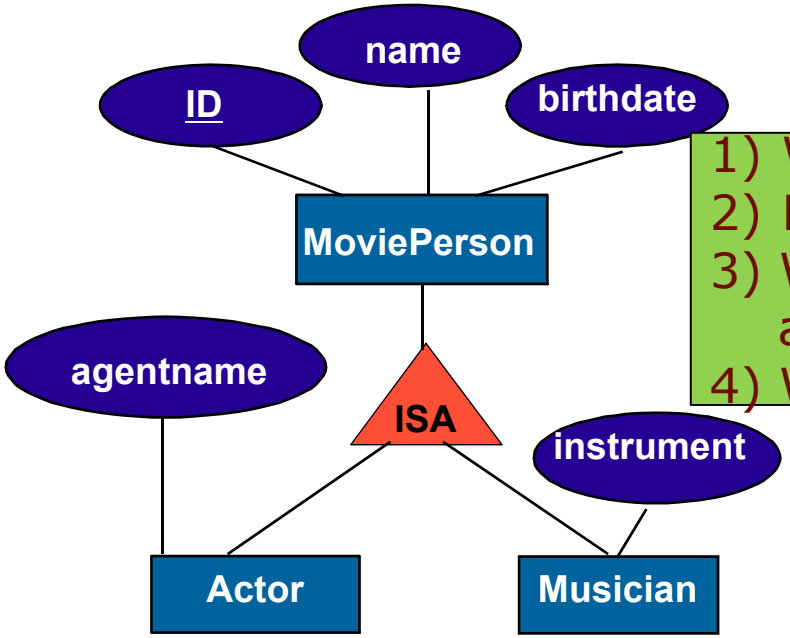
MoviePerson(ID, name, birthdate, agentname, instrument)

Actor(ID, name, birthdate, agentname, instrument)

Musician(ID, name, birthdate, agentname, instrument)

There's way too much duplication here! This will lead to future problems. So, don't use this approach.

Idea 1: Have Only One Table with *all* Attributes (not in Book)



1) What if I'm interested in just actors?
2) How do we identify actors or musicians?
3) What if there is a relationship that only actors can participate in?
4) What if I wanted to add a new subclass?

Great if the subclass hava few/no new attributes and relationships

MoviePerson(ID, name, birthdate, agentname, instrument)

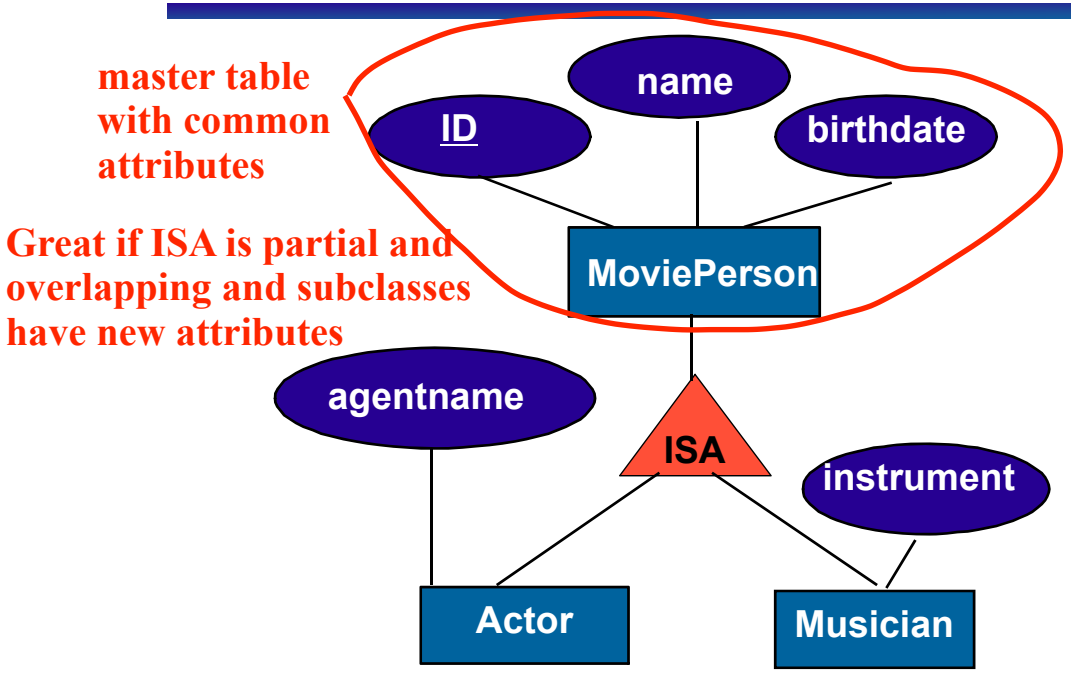
~~Actors(ID, name, birthdate, agentname, instrument)~~

~~Musicians(ID, name, birthdate, agentname, instrument)~~

add 'type' attribute
for example:
'type' = m represent musician
'type' = a represent actor

Lots of space needed for nulls!

