CPSC 304
Introduction to Database Systems

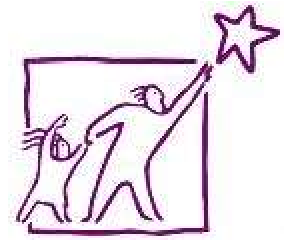
The Relational Model

Textbook Reference
Database Management Systems: 3.1 - 3.5

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.

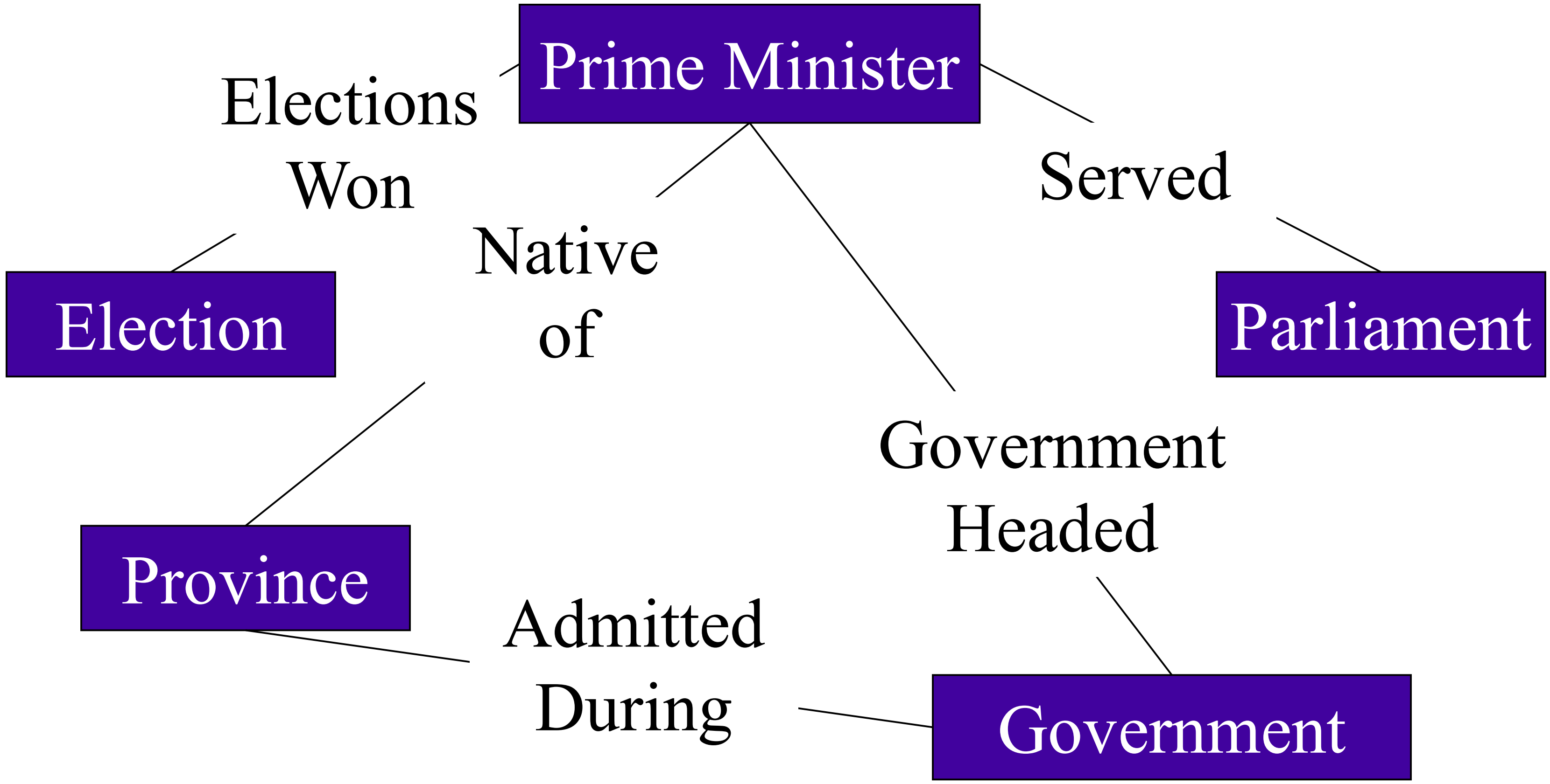
What do we want out of our logical schema representation?

