CS 5200
Database Systems

The Relational Model

References:

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

Hazra Imran

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

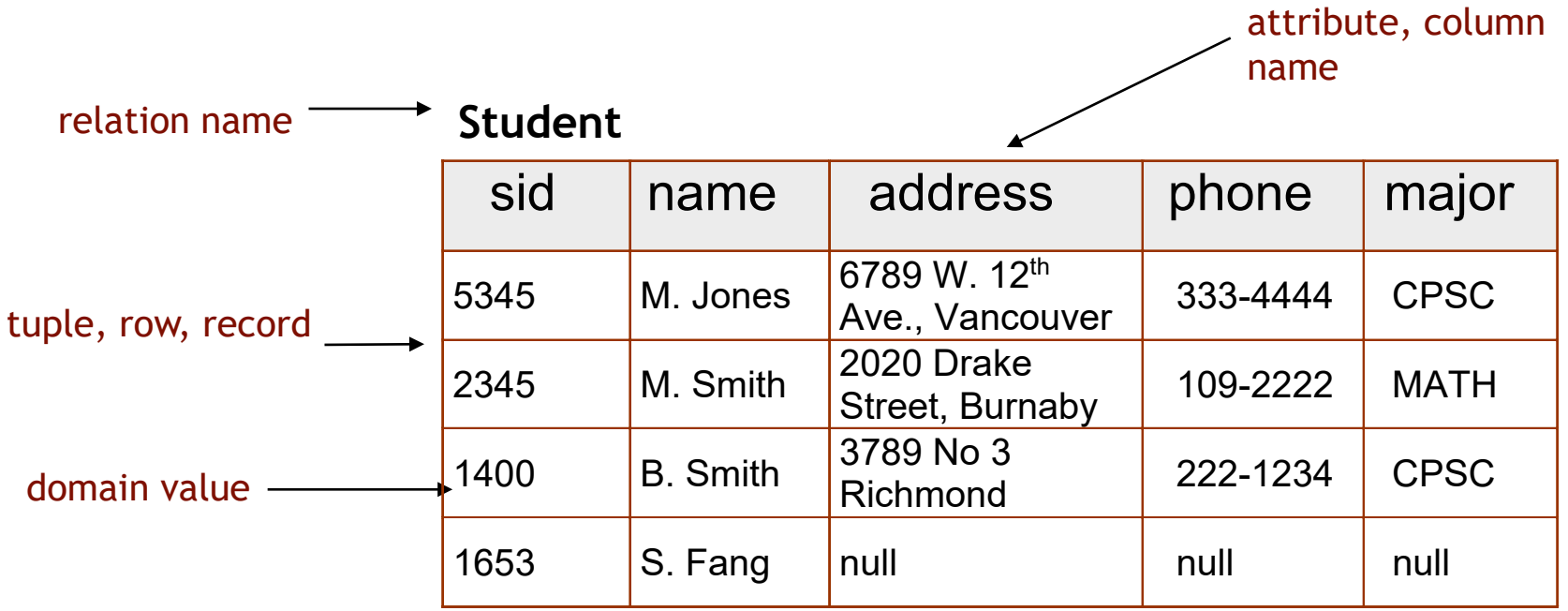
- *Database Instance:* a collection of instances of its relations

Example of a Relation Instance

Student

sid	name	address	phone	major
5345	M. Jones	6789 W. 12th Ave., Vancouver	333-4444	CPSC
2345	M. Smith	2020 Drake Street, Burnaby	109-2222	MATH
1400	B. Smith	3789 No 3 Richmond	222-1234	CPSC
1653	S. Fang	null	null	null

Example of a Relation Instance



sid	name	address	phone	major
5345	M. Jones	6789 W. 12th Ave., Vancouver	333-4444	CPSC
2345	M. Smith	2020 Drake Street, Burnaby	109-2222	MATH
1400	B. Smith	3789 No 3 Richmond	222-1234	CPSC
1653	S. Fang	null	null	null

- 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 $D_1,. ..D_n$, 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

Example of a formal definition

Student

sid	name	address	phone	major
5345	M. Jones	6789 W. 12th Ave., Vancouver	333-4444	CPSC
2345	M. Smith	2020 Drake Street, Burnaby	109-2222	MATH
1400	B. Smith	3789 No 3 Richmond	222-1234	CPSC
1653	S. Fang	null	null	null

Relational Schema

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

Or, without the domains:

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

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.

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)
  - …SQL-2003,2008, 2011, 2016

(http://www.jcc.com/resources/sql-standards)

Raymond Boyce

Don Chamberlain

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

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

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

sid	name	phone
5345	M. Jones	333-4444
1400	B. Smith	222-1234

Student

sid	name	address	phone	major
5345	M. Jones	6789 W. 12th Ave., Vancouver	333-4444	CPSC
2345	M. Smith	2020 Drake Street, Burnaby	109-2222	MATH
1400	B. Smith	3789 No 3 Richmond	222-1234	CPSC
1653	S. Fang	null	null	null

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

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

Student

sid	name	address	phone	major
5345	M. Jones	…	…	CPSC
…	…	….	…	…

Grade

sid	dept	Course#	marks
5345	CPSC	354	86
…	…	….	…

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

```
SELECT name, marks
FROM Student, Grade
WHERE Student.sid = Grade.sid AND dept = 'CPSC'
```

Creating Relations in SQL/DDL

- Creates the Student relation
  - the type (domain) of each attribute is specified
  - enforced when tuples are added or modified

```
CREATE TABLE Student
       (sid INTEGER,
        name  CHAR(20),
        address CHAR(30),
        phone CHAR(13) DEFAULT '999999999',
        major  CHAR(4))
```

String type with different length

sid	name	address	phone	major
5345	M. Jones	…	…	CPSC
…	…	….	…	…

- The Grade table holds information about courses that a students takes

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

sid	dept	Course#	mark
5345	CPSC	354	86
…	…	….	…

Deleting Table

To delete a table use the DROP TABLE statement

DROP TABLE Student

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

Altering Table

- Columns can be added or removed to tables using the **ALTER TABLE** statement
  - ADD to add a column and
  - DROP to remove a column

```
ALTER TABLE Student
ADD 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.