Idea 2: Have 3 Tables, but Remove Excess Attributes from Each



- Superclass table contains all superclass attributes
- Subclass table contains primary key of superclass (as foreign key) and the subclass's attributes

MoviePerson(ID, name, birthdate, ~~agentName, instrument~~)

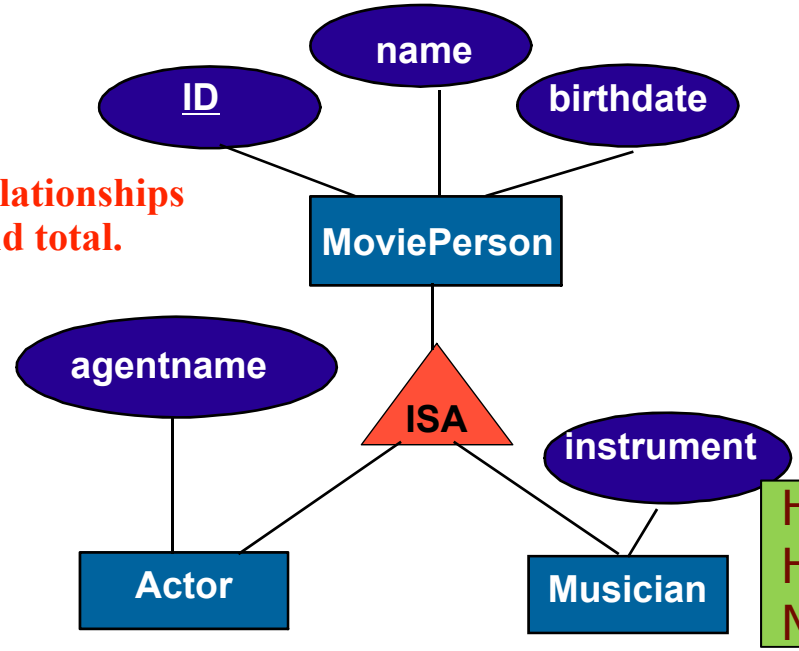
Actor(ID, ~~name, birthdate~~, agentname, ~~instrument~~)

Musician(ID, ~~name, birthdate, agentname~~, instrument)

This works well when concentrating on the superclass (e.g., when most queries are for the superclass).

We have to combine (join) two tables to get all the attributes for a subclass.

Idea 3: Have 2 Tables, but None for the Superclass



Great if ISA relationships are disjoint and total.

- No table for superclass
- One table per subclass
- Subclass tables have:
  - *All* superclass attributes
  - Subclass attributes

How should we store directors?
How do we store actors that are Musicians?