- 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

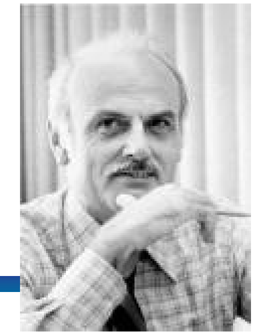


Example IMS (Hierarchical) query: Print the names of all the provinces admitted during a Liberal Government

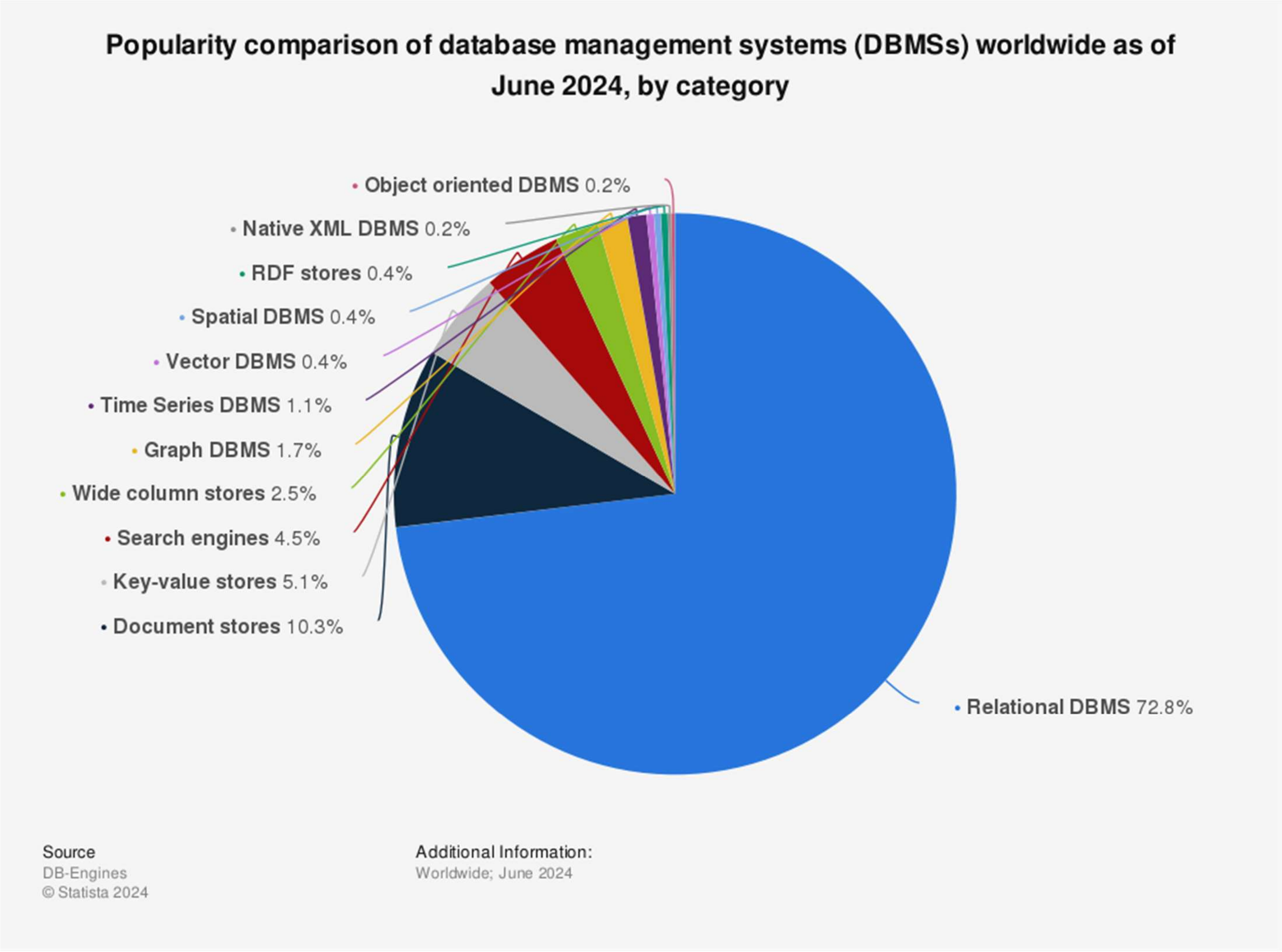
```
DLITPLI:PROCEDURE (QUERY_PCB) OPTIONS (MAIN);

 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 PM_IO_AREA CHAR(65),
  1 PRIME MINISTER DEFINED PM_IO_AREA,
  2 PM_NUMBER CHAR(4),
  2 PM_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 PM_SSA STATIC UNALIGNED,
  2 SEGMENT_NAME CHAR(8) INIT('PM '),
  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,PM_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,PM_IO_AREA,PRIME_MINISTER_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!

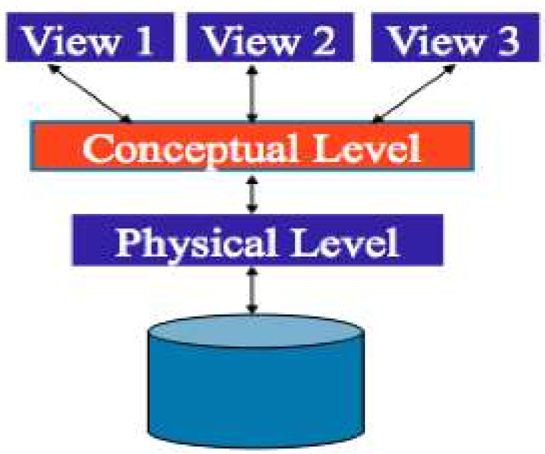


- Introduced by Edgar Codd (IBM) in 1970
- Most widely used model today.
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- Competitor: object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: *object-relational model*
    - Informix Universal Server, UniSQL, O2, Oracle,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


Popularity comparison of database management systems (DBMSs) worldwide as of June 2024, by category
Object oriented DBMS 0.2%
Native XML DBMS 0.2%
RDF stores 0.4%
Spatial DBMS 0.4%
Vector DBMS 0.4%
Time Series DBMS 1.1%
Graph DBMS 1.7%
Wide column stores 2.5%
Search engines 4.5%
Key-value stores 5.1%
Document stores 10.3%
Relational DBMS 72.8%
Source
DB-Engines
© Statista 2024
Additional Information:
Worldwide; June 2024

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*: char[20], *name*: char[20], *address*: char[20], *phone*: char[20], *major*: char[20]).
  - ***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

relation name

attribute, column 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*: char[20], *name*: char[20], *address*: char[20], *phone*: char[20], *major*: char[20])

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: char[20], address: char[20], phone: char[20], major: char[20])

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

- SQL was **NOT** the first relational 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)

Fun talk on history of RDBMSs by Don Chamberlain at SIGMOD 2023

A peek at the SQL Query Language

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 * "

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))
```