```
ALTER TABLE Student
DROP COLUMN gpa;
```

Null Values

- A field can have a special value NULL.
- Null can many things.
  - Field is unknown or missing or unavailable
  - Field may not apply in certain cases.
- Not allowing NULL in some cases is a way to enforce total participation.

Adding Tuples

- Can insert a single tuple using:

INSERT INTO Student (sid, name, address, phone, major)
VALUES ('5345', 'M. Currie', '1235 Burrad St', '882-4444', 'PHYS')

sid	name	address	phone	major
5345	M. Currie	12345 Burrad St	882-4444	PHYS
1234	Smith	234 Kingsway	567-9878	CS
3453	Smith	345 Terminal Ave	567-4532	MATH
…	…	…	…	…

The list of column names is optional

- If omitted the values must be in the same order as the columns

Deleting Tuples

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

```
DELETE
FROM    Student
WHERE   name = ‘Smith’
```

sid	name	address	phone	major
5345	M. Currie	12345 Burrad St	882-4444	PHYS
1234	Smith	234 Kingsway	567-9878	CS
3453	Smith	345 Terminal Ave	567-4532	MATH
…	…	…	…	…

→ *Powerful variants of these commands exist; more later*

Update Tuples

- Use the UPDATE statement to modify a record, or records, in a table
- Note that the WHERE statement is evaluated before the SET statement
- Like DELETE the WHERE clause specifies which records are to be updated

```
UPDATE STUDENT
SET address ='333 Argyle Street'
WHERE   name = 'M. Currie'
```

Integrity Constraints (ICs)

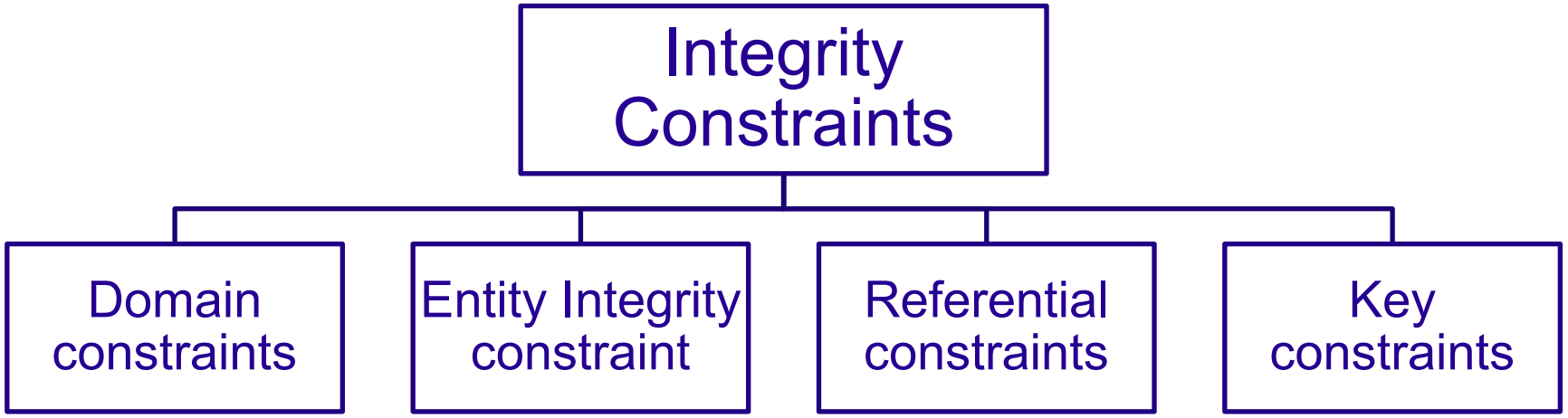
"Integrity is doing the right thing, even when no one is watching" (By : C.S. Lewis)

- **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.
  - Next up: constraints for relational databases

IC


Integrity Constraints
Domain constraints
Entity Integrity constraint
Referential constraints
Key constraints

Candidate Key

- A set of fields is a **superkey** for a relation if :
  - No two distinct tuples can have same values in all key fields.
- **Candidate key** = minimal superkey = no subset of fields is a superkey.
- 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,
         name     CHAR(20),
         address  CHAR(30),
         phone    CHAR(13),
         major    CHAR(4))
```

PRIMARY KEY 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*

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

- Other keys are specified using the **UNIQUE** constraint.
  - The 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

Super key, Candidate key and Primary key

Students

sid	name	address	phone	major

PRIMARY KEY in SQL (cont')

(Ex.1- Normal) "For a given student and course, there is a single mark's value."

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

VS

(Ex.2 - Bad) "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),
     marks    INTEGER,
     PRIMARY KEY(sid,dept,course#),
     UNIQUE (dept,course#,mark) )
```

Grade

sid	dept	Course#	mark

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

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#*, *marks*)
  - *sid* is a foreign key referring to Student:
  - (*dept*, *course#*) is a foreign key referring to Course

Student				
SID	Name	Address	Phone	Major

Grade			
SID	Dept	course#	Marks

Course			
Dept	Course#	cname	credits

Referential Integrity

- All foreign keys reference existing entities.
  - i.e. there are no dangling references
  - all foreign key constraints are enforced

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), marks 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#	marks
5345	CPSC	101	80
5345	RELG	100	45
1400	MATH	200	null
5345	HIST	201	60

Student

sid	name	address	Phone	major
5345	M. Jones	….	…	…
1400	B. Smith	….	…	…
2354	M. Smith	….	…	…

Self Referencing Relations

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

id	sin	name	managerID
1	1000	John	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.

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?
- What should be done if a **Student tuple** is deleted , e.g. sid = 5345? (*Reject it!*)
  - 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

Grade

sid	**dept**	**course#**	marks
5345	CPSC	101	80
5345	RELG	100	45
1400	MATH	200	null
5345	HIST	201	60

Student

sid	name	address	Phone	major
5345	M. Jones	….	…	…
1400	B. Smith	….	…	…
2354	M. Smith	….	…	…

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),
      marks 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 );
```

Oracle does not support “on update”

Where do ICs Come From?

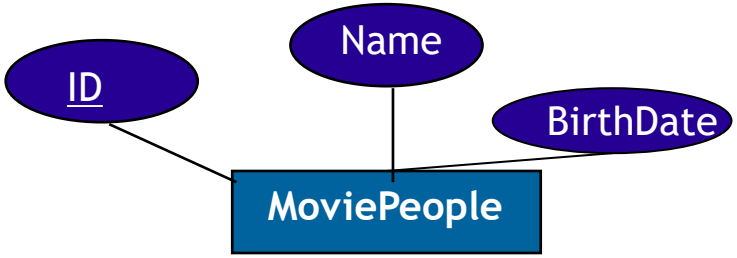
- ICs are based upon the real-world semantics being described (in the database relations).
- We *can* check a database instance to verify an IC, but we *cannot* tell the ICs by looking at the instance.
  - For example, even if all student names differ, we cannot assume that name is a key.
  - An IC is 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
- Others are written in a more general language.

Mapping of ERD to relation model

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))
```

MoviePeople

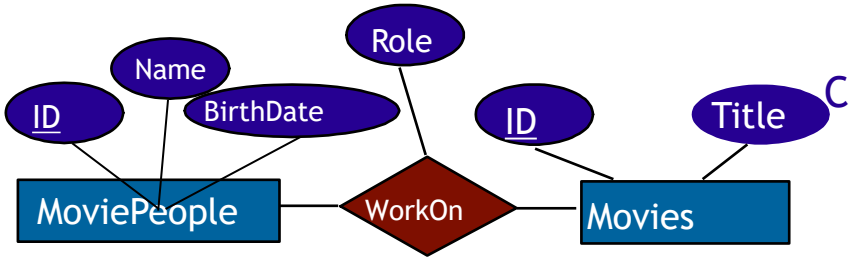
ID	Name	BirthDate
MP001	Chris Columbus	10/09/1958
MP002	Tom Hardy	15/09/1977
MP003	Kate Winslet	05/10/1975

Movie

ID	title
1	Harry Potter
2	The Drop
4	Titanic

Relationship Sets to Tables

- A relationship set 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)
```

PID and MID CANNOT be null and were renamed

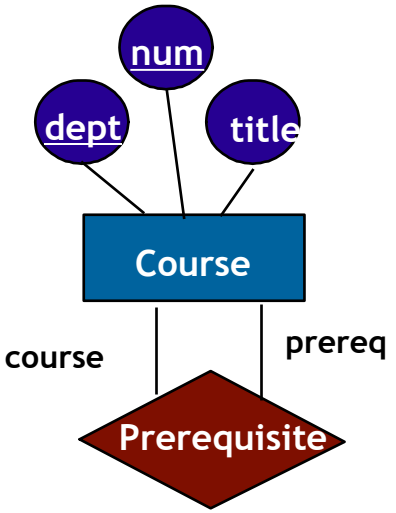
WorksOn

PID	**MID**	Role
MP002	2	Bob Saginowski
MP003	4	Rose DeWitt

Bold column means FK

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))
```

Prerequisite

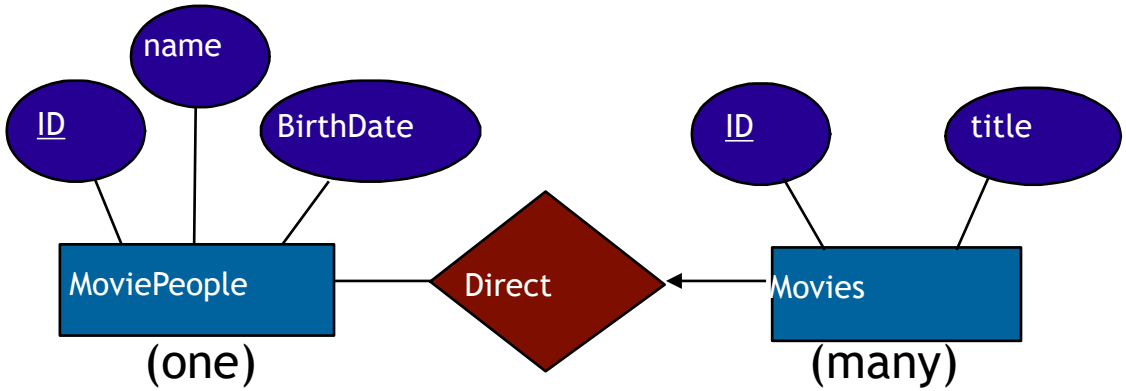
Course_dept	Course_num	prereq_dept	prereq_num
CPSC	304	CPSC	221
CPSC	304	EECE	320
….	…	….	…

Course

dept	num	title
CPSC	304	Database Systems
CPSC	221	Basic Algo and DS
EECE	320	Discrete Structures and Algorithms
….	…	….

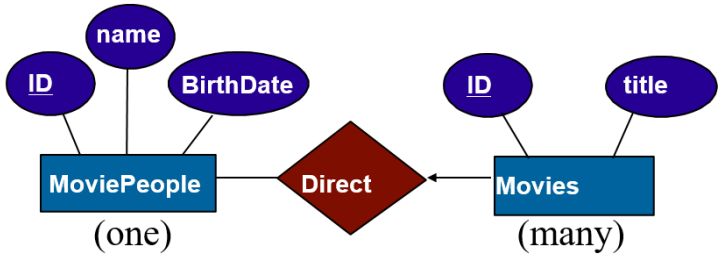
Review: Key Constraints

- 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



- Method 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(
   MID    INTEGER,
  PID     CHAR(11),
   PRIMARY KEY  (MID),
   FOREIGN KEY (PID) REFERENCES MoviePeople,
   FOREIGN KEY (MID) REFERENCES Movies)
```

MoviePeople

ID	Name
MP001	Chris Columbus
MP002	Tom Hardy
MP003	Kate Winslet

Movie

ID	title
1	Harry Potter
2	The Drop
4	Titanic

Direct

MID	PID
1	MP001
2	Null
4	Null

What if Chris is deleted from MoviePeople?