~~MoviePerson(ID, name, birthdate, agentname, instrument)~~

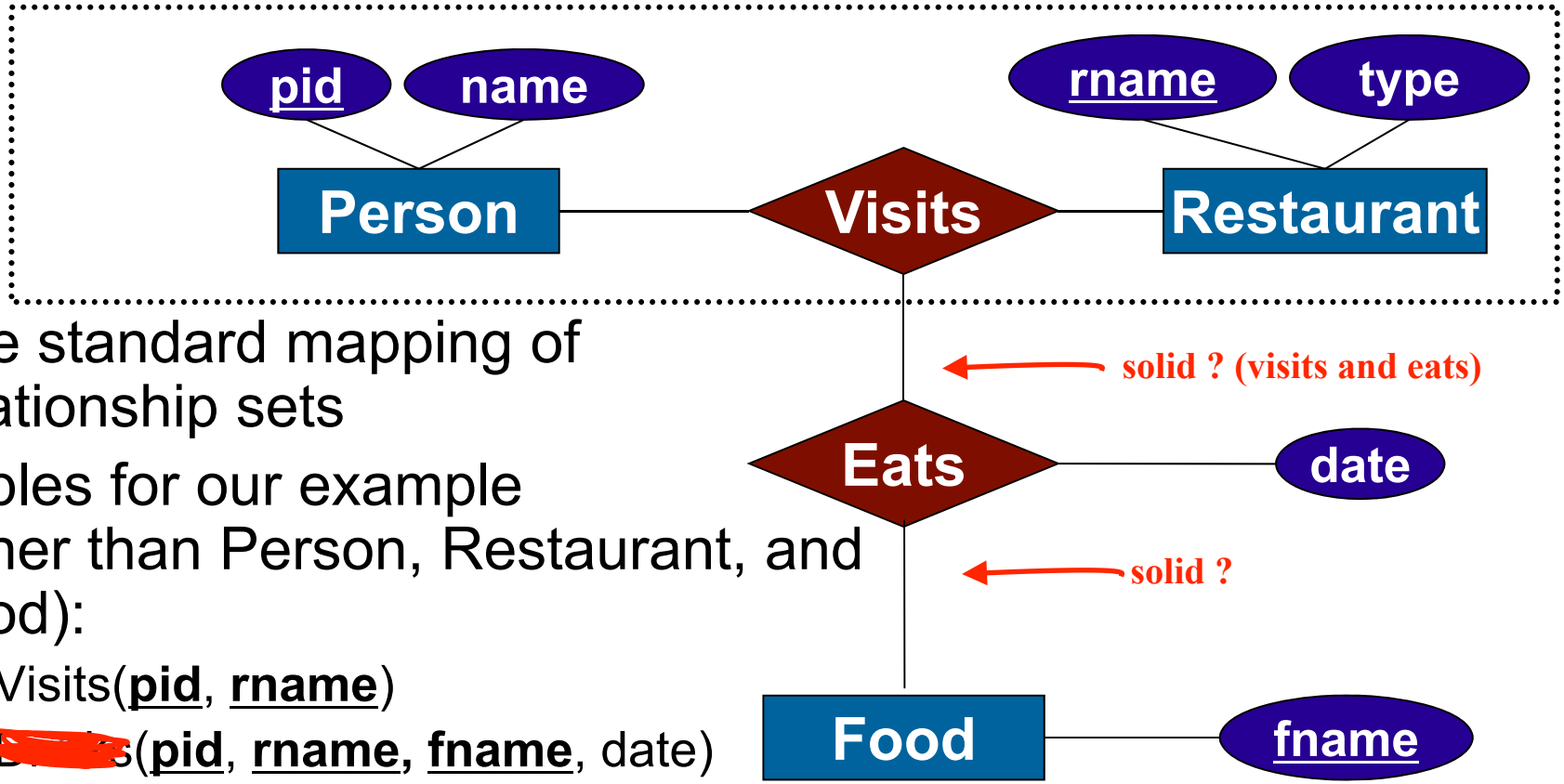
Actor(ID, name, birthdate, agentname, ~~instrument~~)

Musician(ID, name, birthdate, ~~agentname~~, ~~instrument~~)

- This works poorly with relationships involving the superclass.
- If the ISA relation is partial, it cannot be applied (due to loose entities).
- If the ISA relation is not disjoint, it duplicates information.

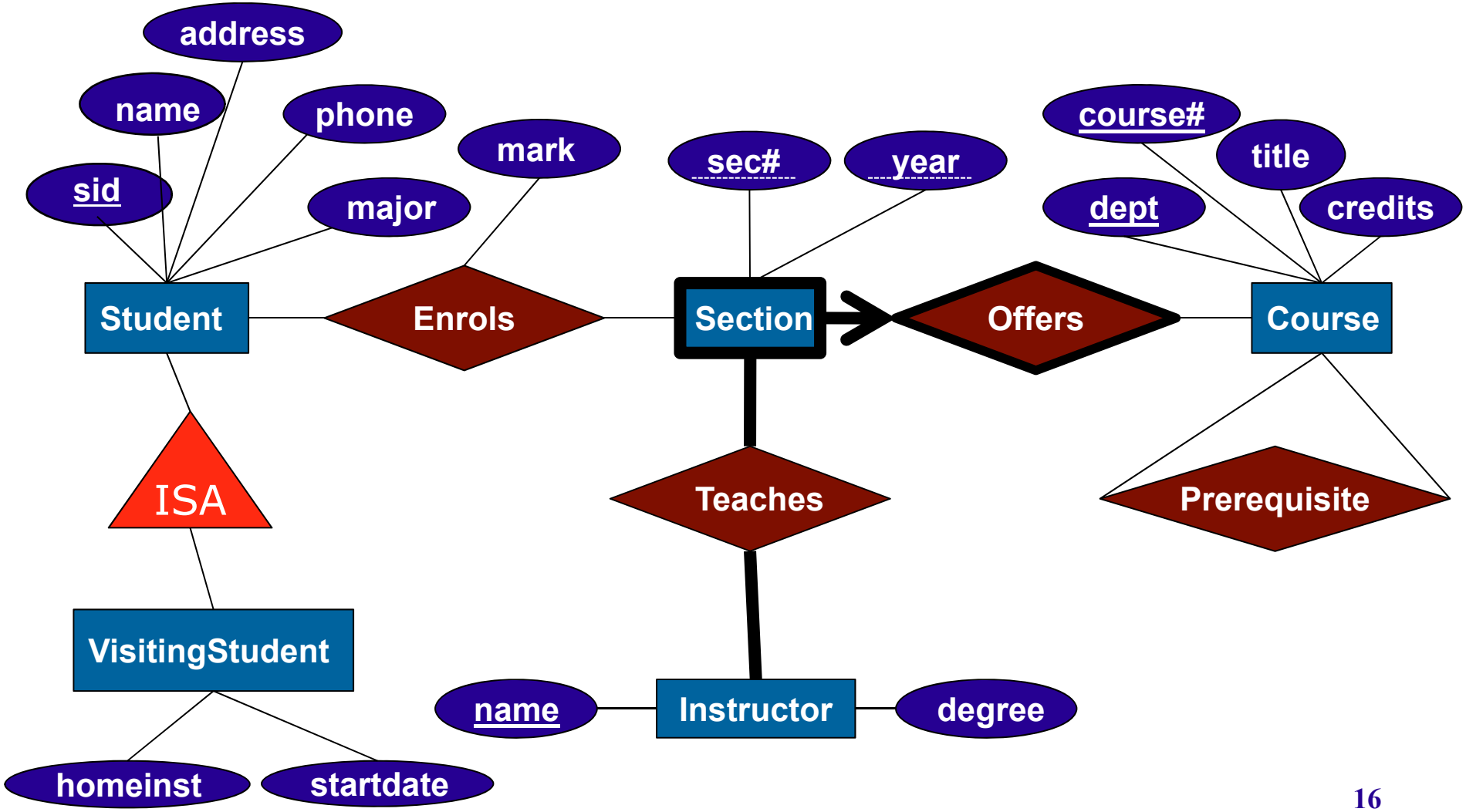
disjoint: e.g., union vs non-union

Translating Aggregations



- Use standard mapping of relationship sets
- Tables for our example (other than Person, Restaurant, and Food):
  - Visits(**pid**, **rname**)
  - Eats(**pid**, **rname**, **fname**, date)
- Special Case:
  - If Eats is total on Food, and Visits has no descriptive attributes, we could keep only the Food table (and discard Visits).

Consider the Following Diagram for a University. List the Tables, Keys, and Foreign Keys when Converting to Relational. Do Not Write SQL DDL.



Sample ER to Relational (Solution)

- Student (sid, name, address, phone, major)
- VisitingStudent (sid, homeinst, startdate)
  - Foreign key: (sid)
- Course (dept, course#, title, credits)
- Instructor(instname, degree)
- Offers(dept, course#, sec#, year)
  - Foreign keys: (dept, course#)

Sample ER to Relational (Solution) (cont.)

- Enrols(sid, dept, course#, sec#, year, mark)
  - Foreign keys: (sid), (dept, course#, sec#, year)
- Teaches(dept, course#, sec#, year, instname)
  - Foreign keys: (dept, course#, sec#, year), (instname)
  - Total participation constraint cannot be enforced, for now.
- Prerequisite(courseDept, course#, preDept, pre#)
  - Foreign keys: (courseDept, course#), (preDept, pre#)

Relational Model: Summary

- Tabular representation of data
- Simple and intuitive
- Currently the most widely used model of DBMS
- Integrity constraints can be specified based on application semantics. DBMS checks for violations.
  - Important ICs are primary and foreign keys.
  - Additional constraints can be defined with assertions (but are expensive to check).
- Powerful and natural query languages exist.
- There are rules to translate ER diagrams to the relational model.

Learning Goals Revisited

- Compare and contrast *logical* and *physical data independence*.
- Define the components (and synonyms) of the relational model: tables, rows, columns, keys, associations, etc.
- Create tables, including the attributes, keys, and field lengths, using SQL's Data Definition Language (DDL).
- Explain and differentiate the kinds of integrity constraints in a database.
- Explain the purpose of referential integrity.
- Enforce referential integrity in a database using SQL's Data Manipulation Language (DML). Determine which delete, insert, or update policy to use when coding rules/defaults for referential integrity. Analyze the impact that a poor choice has.
- Map ER diagrams to the relational model (i.e., DDL), including constraints, weak entity sets, etc.