- 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)

"Integrity is doing the right thing, even when no one is watching" - CS 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.
  - 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
- One or more attributes in a relation form a ***key*** (or ***candidate key***) for a relation, where S is the set of all attributes in the key, if :
  1. No distinct tuples can have the same values for all attributes in the key, and
  2. No subset of S is itself a key.

  (If such a subset exists, then S is only 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))
```

Quick Detour: Keys

Student

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…

Candidate keys:

- sid
- CWL
- SIN

Quick Detour: Keys

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…
4	bPPuff	222	Blossom	Education	18	

In this strange school, every student within the same major must have a unique name.

E.g., There is a student called Blossom in music so the music major can never admit another student named Blossom.

Quick Detour: Keys (primary keys)

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…
4	bPPuff	222	Blossom	Education	18	

<u>Candidate keys</u>:

- sid
- CWL
- SIN
- {name, major}

Pick a candidate key to be your primary key

Quick Detour: Keys (Superkeys)

sid	CWL	SIN	name	major	age	…
1	bpuff1	123	Blossom	Music	18	…
2	bpuff2	234	Buttercup	Physics	18	…
3	bpuff3	456	Bubbles	Education	18	…
4	bPPuff	222	Blossom	Education	18	

<u>Candidate keys</u>:

- sid
- CWL
- SIN
- {name, major}

What is a superkey? A key plus zero or more additional attributes. Examples (not exhaustive) All the candidate keys plus:

- {sid, name}
- {CWL, major}
- {name, major, age}

Just a preview. I promise we'll do more superkeys later.

Keys Constraints in SQL

- A **PRIMARY KEY** constraint specifies a table's primary key
  - values for primary key must be unique
  - a primary key attributes 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) )
```

Keys Constraints in SQL (cont')

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

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

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

Let's look at students and grades again...

Student

sid	name	…
1	Blossom	…
2	Buttercup	…
3	Bubbles	…
4	Blossom	

Grade

sid	dept	cnum	grade
2	CPSC	304	90
2	MATH	221	90
2	EPSE	223	90
1	MUSC	103	90

Do we want tuples in Grade to ever refer to a sid that does not exist?

NO!

Idea: Check the Student table each time we insert a tuple into the Grade table to see if sid exists.

Enforcing Referential Integrity

Student

sid	name	…
1	Blossom	…
2	Buttercup	…
3	Bubbles	…
4	Blossom	

Grade

sid	dept	cnum	grade
2	CPSC	304	90
2	MATH	221	90
2	EPSE	223	90
1	MUSC	103	90

A foreign key is a set of attributes in one relation (e.g., Grades.sid) used to 'reference' a tuple in another relation (e.g., Students.sid).

Enforcing Referential Integrity

Grade

sid	dept	cnum	grade
2	CPSC	304	90
2	MATH	221	90
2	EPSE	223	90
1	MUSC	103	90

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

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,
    FOREIGN KEY (dept, course#) REFERENCES Course(dept, cnum))
```

Sometimes you can not specify which attributes are referenced, but in this case they are needed. Never hurts to include them!

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	….	…	…

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(sid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
  FOREIGN KEY (dept, course#)
      REFERENCES
        Course(dept,course#)
        ON DELETE SET DEFAULT
        ON UPDATE CASCADE );
```

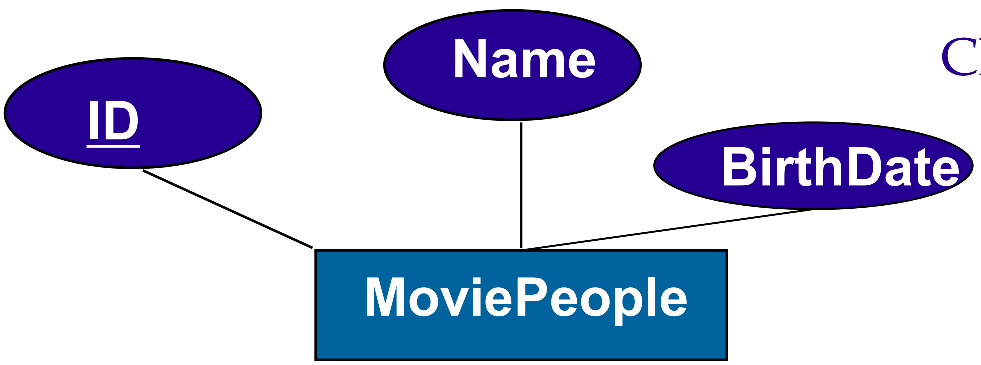
Where do ICs Come From?

- 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

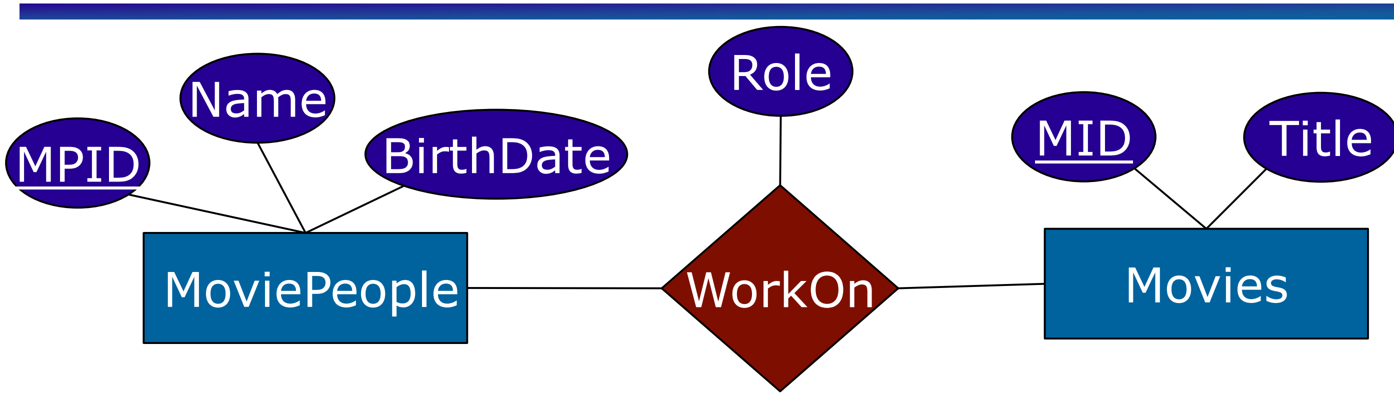
Entity sets to tables.

- 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

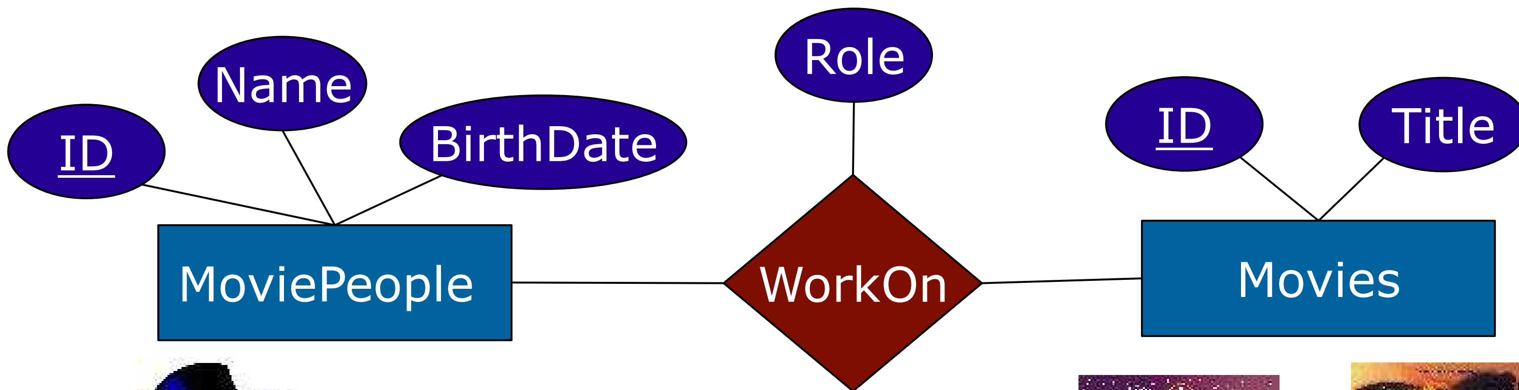


MPID and MID CANNOT be null

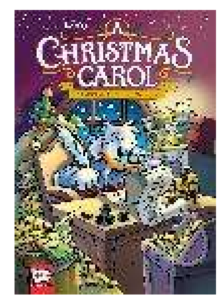
- A relationship set id is mapped to a single relation (table).
- Simple case: relationship has no constraints (i.e. many-to-many)
- In this case, attributes of the table must include:
  - Keys for each participating entity set as foreign keys.
    - This is a *key* for the relation.
  - All descriptive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

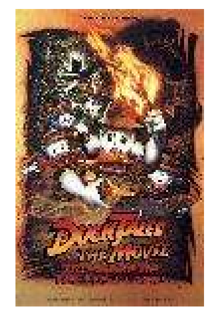
```
CREATE TABLE WorkOn(
  MPID     CHAR(11),
  MID    INTEGER,
  Role    CHAR(20),
  PRIMARY KEY (MPID, MID),
  FOREIGN KEY (MPID)
     REFERENCES MoviePeople,
 FOREIGN KEY (MID)
     REFERENCES Movies)
```

Example: Many to Many Relationships

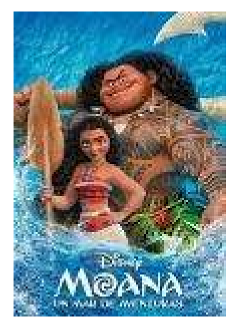


Scrooge McDuck has worked on two movies

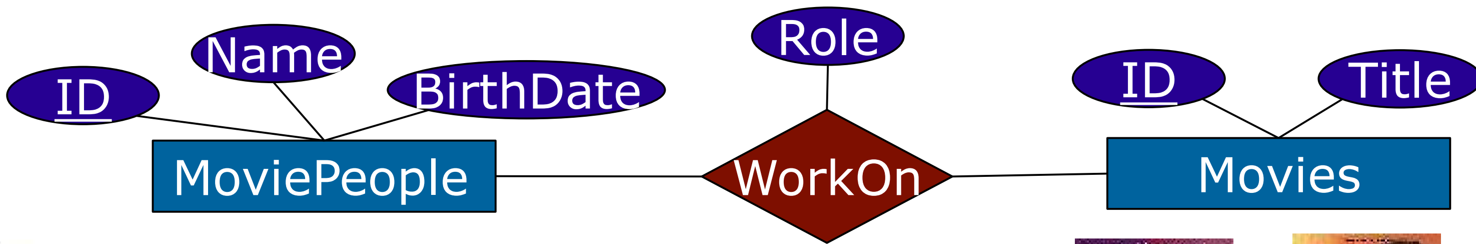




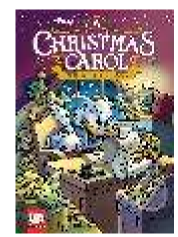
Pua has worked on one movie (Moana)

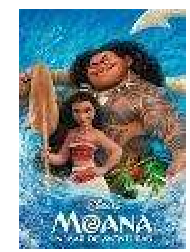


Example: Many to Many Relationships



Can we reduce redundancy by combining these tables in any way?





MoviePeople

ID	Name	Birthdate
1	Scrooge	…
2	Pua	…

WorkOn

MPID	**MID**	Role
1	1	…
1	2	…
2	3	…

Movies

ID	Title
1	Ducktales
2	A Christmas Carol
3	Moana

Example: Many to Many Relationships

MoviePeople

ID	Name	Birthdate
1	Scrooge	…
2	Pua	…

Movies

ID	Title
1	Ducktales
2	A Christmas Carol
3	Moana

WorkOn

MPID	**MID**	Role
1	1	…
1	2	…
2	3	…

Can we integrate the information in WorkOn into MoviePeople?

Example: Many to Many Relationships

MoviePeople

ID	Name	Birthdate	MID
1	Scrooge	…	
2	Pua	…	

Movies

ID	Title
1	Ducktales
2	A Christmas Carol
3	Moana

WorkOn

MPID	MID	Role
1	1	…
1	2	…
2	3	…

Do we put MID 1 or MID 2? We can't have both in the column (i.e., we can't store a list there)

Not much we can do in terms of reducing tables

Example: Many to Many Relationships

MoviePeople

ID	Name	Birthdate
1	Scrooge	…
2	Pua	…

Movies

ID	Title	MPID
1	Ducktales	
2	A Christmas Carol	
3	Moana	

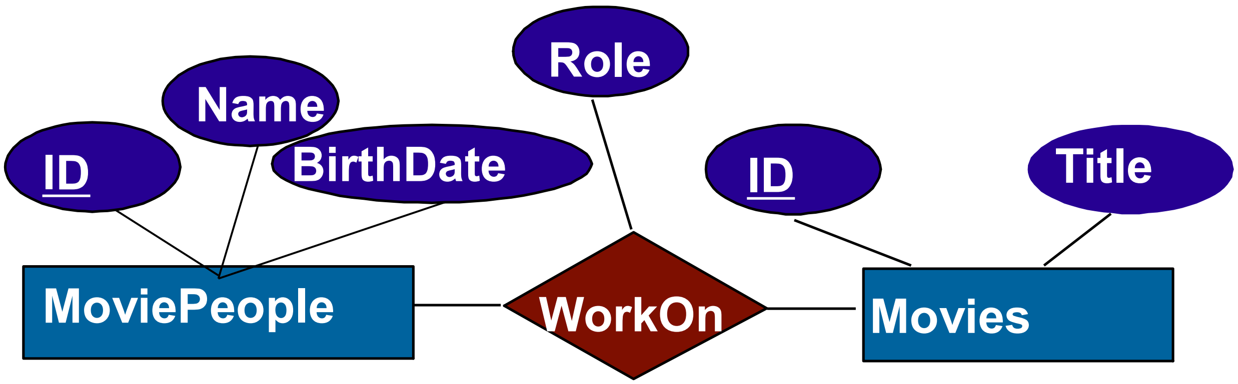
WorkOn

MPID	MID	Role
1	1	…
1	2	…
2	3	…

What if we have more than one person work on this movie? Same problem as before!

Not much we can do in terms of reducing tables

Relationship Sets to Tables



MoviePeople

ID	Name	Birth Date
1		
2		

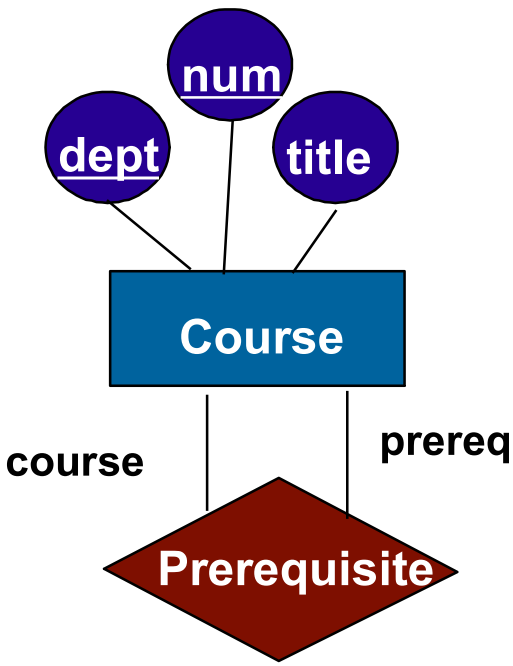
WorkOn

MoviePeople ID	Movie ID
1	1
2	1

Movies

ID	Title
1	
2	

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))
```

To motivate examples on upcoming slides, let’s talk about getting a PhD

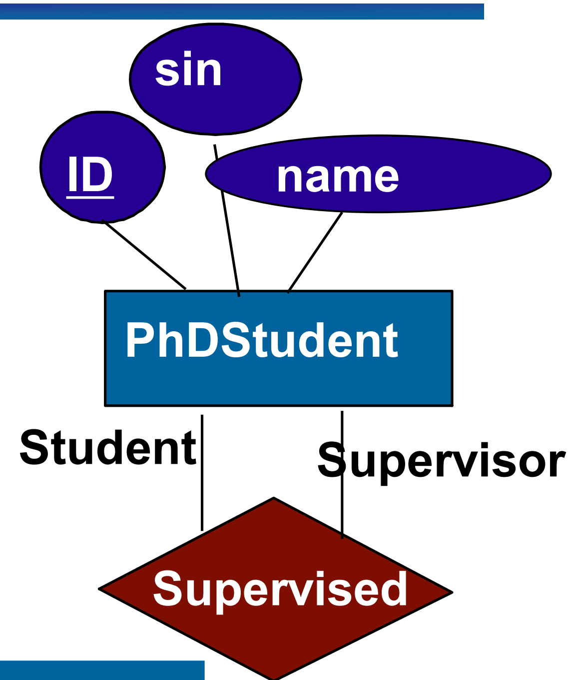
- PhD students all have to have advisors, all of whom also have had advisors (etc.)
- There exist databases where you can go back hundreds of years to see people’s academic lineage
- Out of curiosity, and to make this more interesting for you (you’re welcome), you can use https://www.genealogy.math.ndsu.nodak.edu/ to look this information up

Rachel's academic genealogy:

- Rachel Pottinger
- Phil Bernstein
- Catriel Beeri
- Eli Shamir
- Shmuel Agmon
- Szolem Mandelbroit
- Jacques Salomon Hadamard
- C Emile (Charles) Picard
- Gaston Darboux
- Michel Chasles
- Simeon Denis Poisson
- