CPSC 304
Introduction to Database Systems

The Relational Model

Textbook Reference
Database Management Systems: 3.1 - 3.5

Hassan Khosravi
Borrowing many slides from Rachel Pottinger

Databases – the continuing saga

- So far we've learned that databases are handy for many reasons
- Before we can use them, we must design them
- In our last very exciting episode, we showed how to use ER diagrams to design the *conceptual schema*
- But the conceptual schema can only get us so far; we need to store data!
- Now we'll learn to use a *logical schema* to actually store the data. We'll be using the *relational model*.

Learning Goals

- 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 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 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.

What do we want out of our logical schema representation?

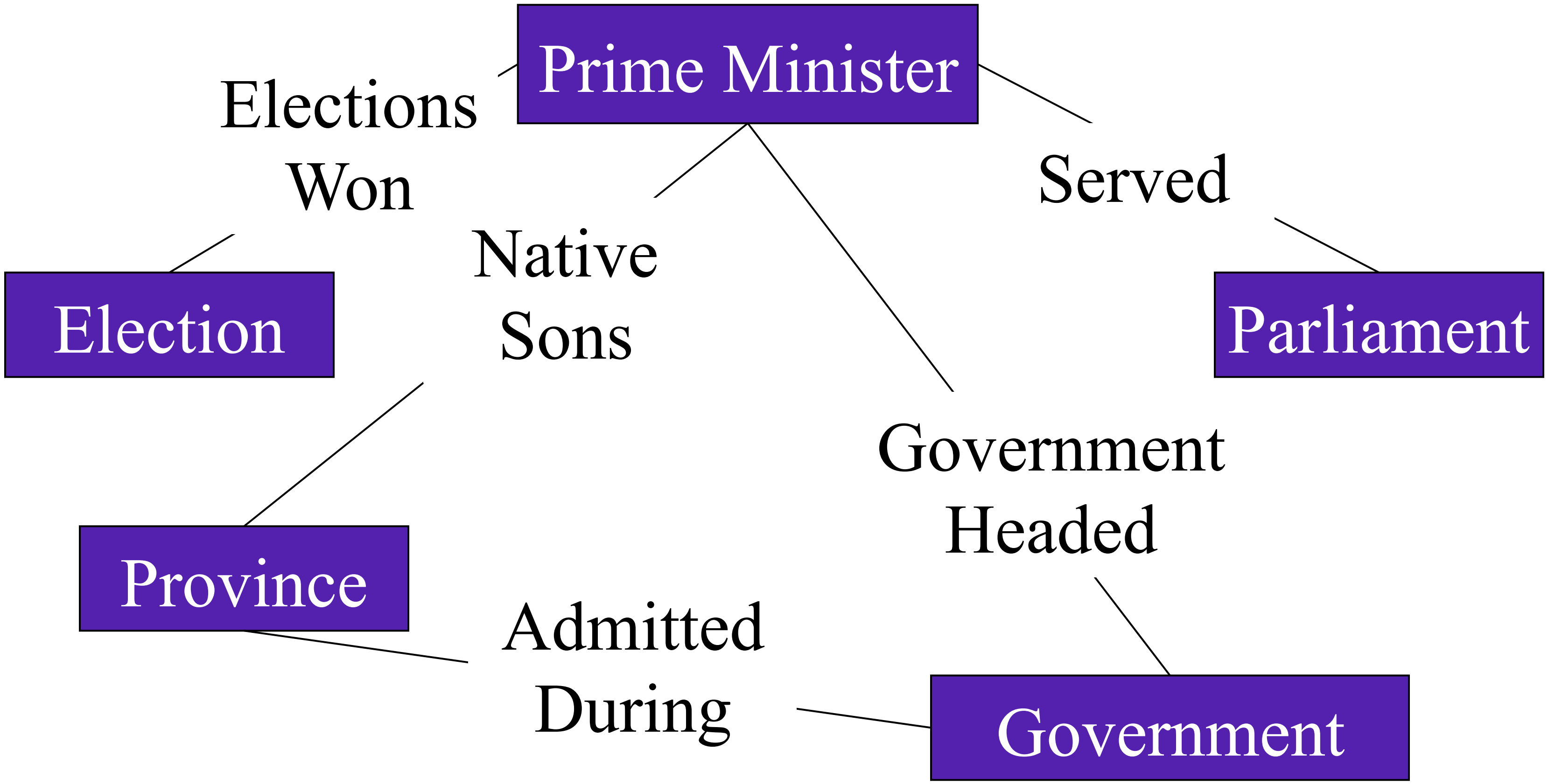
Ability to store data – w/o worrying about blocks on disk

- Ability to query data easily
- A representation that is easy to understand
- A representation that we can easily adapt from conceptual schema
- separate from application programming language

How did we get the relational model?

- Prior to the relational model, there were two main contenders
  - Network databases
  - Hierarchical databases
- Network databases had a complex data model
- Hierarchical databases integrated the application in the data model

Example Hierarchical Model



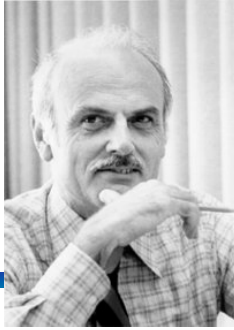
Looks similar to ER diagrams
but has fewer concepts... But let's see how you query it…

Example IMS (Hierarchical) query: Print the names of all the provinces admitted during a Liberal Government

```
DLITPLI:PROCEDURE (QUERY_PCB) OPTIONS (MAIN);

 DECLARE QUERY_PCB POINTER;
 /*Communication Buffer*/
 DECLARE 1 PCB BASED(QUERY_PCB),
  2 DATA_BASE_NAME CHAR(8),
  2 SEGMENT_LEVEL CHAR(2),
  2 STATUS_CODE CHAR(2),
  2 PROCESSING_OPTIONS CHAR(4),
  2 RESERVED_FOR_DLI FIXED BIRARY(31,0),
  2 SEGMENT_NAME_FEEDBACK CHAR(8)
  2 LENGTH_OF_KEY_FEEDBACK_AREA FIXED BINARY(31,0),
  2 NUMBER_OF_SENSITIVE_SEGMENTS FIXED BINARY(31,0),
  2 KEY_FEEDBACK_AREA CHAR(28);
 /* I/O Buffers*/
 DECLARE PRES_IO_AREA CHAR(65),
  1 PRESIDENT DEFINED PRES_IO_AREA,
  2 PRES_NUMBER CHAR(4),
  2 PRES_NAME CHAR(20),
  2 BIRTHDATE CHAR(8)
  2 DEATH_DATE CHAR(8),
  2 PARTY CHAR(10),
  2 SPOUSE CHAR(15);
 DECLARE SADMIT_IO_AREA CHAR(20),
  1 province_ADMITTED DEFINED SADMIT_IO_AREA,
  2 province_NAME CHAR(20);
 /* Segment Search Arguments */
 DECLARE 1 PRESIDENT_SSA STATIC UNALIGNED,
  2 SEGMENT_NAME CHAR(8) INIT('PRES '),
  2 LEFT_PARENTHESIS CHAR (1) INIT('('),
  2 FIELD_NAME CHAR(8) INIT ('PARTY '),
  2 CONDITIONAL_OPERATOR CHAR (2) INIT('='),
  2 SEARCH_VALUE CHAR(10) INIT ('Liberal '),
2 RIGHT_PARENTHESIS CHAR(1) INIT(')');
 DECLARE 1 province_ADMITTED_SSA STATIC UNALIGNED,
  2 SEGMENT_NAME CHAR(8) INIT('SADMIT ');
 /* Some necessary variables */
 DECLARE GU CHAR(4) INIT('GU '),
  GN CHAR(4) INIT('GN '),
  GNP CHAR(4) INIT('GNP '),
  FOUR FIXED BINARY (31) INIT (4),
  SUCCESSFUL CHAR(2) INIT(' '),
  RECORD_NOT_FOUND CHAR(2) INIT('GE');
 /*This procedure handles IMS error conditions */
 ERROR;PROCEDURE(ERROR_CODE);
  *
  *
  *
 END ERROR;
 /*Main Procedure */
 CALL PLITDLI(FOUR,GU,QUERY_PCB,PRES_IO_AREA,PRESIDENT_SSA);
 DO WHILE(PCB.STATUS_CODE=SUCCESSFUL);
  CALL PLITDLI(FOUR,GNP,QUERY_PCB,SADMIT_IO_AREA,province_ADMITTED_SSA);
  DO WHILE(PCB.STATUS_CODE=SUCCESSFUL);
   PUT EDIT(province_NAME)(A);
  CALL PLITDLI(FOUR,GNP,QUERY_PCB,SADMIT_IO_AREA,province_ADMITTED_SSA);
   END;
  IF PCB.STATUS_CODE NOT = RECORD_NOT_FOUND
   THEN DO;
    CALL ERROR(PCB.STATUS_CODE);
    RETURN;
    END;
   CALL PLITDLI(FOUR,GN,QUERY_PCB,PRES_IO_AREA,PRESDIENT_SSA);
  END;
  IF PCB.STATUS_CODE NOT = RECORD_NOT_FOUND
   THEN DO;
    CALL ERROR(PCB.STATUS_CODE);
    RETURN;
    END;
END DLITPLI;
```

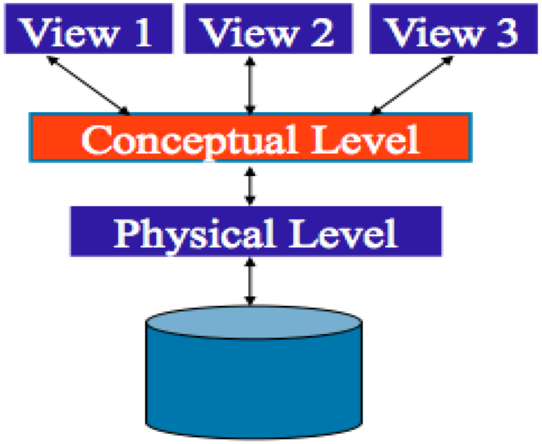
Relational model to the rescue!



- Introduced by Edgar Codd (IBM) in 1970
- Most widely used model today.
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- Competitor: object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: *object-relational model*
    - Informix Universal Server, UniSQL, O2, Oracle, DB2
- Recent competitors(triggered by the needs of Web):
  - XML data model
  - NoSQL

Key points of the relational model

- Exceedingly simple to understand – main abstraction is represented as a table
- **Physical Data Independence** –ability to modify physical schema w/o changing logical schema
- **Logical Data Independence** – done with views
  - Ability to change the conceptual schema without changing applications



Structure of Relational Databases

- ***Relational database:*** a set of ***relations***
- ***Relation:*** made up of 2 parts:
  - ***Schema*** : specifies name of relation, plus name and **domain** (type) of each **attribute**.
    - e.g., Student (*sid*: string, *name*: string, *address*: string, *phone*: string, *major*: string).
  - ***Instance*** : a ***table***, with rows and columns.
    #Rows = cardinality
    #Columns = arity / degree
- ***Relational Database Schema:*** collection of schemas in the database
- ***Database Instance:*** a collection of instances of its relations

Example of a Relation Instance

attribute, column name

relation name

Student

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC
94001150	S. Wang	null	null	null

tuple, row, record

domain value

- degree/arity = 5; Cardinality = 4,
- Order of rows isn't important
- Order of attributes isn't important (except in some query languages)

Formal Structure

- Formally, a relation r is a set $(a_1, a_2, \ldots, a_n)$ where a_i is in D_i , the domain (set of allowed values) of the i-th attribute.

- Attribute values are atomic, i.e., integers, floats, strings

- A domain contains a special value ***null*** indicating that the value is not known.

- If $A_1, \ldots, A_n$ are attributes with domains $D_1, \ldots D_n$, then
  $(A_1{:}D_1, \ldots, A_n{:}D_n)$
  is a ***relation schema*** that defines a relation type – sometimes we leave off the domains
- Student (*sid*: string, *name*: string, *address*: string, *phone*: string, *major*: string).

Example of a formal definition

Student

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC
94001150	S. Wang	null	null	null

Student(sid: integer, name: string, address: string, phone: string, major: string)

Or, without the domains:

Student (sid, name, address, phone, major)

Clicker Question

- Here is a table representing a relation named R. Identify the attributes, schema, and tuples of R Which of the following is NOT a true statement about R?

A	B	C
0	1	2
3	4	5
6	7	8
9	10	11

A. R has four tuples.
B. B is an attribute of R.
C. (6,7,8) is a tuple of R.
D. The schema of R is R(A,B,C).
E. None of the above

E. All are true

Relational Query Languages

- A major strength of the relational model: simple, powerful *querying* of data.
- Queries can be written intuitively; DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows optimizer to extensively re-order operations, while ensuring that the answer does not change.

Raymond Boyce

Don Chamberlain

The SQL Query Language

- Developed by IBM (System R) in the 1970s
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

A peek at the SQL Query Language (1/2)

Students

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC

- Find the id's, names and phones of all CPSC students:

```
SELECT  sid, name, phone
FROM    Students
WHERE   major="CPSC"
```

sid	name	phone
99111120	G. Jones	889-4444
94001020	A. Smith	222-2222

- To select whole rows , replace "SELECT sid, name, phone " with "SELECT * "

A peek at the SQL Query Language (2/2): Querying Multiple Tables

Student

sid	name	address	phone	major
99111120	G. Jones	…	…	CPSC
…	…	….	…	…

Grade

sid	dept	course#	mark
99111120	CPSC	122	80
…	…	….	…

- To select id and names of the students who have taken some CPSC course, we write:

```
SELECT sid, name
FROM Student, Grade
WHERE Student.sid = Grade.sid AND dept = ‘CPSC’
```

Compare with Hierarchical Example

Simple, eh?

- We'll see more about how to query (data manipulation language) in Chapter 5.
- But you can't query without having a place to store your data, so back to how to create relations (data definition language)

Creating Relations in SQL/DDL

- The statement on the right creates the Student relation
  - the type (domain) of each attribute is specified and enforced when tuples are added or modified

```
CREATE TABLE Student
      (sid        INTEGER,
       name       CHAR(20),
       address    CHAR(30),
       phone      CHAR(13),
       major      CHAR(4)) x
```

- The statement on right creates Grade information about courses that a student takes

```
CREATE TABLE Grade
(sid        INTEGER,
 dept       CHAR(4),
 course#    CHAR(3),
 mark       INTEGER)
```

Destroying and Altering Relations

```
DROP TABLE Student
```

- Destroys the relation Student. Schema information *and* tuples are deleted.

```
ALTER TABLE Student
        ADD COLUMN gpa REAL;
```

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

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT
INTO     Student (sid, name, address, phone, major)
VALUES  ('52033688', 'G. Chan', '1235 W. 33, Van',
          '882-4444', 'PHYS')
```

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

```
DELETE
FROM    Student
WHERE   name = 'Smith'
```

☛ *Powerful variants of these commands exist; 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
  - Avoids data entry errors, too!
- The types of IC's depend on the data model.
  - What did we have for ER diagrams?
  - Next up: constraints for relational databases

Keys Constraints (for Relations)

- Similar to those for entity sets in the ER model
- A set $S=\{S_1, S_2, \ldots, S_m\}$ of attributes in an *n*-ary relation ($1 \leq m \leq n$) is a ***key*** (or ***candidate key***) for a relation if :
  1. No distinct tuples can have the same values in all key attributes, and
  2. No subset of S is itself a key (according to (1)).

  (If such a subset exists, then S is a *superkey* and not a key.)
- One of the possible keys is chosen (by the DBA) to be the ***primary key*** (PK).
- e.g.
  - {*sid, name*} is a superkey
  - ***sid*** is the primary key for Students

```
CREATE TABLE Student
  (sid      INTEGER  PRIMARY KEY,
   name     CHAR(20),
   address  CHAR(30),
   phone    CHAR(13),
   major    CHAR(4))
```

Keys Constraints in SQL

- A **PRIMARY KEY** constraint specifies a table's primary key
  - values for primary key must be unique
  - a primary key attribute cannot be *null*
- Other keys are specified using the **UNIQUE** constraint
  - values for a group of attributes must be unique (if they are not null)
  - these attributes can be *null*
- Key constraints are checked when
  - new values are inserted
  - values are modified

Keys Constraints in SQL (cont')

- (Ex.1- Normal) "For a given student and course, there is a single grade."

```
CREATE TABLE Grade
   (sid        INTEGER,
    dept       CHAR(4),
    course#    CHAR(3),
    mark       INTEGER,
    PRIMARY KEY  (sid,dept,course#) )
```

vs.

- (Ex.2 - Silly) "Students can take a course once, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Grade2
   (sid        INTEGER,
    dept       CHAR(4),
    course#    CHAR(3),
    mark       CHAR(2),
    PRIMARY KEY (sid,dept,course#),
    UNIQUE (dept,course#,mark) )
```

For single attribute keys, can also be declared on the same line as the attribute

Clicker question

- Consider the table definition: CREATE TABLE Emps (
  (1) id INT,
  Numbers denote lines only → (2) sin INT,
  (3) name CHAR(20),
  (4) managerID INT);
- Which of the following is **not** a legal addition?

A. Add UNIQUE just before the commas on lines (2) and (3) and add PRIMARY KEY just before the comma on line (1).

B. Add PRIMARY KEY just before the commas on lines (1) and (2).

C. Add UNIQUE just before the comma on line (1), and add PRIMARY KEY just before the comma on line (2).

D. All are legal

E. None are legal

Clicker question

- Consider the table definition: CREATE TABLE Emps (

  Numbers denote lines only → (1) id INT,
  (2) sin INT,
  (3) name CHAR(20),
  (4) managerID INT);

- Which of the following is **not** a legal addition?

A. Add UNIQUE just before the commas on lines (2) and (3) and add PRIMARY KEY just before the comma on line (1).

B. Add PRIMARY KEY just before the commas on lines (1) and (2). Not legal - (can't have 2 primary keys)

C. Add UNIQUE just before the comma on line (1), and add PRIMARY KEY just before the comma on line (2).

D. All are legal

E. None are legal

Foreign Keys Constraints

- ***Foreign key*** : Set of attributes in one relation used to 'reference' a tuple in another relation.
  - Must correspond to the primary key of the other relation.
  - Like a 'logical pointer'.
- E.g.:
  Grade(*sid*, *dept*, *course#*, *grade*)
  - *sid* is a foreign key referring to Student:
  - (*dept*, *course#*) is a foreign key referring to Course
- *Referential integrity*: All foreign keys reference existing entities.
  - i.e. there are no dangling references
  - all foreign key constraints are enforced
  - Example of a data model without Referential Integrity? HTML

Foreign Keys in SQL

- Only students listed in the Student relation should be allowed to have grades for courses that are listed in the Course relation.

```
CREATE TABLE Grade
   (sid INTEGER,  dept  CHAR(4), course# CHAR(3), mark INTEGER,
    PRIMARY KEY  (sid,dept,course#),
    FOREIGN KEY (sid) REFERENCES Student(sid),
    FOREIGN KEY (dept, course#) REFERENCES Course(dept, cnum))
```

Primary key in Student

Primary key in Course

Grade

sid	dept	course#	mark
53666	CPSC	101	80
53666	RELG	100	45
53650	MATH	200	null
53666	HIST	201	60

Student

sid	name	address	Phone	major
53666	G. Jones	….	…	…
53688	J. Smith	….	…	…
53650	G. Smith	….	…	…

Self Referencing Relations

Goal: have managerID be foreign key reference for same table Emps.

id	sin	name	managerID
1	1000	Jane	Null
2	1001	Jack	1

Could foreign key be null?

For referential integrity to hold in a relational database, any field in a table that is declared a foreign key should contain either a null value, or only values from a parent table's primary key.

Clicker question

- Consider the table definition: CREATE TABLE Emps (
  (1) id INT,
  Numbers denote lines only → (2) sin INT,
  (3) name CHAR(20),
  (4) managerID INT);

Goal: have managerID be foreign key reference for same table Emps. Which of the following is **not** legal? (does not have to achieve all goals)

A. Add FOREIGN KEY (managerID) REFERENCES Emps(id) before the) on line (4).

B. Add PRIMARY KEY just before the comma on lines (1) and (2), and add REFERENCES Emps(id) before the) on line (4).

C. Add PRIMARY KEY just before the comma on line (1), add UNIQUE just before the comma on line (2), and add FOREIGN KEY REFERENCES Emps(sin) before the) on line (4).

D. All are legal

E. None are legal

Clicker question

- Consider the table definition: CREATE TABLE Emps (

  (1) id INT,

  Numbers denote lines only → (2) sin INT,

  (3) name CHAR(20),

  (4) managerID INT);

Goal: have managerID be foreign key reference for same table Emps. Which of the following is **not** legal? (does not have to achieve all goals)

A. Add FOREIGN KEY (managerID) REFERENCES Emps(id) before the) on line (4). Not legal – you need to reference a key

B. Add PRIMARY KEY just before the comma on lines (1) and (2), and add REFERENCES Emps(id) before the) on line (4). >1 Primary key

C. Add PRIMARY KEY just before the comma on line (1), add UNIQUE just before the comma on line (2), and add FOREIGN KEY REFERENCES Emps(sin) before the) on line (4). Must reference primary key

D. All are legal

E. None are legal Correct Answer

Enforcing Referential Integrity

- *sid* in Grade is a foreign key that references Student.
- What should be done if a Grade tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a **Student tuple** is deleted?
  - Also delete all Grade tuples that refer to it?
  - Disallow deletion of this particular Student tuple?
  - Set sid in Grade tuples that refer to it, to *null*, (the special value denoting *`unknown'* or *`inapplicable'*.)
    - problem if sid is part of the primary key
  - Set sid in Grade tuples that refer to it, to a *default sid*.
- Similar if primary key of a Student tuple is updated

Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates.
  - Default is **NO ACTION** (*delete/update is rejected*)
  - **CASCADE** (also updates/ deletes all tuples that refer to the updated/ deleted tuple)
  - **SET NULL / SET DEFAULT** (referencing tuple value is set to the default foreign key value)

```
CREATE TABLE Grade
  (sid CHAR(8), dept CHAR(4),
    course# CHAR(3), mark INTEGER,
   PRIMARY KEY (sid,dept,course#),
   FOREIGN KEY (sid)
       REFERENCES Student
        ON DELETE CASCADE
        ON UPDATE CASCADE
   FOREIGN KEY (dept, course#)
       REFERENCES Course
        ON DELETE SET DEFAULT
        ON UPDATE CASCADE );
```

Clicker question

Consider the following table definition.

```
CREATE TABLE  BMW  ( bid  INTEGER, sid INTEGER,  …
                    PRIMARY KEY (bid),
                    FOREIGN KEY (sid) REFERENCES STUDENTS
                                      ON DELETE CASCADE);
```

If bid = 1000 and sid = 5678 for a row in Table BMW, choose the best answer

A. If the row for sid value 5678 in STUDENTS is deleted, then the row with bid = 1000 in BMW is automatically deleted.

B. If a row with sid value 5678 in BMW is deleted, then the row with sid=5678 in STUDENTS is automatically deleted.

C. Both of the above.

Clicker question

Consider the following table definition.

```
CREATE TABLE  BMW  ( bid  INTEGER, sid INTEGER,  …
                    PRIMARY KEY (bid),
                    FOREIGN KEY (sid) REFERENCES STUDENTS
                                 ON DELETE CASCADE);
```

If bid = 1000 and sid = 5678 for a row in Table BMW, choose the best answer

A. If the row for sid value 5678 in STUDENTS is deleted, then the row with bid = 1000 in BMW is automatically deleted. A is correct

B. If a row with sid value 5678 in BMW is deleted, then the row with sid=5678 in STUDENTS is automatically deleted.

C. Both of the above.

BMW

bid	Sid
1000	5678

Student

sid	name	Address
5678	James	Null

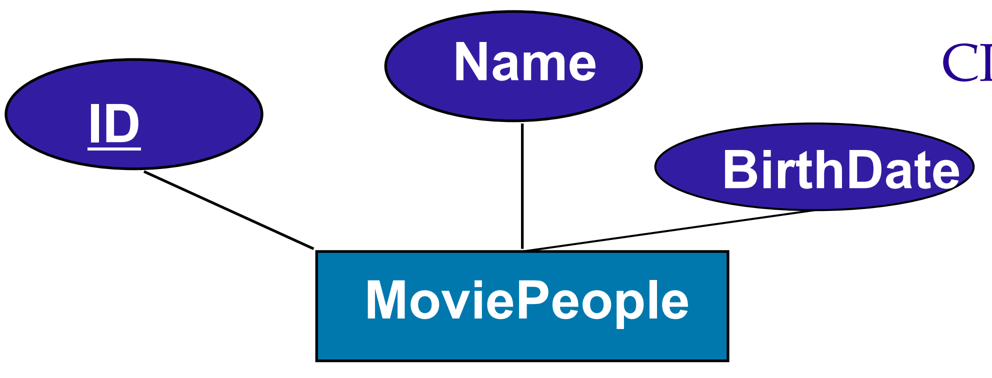
Where do ICs Come From?

- ICs are based upon the real-world semantics being described (in the database relations).
- We *can* check a database instance to verify an IC, but we *cannot* tell the ICs by looking at the instance.
  - For example, even if all student names differ, we cannot assume that name is a key.
  - An IC is a statement about *all possible* instances.
- All constraints must be identified during the conceptual design.
- Some constraints can be explicitly specified in the conceptual model
  - Key and foreign key ICs are shown on ER diagrams.
- Others are written in a more general language.

Logical DB Design: ER to Relational

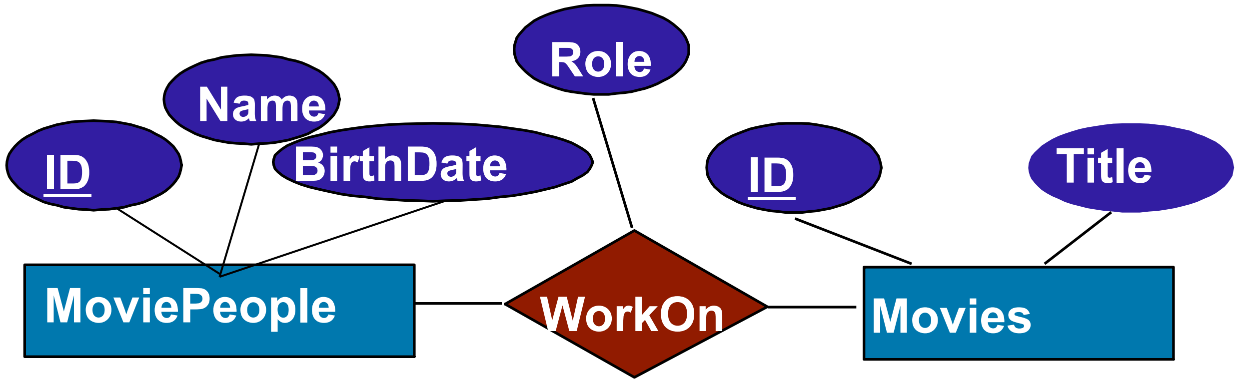
Entity sets to tables.

- Each entity set is mapped to a table.
  - entity attributes become table attributes
  - entity keys become table keys




```
CREATE TABLE MoviePeople
    (ID     CHAR(11),
     Name CHAR(20),
     BirthDate DATE,
     PRIMARY KEY  (ID))
```

Relationship Sets to Tables

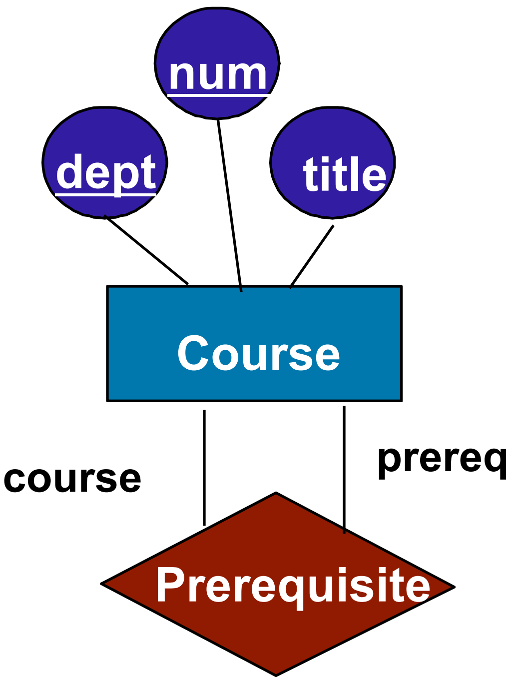


PID and MID CANNOT be null and were renamed

- A relationship set id is mapped to a single relation (table).
- Simple case: relationship has no constraints (i.e. many-to-many)
- In this case, attributes of the table must include:
  - Keys for each participating entity set as foreign keys.
    - This is a *key* for the relation.
  - All descriptive attributes.

```
CREATE TABLE WorkOn(
  PID      CHAR(11),
  MID      INTEGER,
  Role     CHAR(20),
  PRIMARY KEY (PID, MID),
  FOREIGN KEY (PID)
     REFERENCES MoviePeople,
  FOREIGN KEY (MID)
     REFERENCES Movies)
```

Relationship Sets to Tables (cont')

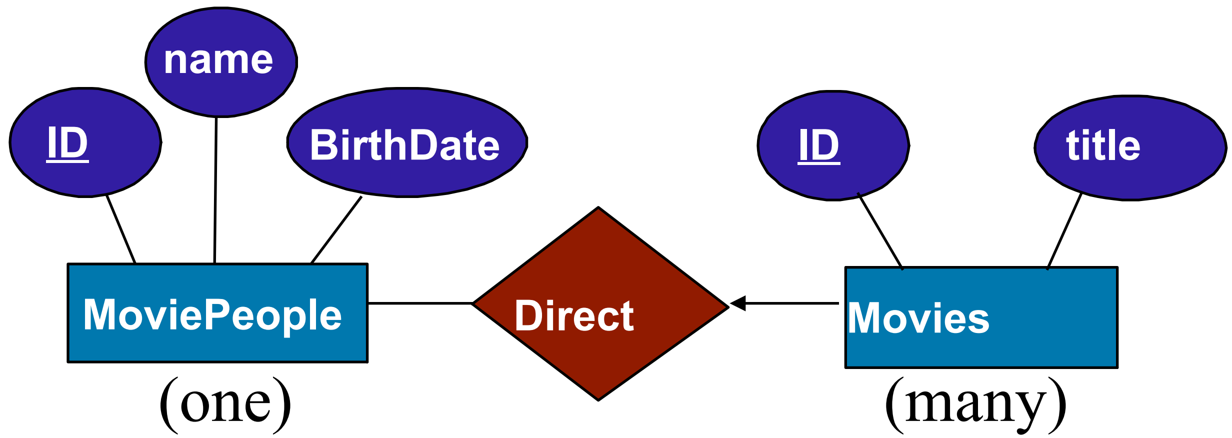


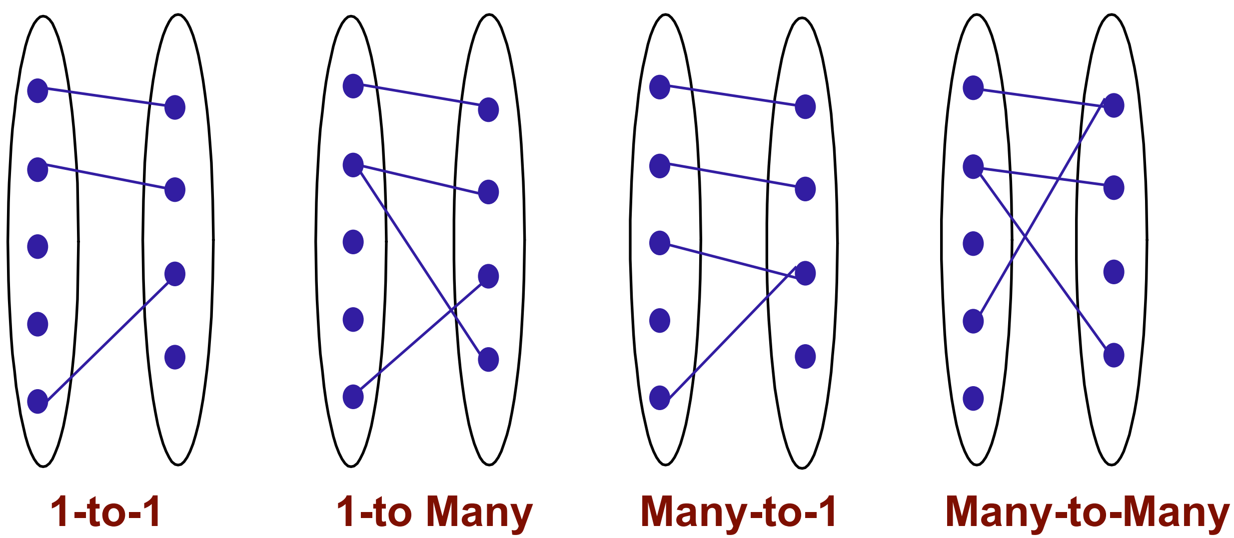
- In some cases, we need to use the roles:

```
CREATE TABLE Prerequisite(
  course_dept    CHAR(4),
  course_num     CHAR(3),
  prereq_dept    CHAR(4),
  prereq_num     CHAR(3),
  PRIMARY KEY (course_dept, course_num,
               prereq_dept, prereq_num),
  FOREIGN KEY (course_dept, course_num)
    REFERENCES Course(dept, num),
  FOREIGN KEY (prereq_dept, prereq_num)
    REFERENCES Course(dept, num))
```

Review: Key Constraints

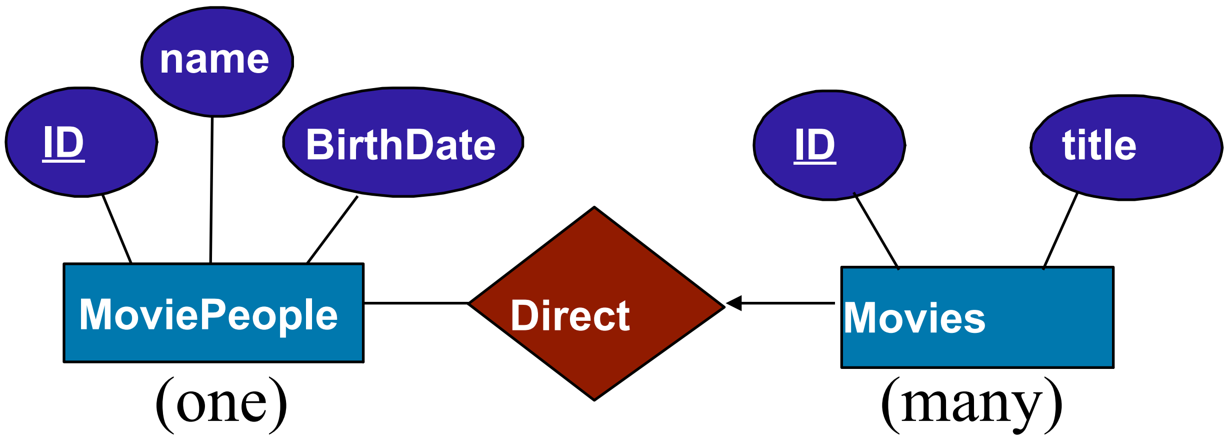
- Each movie has at most one director, according to the *key constraint* on Direct.





Translation to relational model?

Relationship Sets with Key Constraints



- Each movie has at most one director, according to the *key constraint* on Direct.
- How can we take advantage of this?

Translating ER Diagrams with Key Constraints

- Refinement 1 (unsatisfactory):
  - Create a separate table for Direct:
  - Note that MID is the key now!
  - Create separate tables for MoviePeople and Movies.

```
CREATE TABLE  Direct(
  PID     CHAR(11),
  MID     INTEGER,
  PRIMARY KEY  (MID),
  FOREIGN KEY (PID) REFERENCES MoviePeople,
  FOREIGN KEY (MID) REFERENCES Movies)
```

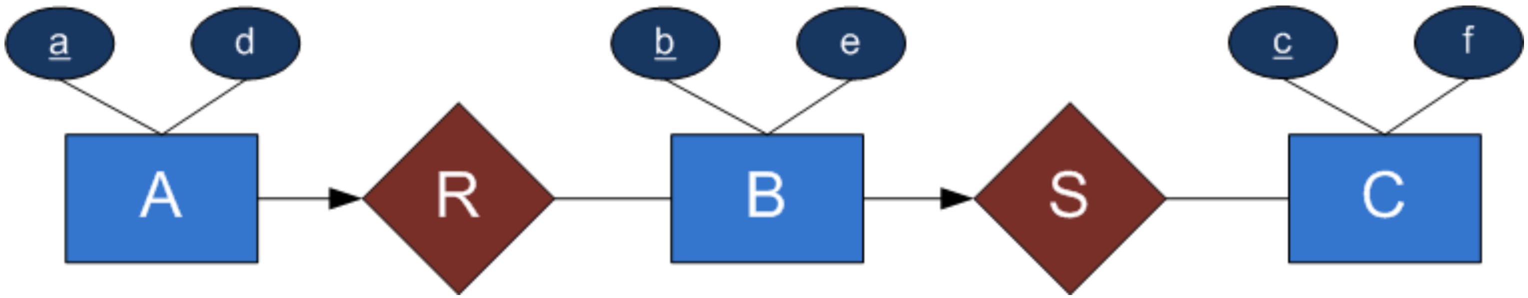
unsatisfactory

- Method 2 (better)
  - Since each movie has a unique director, we can **combine Direct and Movies into one table.**
  - Create another table for MoviePeople

```
CREATE TABLE  Directed_Movie(
  MID        INTEGER,
  title      CHAR(20),
  PID        CHAR(11),
  PRIMARY KEY  (MID),
  FOREIGN KEY (PID) REFERENCES MoviePeople
             ON DELETE SET NULL
             ON UPDATE CASCADE)
```

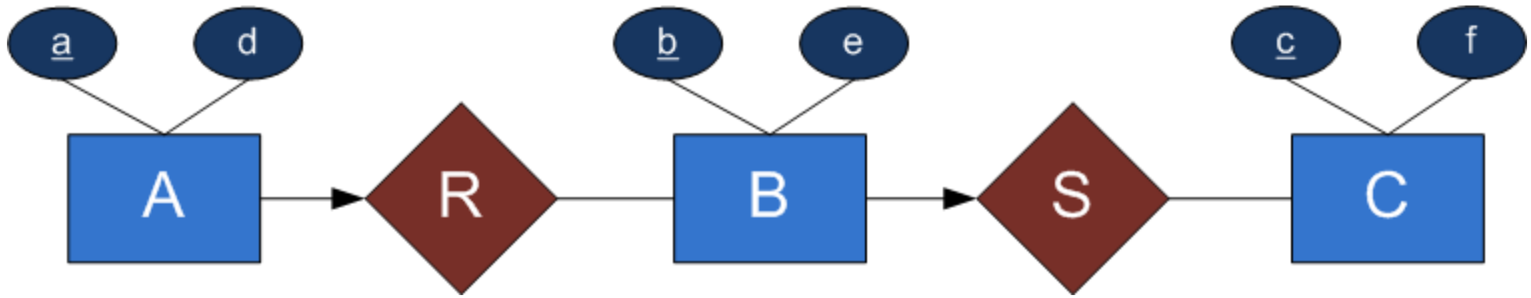
Oracle does not support “on update”

Clicker Question



Translate the ER diagram to relational schemas. Underline key attributes, and make FKs bold.

Clicker Question



Translate the ER diagram to relational.

Which of the following appears in your relational schema:

A. AR(a,**b**,d)

B. BS(b,**c**,e) Correct

C. S(b,**c**)

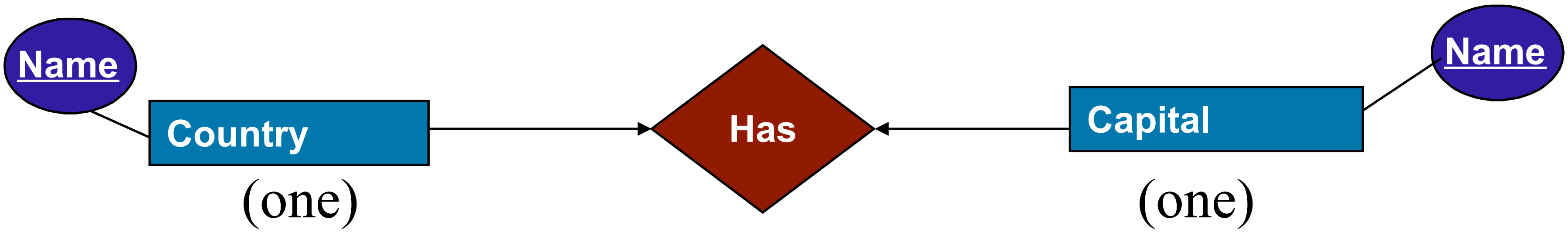
D. All of these

E. None of these

~~A~~
~~B~~ AR
~~C~~ BS → AR(a,**b**,d)
~~R~~ C BS(b,**c**,e)
C(c,f)

Could we move f to BS and remove C?

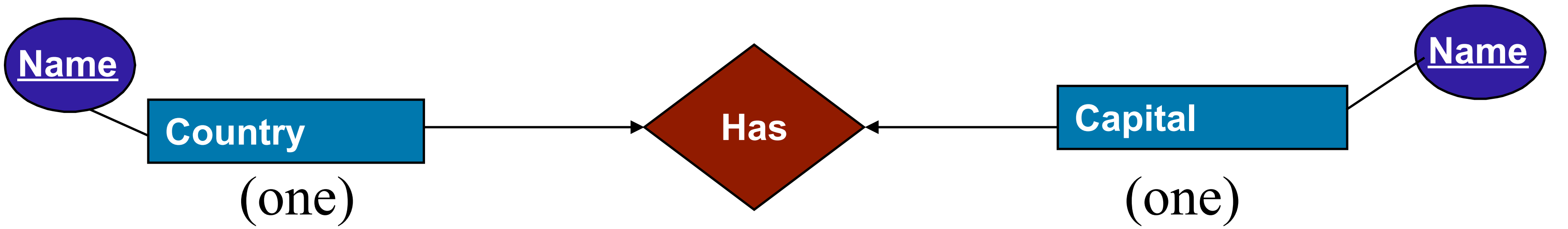
Relationship Sets with Key Constraints (one to one case)



Which schema below is a reasonable translation from ER to relations?

A. Country(<u>coName</u>, caName)
B. Country(name), Capital(name)
C. Capital (<u>caName</u>, coName)
D. Both A and C
E. All of A, B, and C

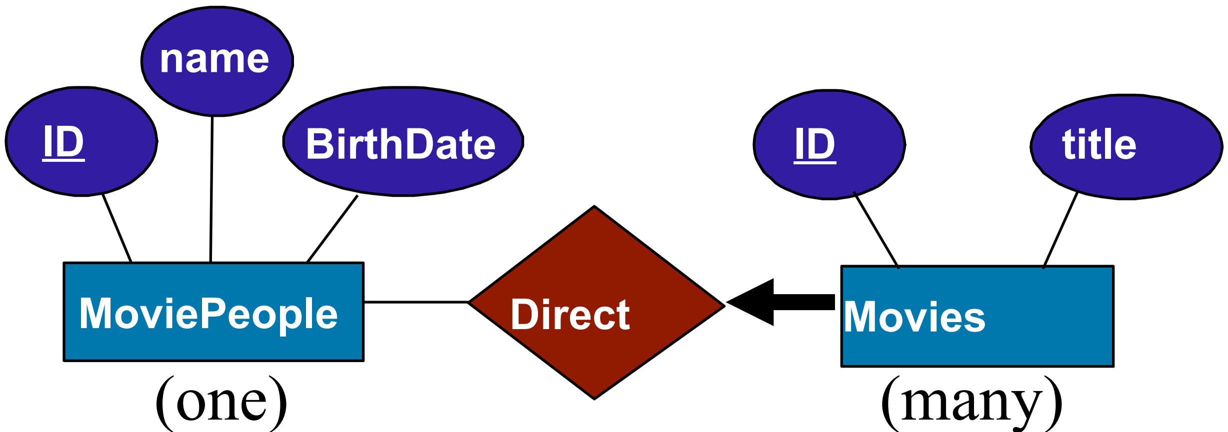
Relationship Sets with Key Constraints (one to one case)



Which schema below is a reasonable translation from ER to relations?

A. Country(<u>coName</u>, caName)
B. Country(name), Capital(name)
C. Capital (<u>caName</u>, coName)
D. Both A and C Correct
E. All of A, B, and C

Translating Participation Constraints



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

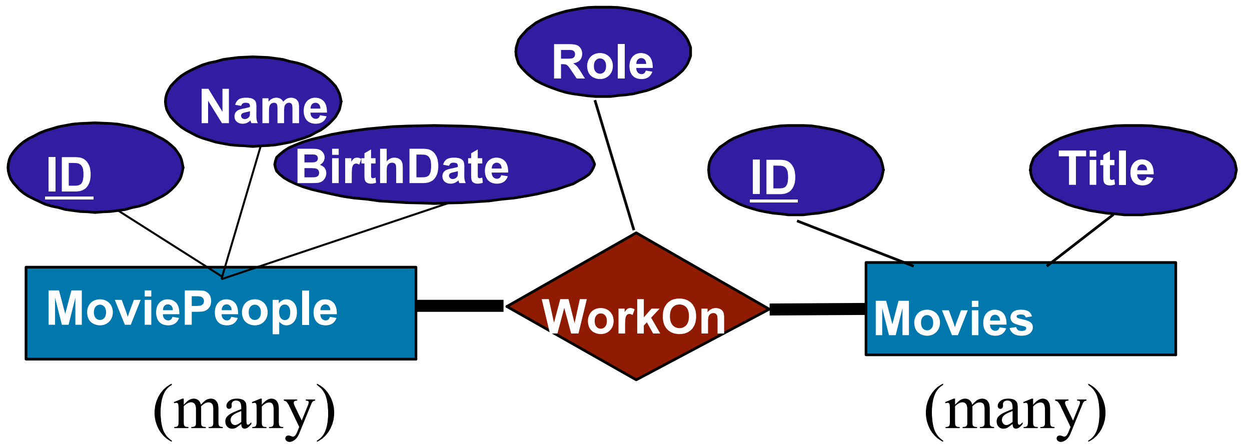
Participation Constraints in SQL

- Using method 2 (add 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 deletion of a director before he/she is replaced

```
CREATE TABLE  Directed_Movie(
  MID         INTEGER,
  title       CHAR(20),
  PID         CHAR(11), NOT NULL
  PRIMARY KEY  (MID),  PK Not null by default
  FOREIGN KEY (PID) REFERENCES MoviePeople
                ON DELETE NO ACTION
                ON UPDATE CASCADE)
```

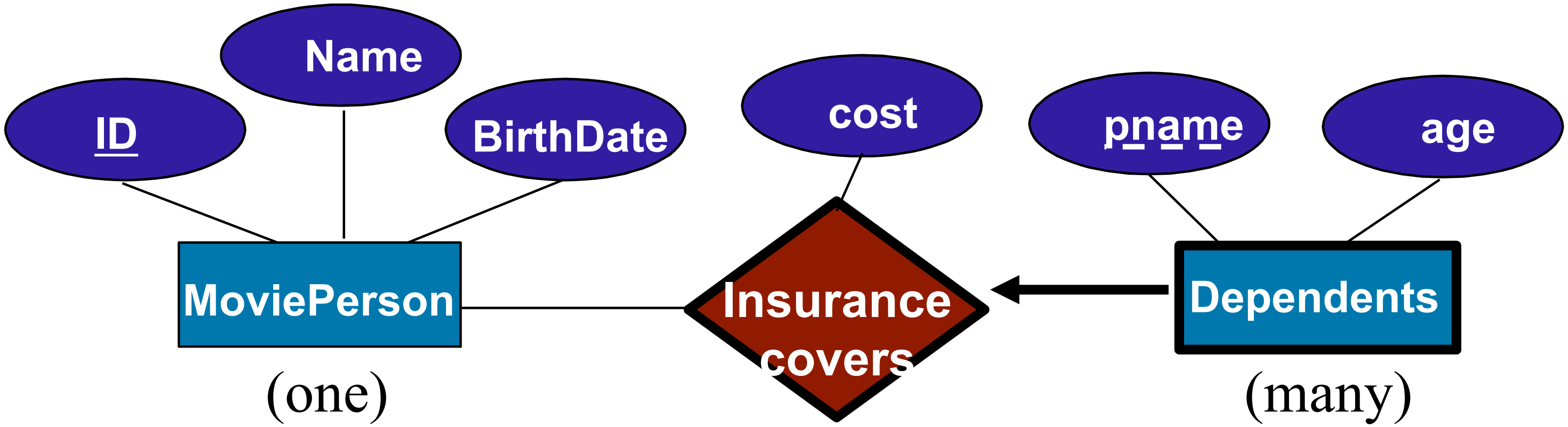
- **Note: We cannot express this constraint if method 1 is used for Direct.**

Participation Constraints in SQL (cont')



- How can we express that "every movie person works on a movie and every movie has some movie person in it"?
- Neither foreign-key nor not-null constraints in WorkOn can do that.
- We need assertions (later)

Translating Weak Entity Sets



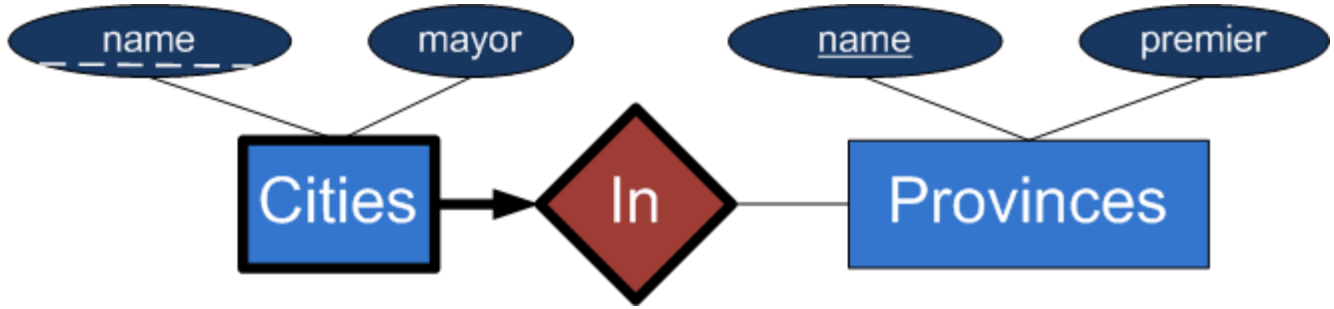
- A ***weak entity*** is identified by considering the primary key of the *owner* (strong) entity.
  - Owner entity set and weak entity set participate in a one-to-many identifying relationship set.
  - Weak entity set has total participation.
- What is the best way to translate it?

Translating Weak Entity Sets(cont')

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

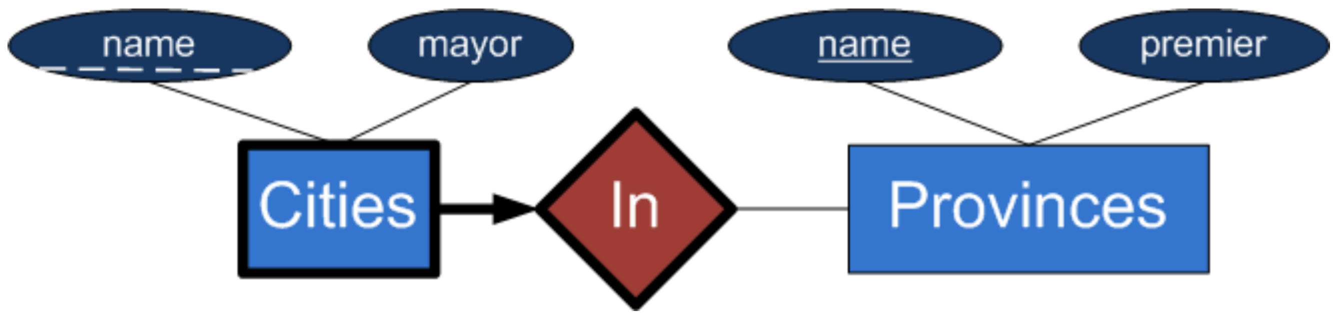
```
CREATE TABLE Dep_Insurance (
  pname  CHAR(20),
  age    INTEGER,
  cost   REAL,
  ID     CHAR(11)
  PRIMARY KEY (ID, pname),
  FOREIGN KEY (ID) REFERENCES MoviePeople,
    ON DELETE CASCADE)
```

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?

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

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

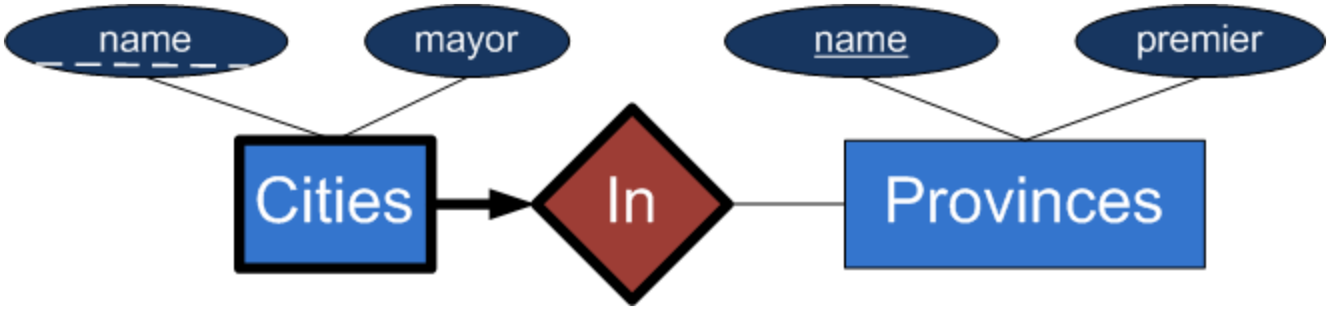
C. Cities(cname, **pname**, mayor), Provinces(pname, premier)

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

E. None of the above

A. Cities W.E, so needs pname
B. Cname not FK
C. Right
D. Don't need separate In

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?

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

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

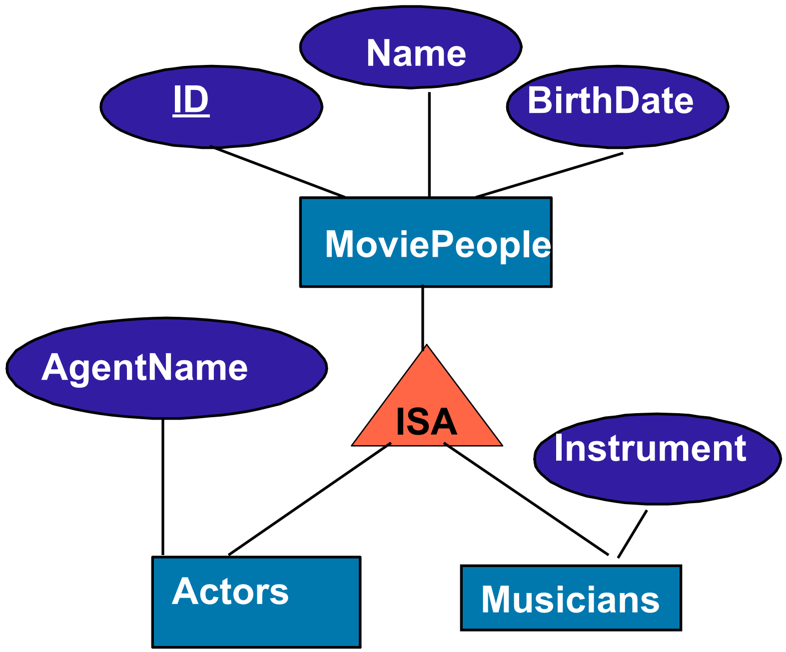
C. Cities(cname, **pname**, mayor), Provinces(pname, premier)

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

E. None of the above

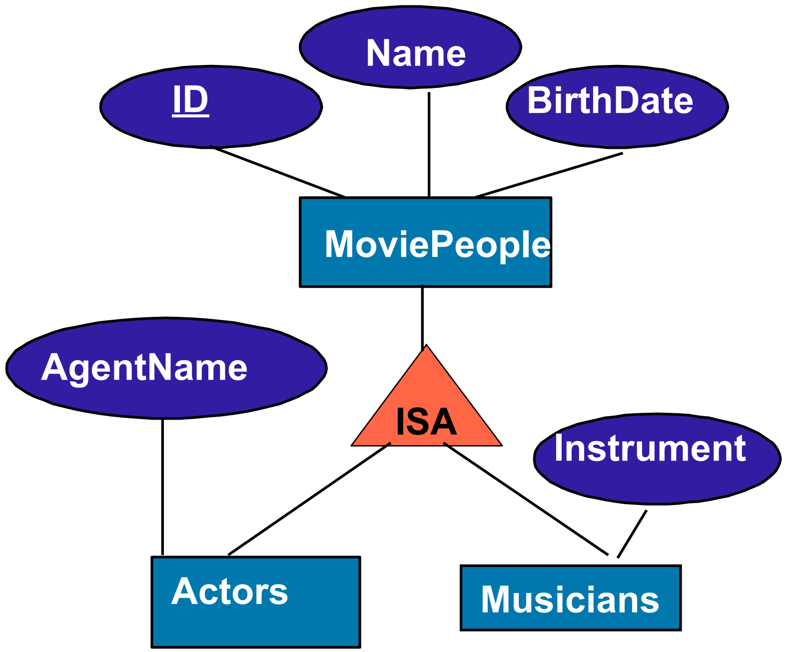
Cities
Provinces
In
→
Cities(cname, **pname**, mayor)
Provinces(pname, premier)

Translating ISA Hierarchies to Relations



What is the best way to translate this into tables?

Totally unsatisfactory attempt: Safest but with lots of duplication (not in book)



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

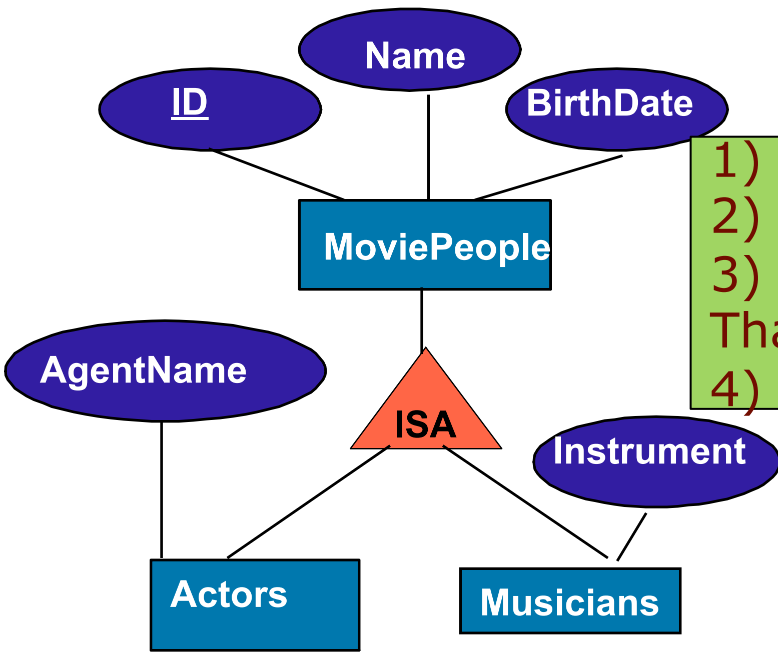
MoviePeople(<u>ID</u>, Name, BirthDate, AgentName, Instrument)

Actors(<u>ID</u>, Name, BirthDate, AgentName, Instrument)

Musicians(<u>ID</u>, Name, BirthDate, AgentName, Instrument)

Way too much duplication! Not a good answer!

Method 1:have only one table with *all* attributes (not in book)



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

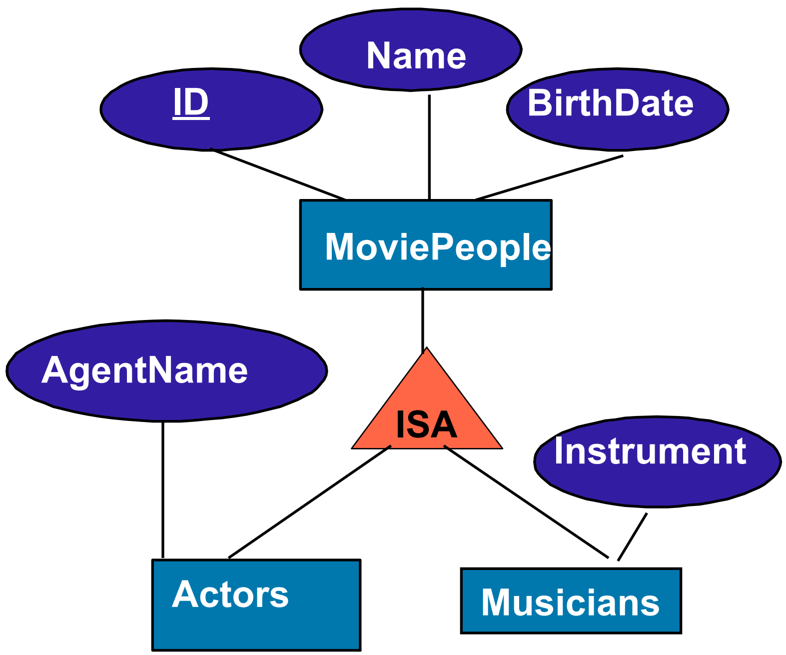
MoviePeople(ID, Name, BirthDate, AgentName, Instrument)

~~Actors(ID, Name, BirthDate, AgentName, Instrument)~~

~~Musicians(ID, Name, BirthDate, AgentName, Instrument)~~

- Lots of space needed for nulls

Method 2: 3 tables, remove excess attributes



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

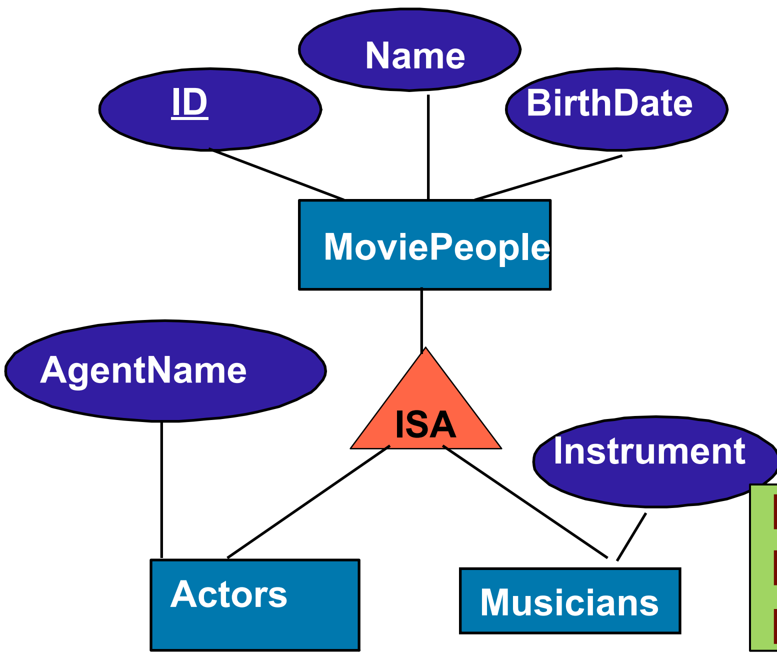
MoviePeople(ID, Name, BirthDate, ~~AgentName, Instrument~~)

Actors(ID, ~~Name, BirthDate~~, AgentName, ~~Instrument~~)

Musicians(ID, ~~Name, BirthDate, AgentName~~, Instrument)

- Works well for concentrating on superclass.
- Have to combine two tables to get all attributes for a subclass

Method 3: 2 tables, none for superclass



- 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?

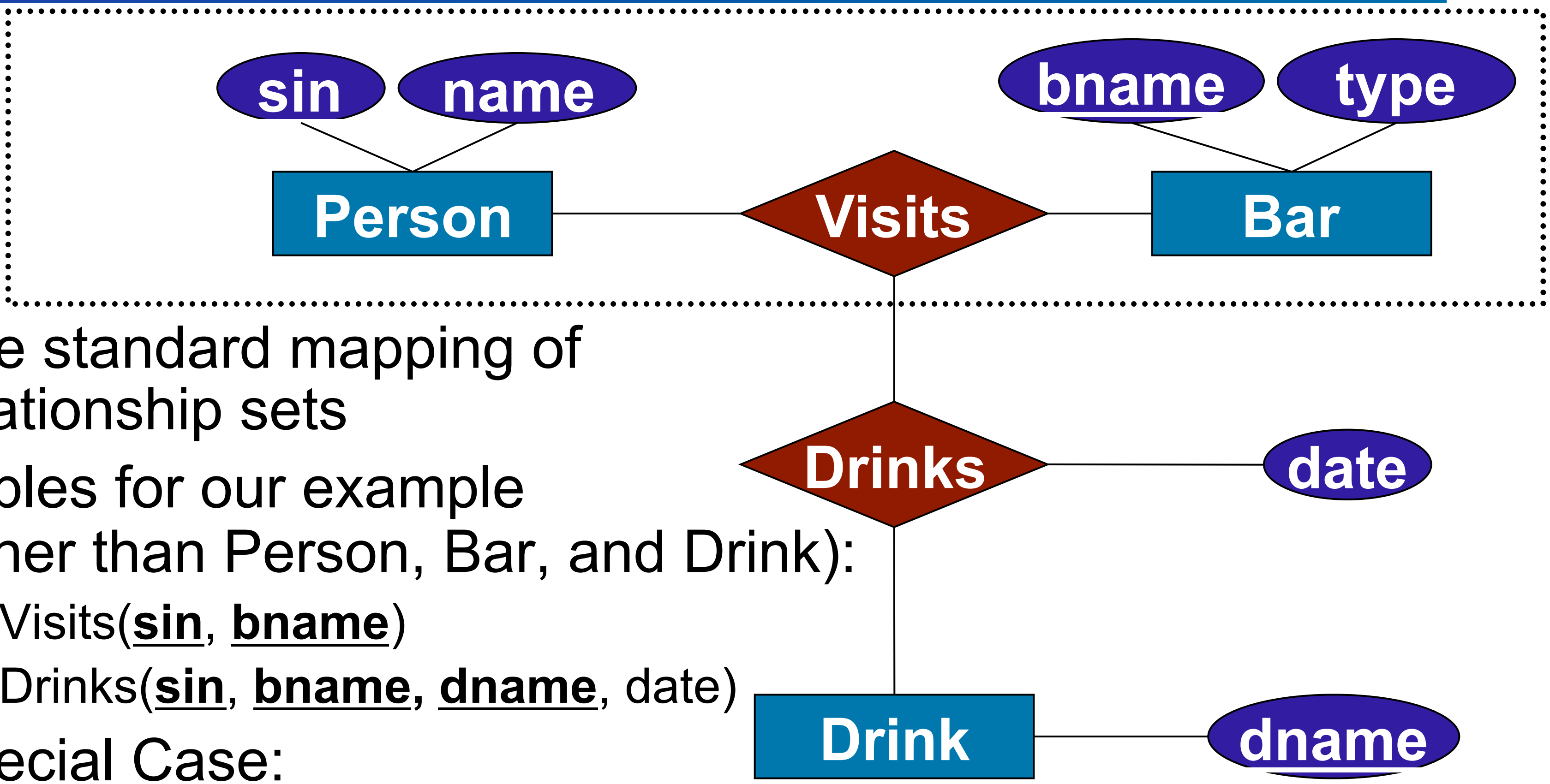
~~MoviePeople(ID, Name, BirthDate, AgentName, Instrument)~~

Actors(ID, Name, BirthDate, AgentName, ~~Instrument~~)

Musicians(ID, Name, BirthDate, ~~AgentName~~, Instrument)

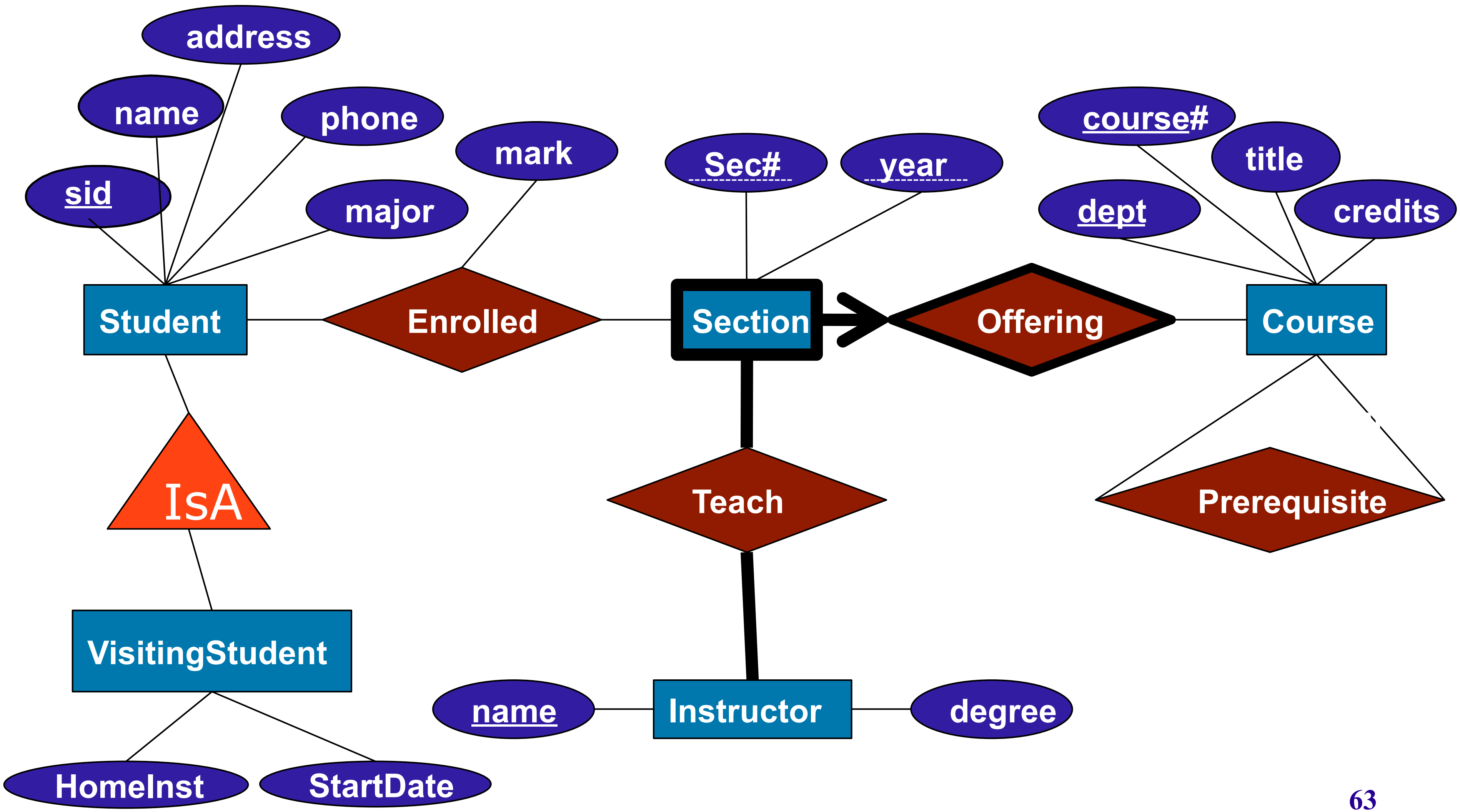
- Works poorly with relationships to superclass
- If ISA-relation is partial, it cannot be applied (loose entities)
- If ISA-relation is not disjoint, it duplicates info

Translating Aggregation



- Use standard mapping of relationship sets
- Tables for our example (other than Person, Bar, and Drink):
  - Visits(<u>**sin**</u>, <u>**bname**</u>)
  - Drinks(<u>**sin**</u>, <u>**bname,**</u> <u>**dname**</u>, date)
- Special Case:
  - If Visits is total on Drinks and Visits has no descriptive attributes we could keep only the Drinks table (discard Visits).

Consider the following diagram for a university. List the tables, keys, and foreign keys when converted to relational. Do not write SQL DDL.



Sample ER to Relational Solution

- *Student (sid, name, address, phone, major)*
- *VisitStudent (sid, homeInst, startDate)*
  - *Foreign keys : (sid)*
- *Course (dept, course#, title, credits)*
- *Instructor(insName, degree)*
- Offering(dept, course#, sec#, year)
  - Foreign keys : (dept, course#)

Sample ER to Relational Solution (cont)

- *Teach(dept, course#, sec#, year, insName)*
  - *Foreign keys: (dept, course#, sec#,*year), (insName)
  - Total participation constraint cannot be enforced for now
- *Enrolled (sid, dept, course#, sec#, year, mark)*
  - *Foreign keys : (sid), (dept, course#, sec#, year)*
- *Prerequisite (courseDept , course#, preDept, pre#)*
  - *Foreign keys : (courseDept , course#), (preDept, pre#)*

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified, based on application semantics. DBMS checks for violations.
  - Important ICs: primary and foreign keys
  - Additional constraints can be defined with assertions (but are expensive to check)
- Powerful and natural query languages exist.
- Rules to translate ER to 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 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 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.