Vs.

- 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)
```

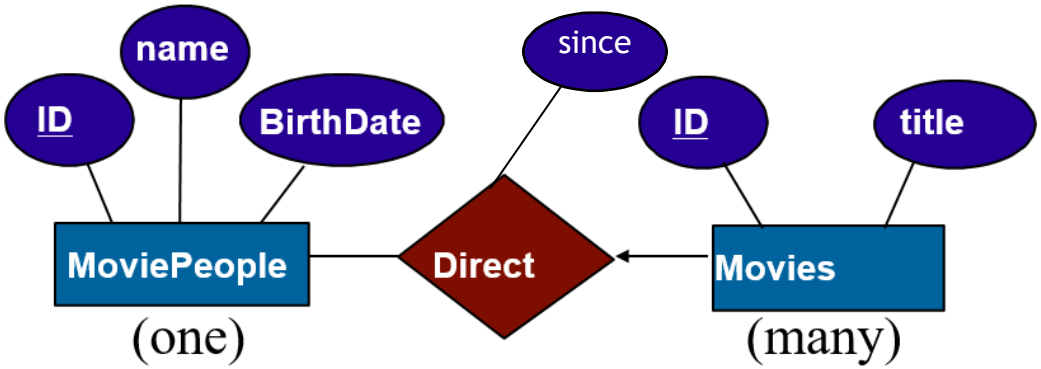
Directed_Movie

ID	title	PID
1	Harry Potter	MP001
2	The Drop	Null
4	Titanic	Null

MoviePeople

ID	Name
MP001	Chris Columbus
MP002	Tom Hardy
MP003	Kate Winslet

If relationship has descriptive attributes then?



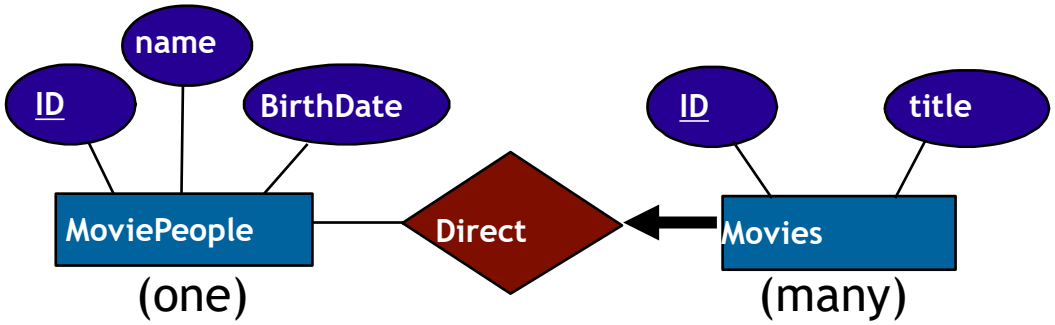
Directed_Movie

ID	title	**PID**	Since
1	Harry Potter	MP001	4 Jan 2018
2	The Drop	NULL	NULL
4	Titanic	NULL	NULL

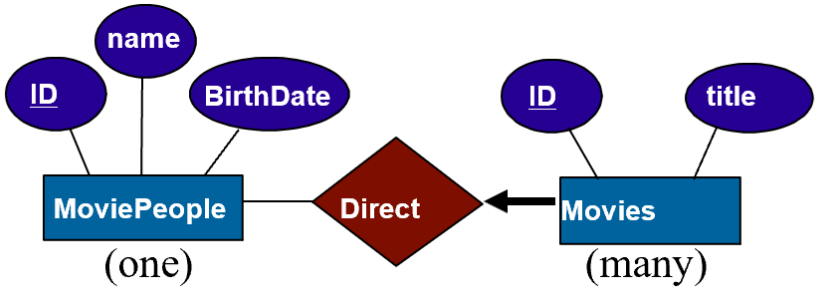
MoviePeople

ID	Name
MP001	Chris Columbus
MP002	Tom Hardy
MP003	Kate Winslet

What if “Every Movie must have a director”



What if "Every Movie must have a director" directed_movie table




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

Vs.

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

A MoviePeople tuple cannot be deleted if it is pointed to by a directed_movie tuple

MoviePeople

ID	Name
MP001	Chris Columbus
MP002	Tom Hardy
MP003	Kate Winslet

Movie

ID	title
1	Harry Potter
2	The Drop
4	Titanic

Direct

MID	**PID**
1	MP001
2	Null
4	Null

Directed_Movie

ID	title	**PID**
1	Harry Potter	MP001

MoviePeople

ID	Name
MP001	Chris Columbus
MP002	Tom Hardy
MP003	Kate Winslet

What if Chris is deleted from MoviePeople?

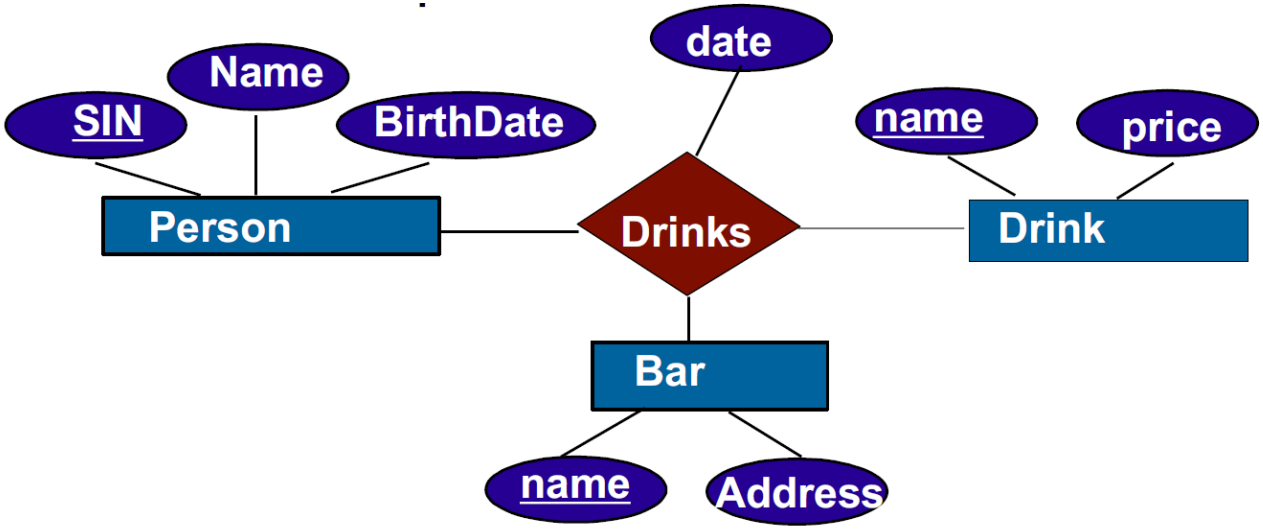
Summary

- When there is no cardinality constraint (i.e., M:M relationship), one table per participating entity set and one separate table for relationship set.

- When there is a cardinality constraint (e.g., many-to-one),
  - one table for the “one” side (e.g., MoviePeople)
  - one table for the “many” side which also captures the relationship (e.g. Movies + Direct)

Key constraints on non-binary relationships

- Ternary relation Drinks relates entity sets Person, Bar and Drink, and has descriptive attribute *date*.

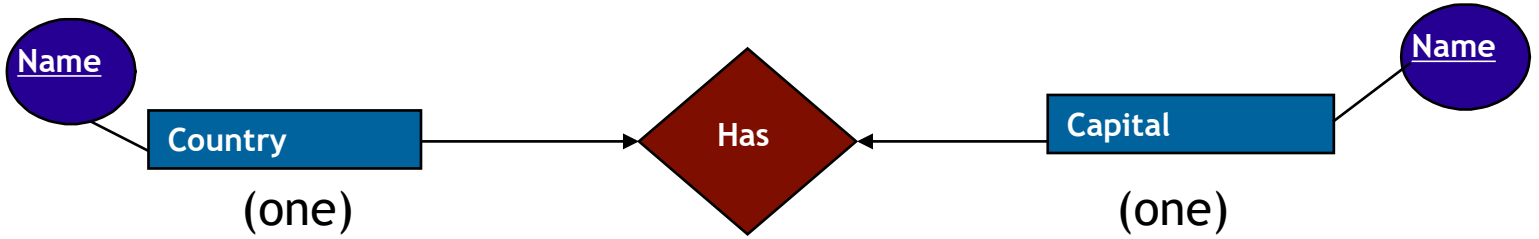


We'd covered basic translating of ER to relational

- Short version: everything's a table

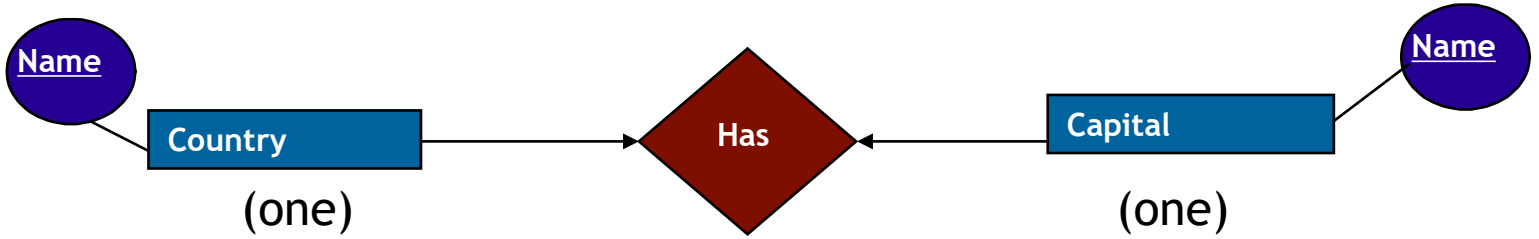
- Slightly longer version:
  - In many to many relationships, create one table per entity and one table per relationship. Link the two by foreign keys
  - In 1:M, merge the relationship with the entity having cardinality M
  - For total participation, use NOT NULL constraint

Relationship Sets with Key Constraints (one to one case)



- Let's assume we went with Country(coName, caName) and all attributes have type Char(20) and we're not creating a separate relation for Capital. Write the SQL DDL that you would need for this relation.

Relationship Sets with Key Constraints (one to one case)



- Let's assume we went with Country(coName, caName) and all attributes have type Char(20) and we're not creating a separate relation for Capital. Write the SQL DDL that you would need for this relation.

```
Create table country(
coname char(20) Primary key,
Caname char(20) unique NOT NULL
);
```

Let see … (SQL Server)

```
Create table country(
coname char(20) Primary key,
Caname char(20) unique
);
```

```
insert into country values ('Canada', NULL);
insert into country values ('India', 'New Delhi');
insert into country values ('Spain', NULL);
```

```
Violation of UNIQUE KEY constraint 'UQ__country__9EF576D51C2A0B95'.
Cannot insert duplicate key in object 'dbo.country'. The duplicate
key value is (<NULL>).
The statement has been terminated.
```

	coname	Caname
1	Canada	NULL
2	India	New Delhi

```
Create table country(
coname char(20) Primary key,
Caname char(20) unique not null,
);
```

```
insert into country values ('Spain', 'MADRID');
insert into country values ('India', 'New Delhi');
insert into country values ('Canada', 'OTTAWA');
insert into country values ('Spain', NULL);
insert into country values ('France', 'MADRID');
```

	coname	Caname
1	Spain	MADRID
2	India	New Delhi
3	Canada	OTTAWA

Let see… (Oracle)

```
create table country1(
coname char(20) Primary key,
Caname char(20) unique
);
```

```
insert into country1 values ('Canada', NULL);
insert into country1 values ('India', 'New Delhi');
insert into country1 values ('Spain', NULL);
```

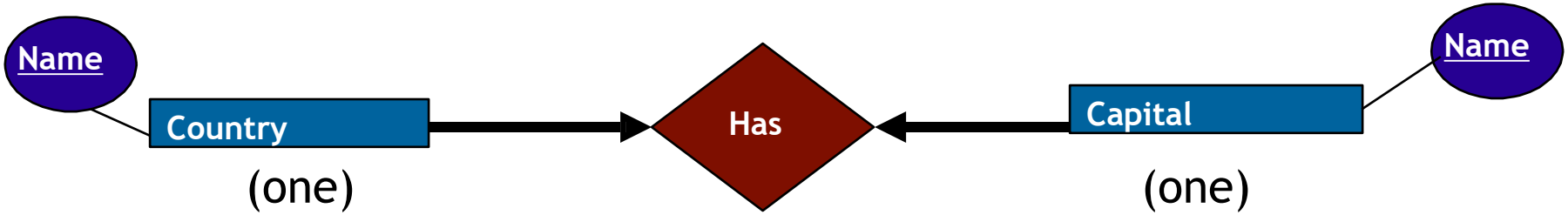
CONAME	CANAME
Canada	-
India	New Delhi
Spain	-

Create table country2(
coname char(20) Primary key,
Caname char(20) unique NOT NULL
);

```
insert into country2 values ('India', 'New Delhi');
insert into country2 values ('Canada', 'OTTAWA');
insert into country2 values ('Spain', 'MADRID');
insert into country2 values ('Spain', NULL);
insert into country2 values ('France', 'MADRID');
```

CONAME	CANAME
India	New Delhi
Canada	OTTAWA
Spain	MADRID

Relationship Sets with Key Constraint + total participation

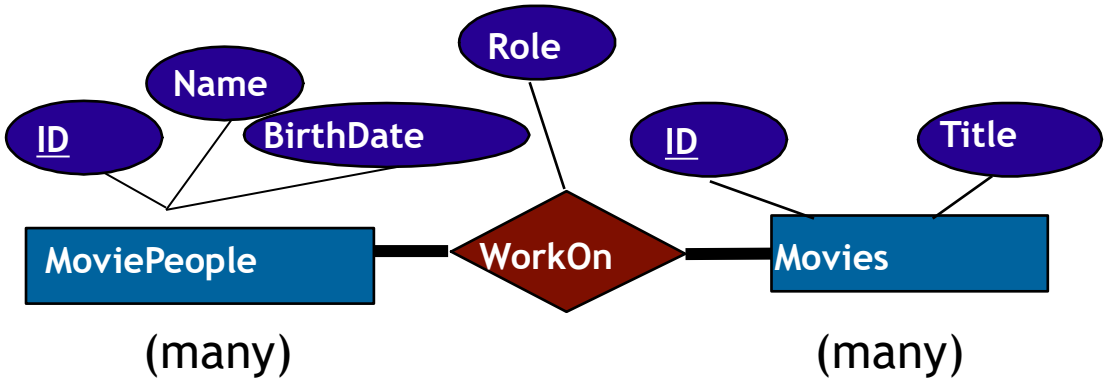


Let's assume we went with Country(coName, caName). Do we need a separate relation for Capital?

A. Yes
B. No
C. It depends

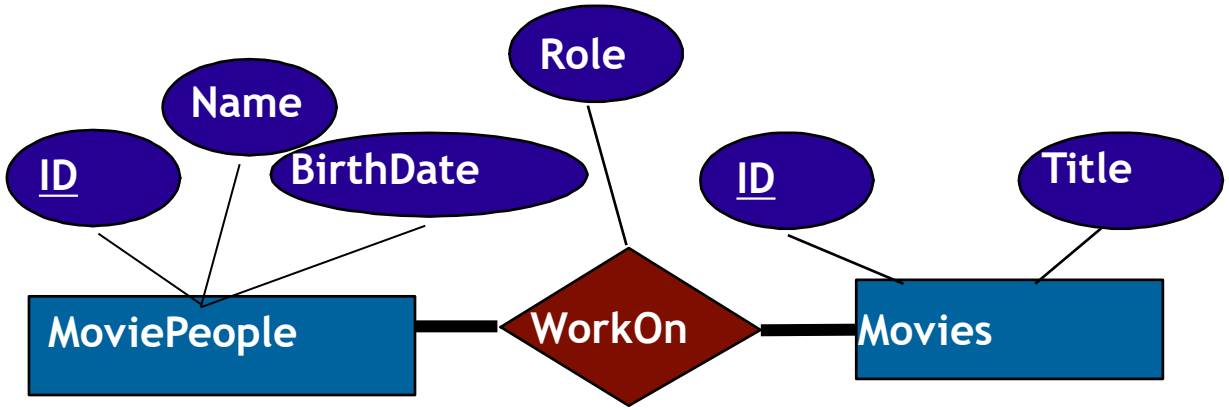
Participation Constraints in SQL (cont')

- 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)

Let's see why we can't model this participation constraint using null constraint



MoviePeople

ID	Name	BirthDate
MP001	Chris Columbus	10/09/1958
MP002	Tom Hardy	15/09/1977
MP003	Kate Winslet	05/10/1975
MP004	Christian Bale	30/01/1974

Movie

ID	title
1	Harry Potter
2	The Drop
3	Legend
4	Titanic

WorksOn

NOT NULL by default

PID	MID	Role
MP002	2	Bob Saginowski
MP003	4	Rose DeWitt

- We don't have all movies and all MoviePeople in the Works on. But its okay.
- Instance is a **legal** one!

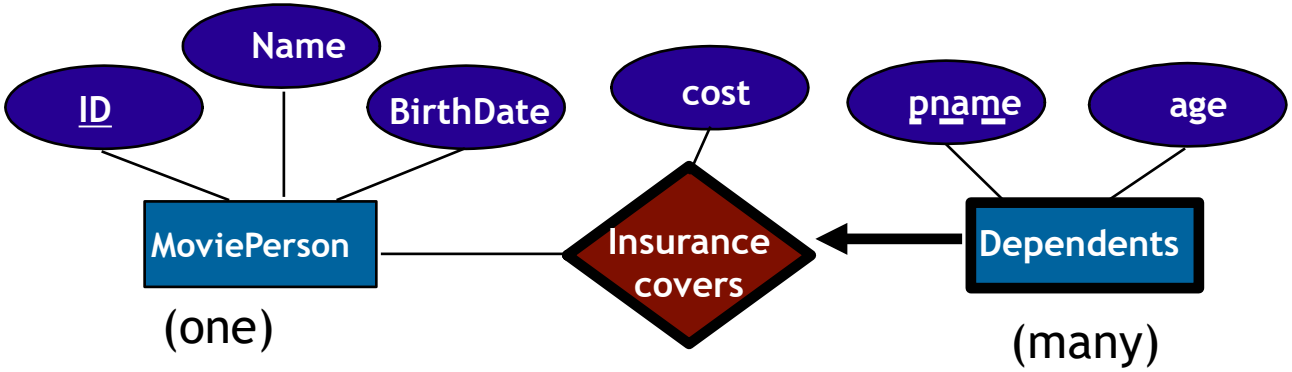
Glimpse of assertion

```
CREATE ASSERTION  totalEmployment
CHECK
( NOT EXISTS ((SELECT PID FROM MoviePeople)
                        EXCEPT
                       (Select PID FROM WorkIn)));
```

Translating Weak Entity Sets

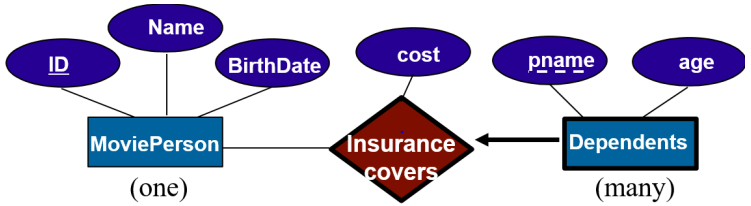
A ***weak entity*** is identified by considering the primary key of the *owner* (strong) entity.

- 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')



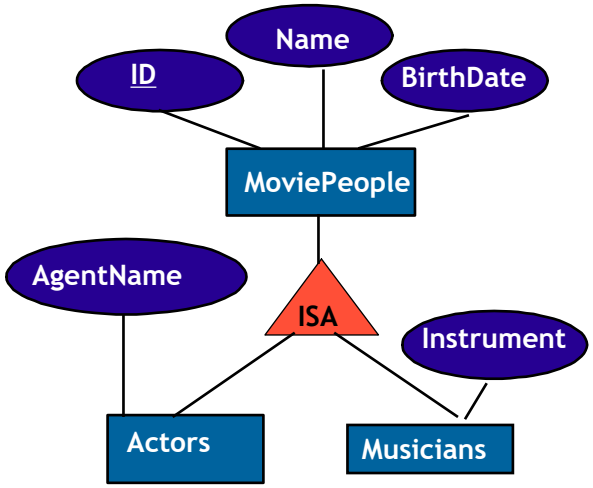
- Weak entity set and its identifying relationship set are translated into a single table.
  - Primary key would consist of the owner's primary key and weak entity's partial key
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
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)
```

Same can be done in 1: M as well. If relationship has descriptive attribute, we can merge it. Just remember to have "CASCADING constraint" .

Translating ISA Hierarchies to Relations

What is the best way to translate this into tables?



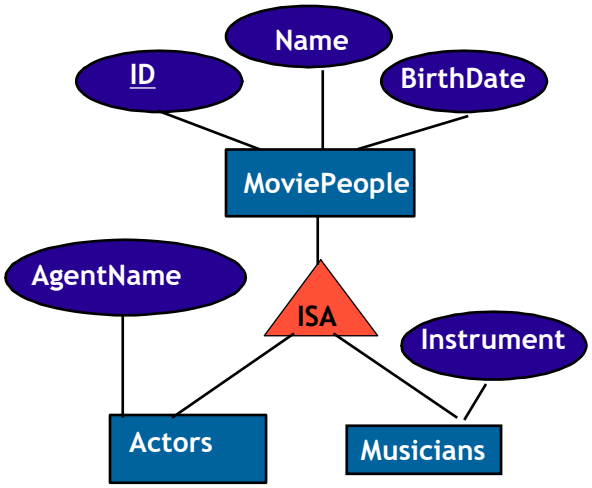
Totally unsatisfactory attempt: Safest but with lots of duplication

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

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

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

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



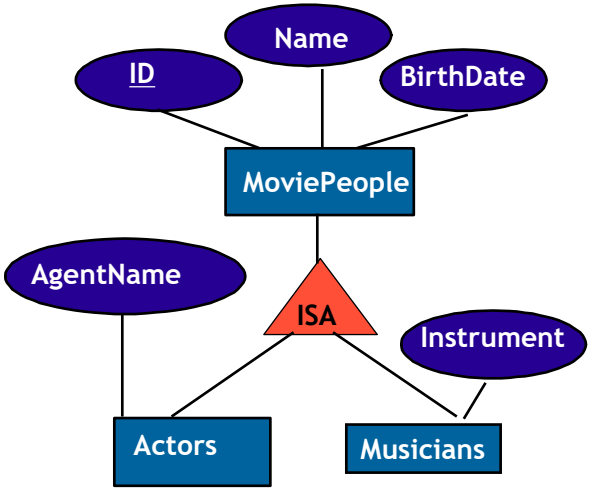
Method 1:have only one table with *all* attributes

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

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

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

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



- Lots of space needed for nulls
- Excellent method if subclasses do not have any new attributes and relationships

Method 2: 3 tables, remove excess attributes

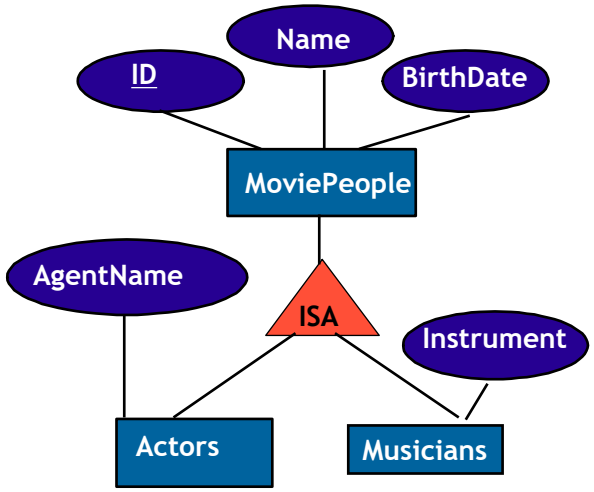
MoviePeople(ID, Name, BirthDate, ~~AgentName, Instrument~~)

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

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

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

- 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

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

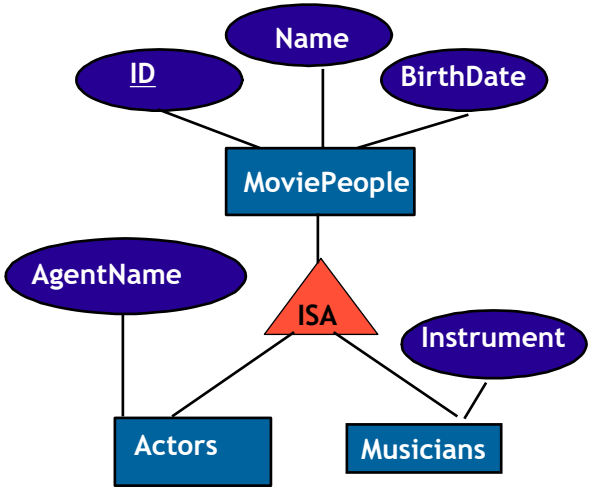
Actors(ID, Name, BirthDate, AgentName, ~~Instrument~~)

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

- 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?

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



Summary - ISA

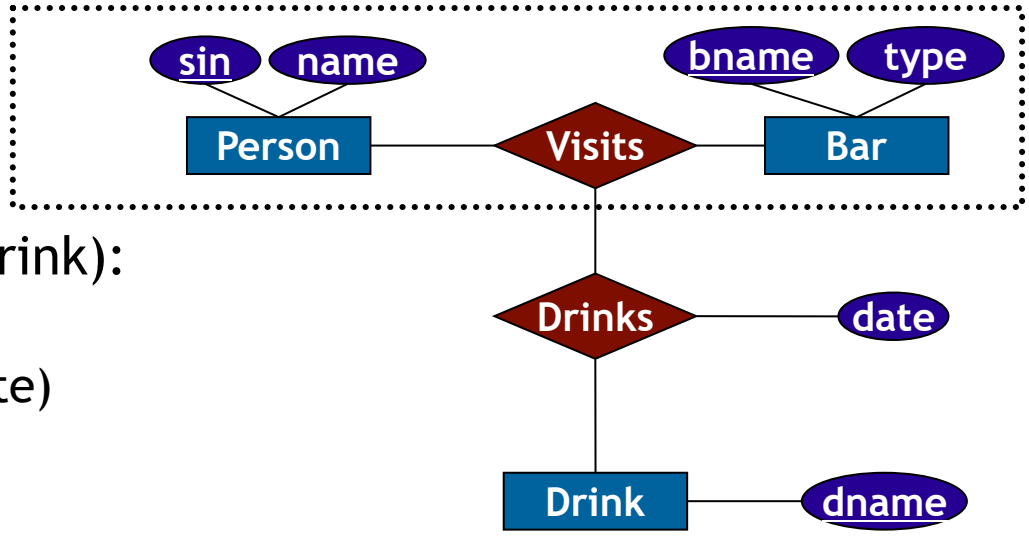
- If subclasses have no new attributes or relationships, or if they all have the same attributes and relationships
  - One table with all attributes

- By default, ISA has non-covering and disjoint constraints
  - If ISA is covering and disjoint, subclasses have different attributes or different relationships and there is no need to keep the superclass
  - No table for the superclass
  - One table for each subclass with all attributes

- Otherwise (if ISA is non-covering or overlapping and subclasses have new and different attributes….)
  - A table for the superclass and one for each subclass

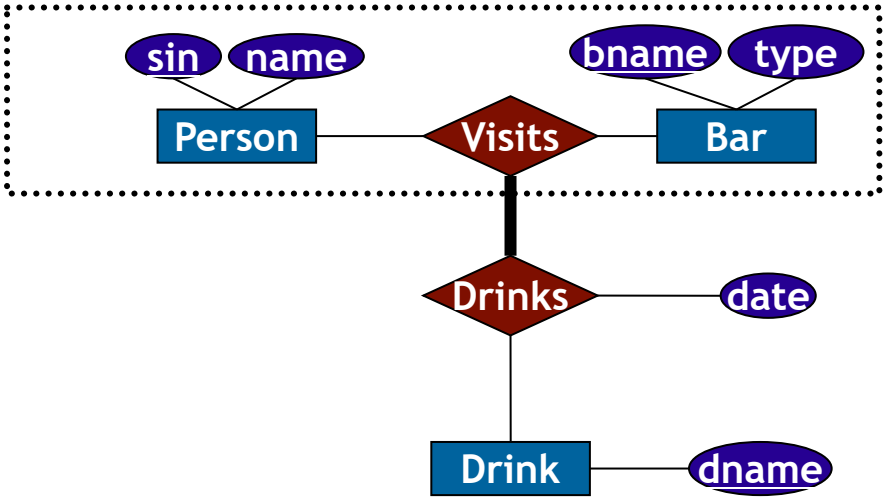
Translating Aggregation

- Use standard mapping of relationship sets
- Tables for our example (other than Person, Bar, and Drink):
  - Visits(**sin**, **bname**)
  - Drinks(**sin**, **bname**, **dname**, date)



Aggregation – Special Case

- There is a case where no table is required for an aggregate entity
  - Even where there are **no** cardinality constraints
    - If there is total participation between the aggregate entity and its relationship, and
    - If the aggregate entity does not have any descriptive attributes



If Visits is total on Drinks and Visits has no descriptive attributes we could keep only the Drinks

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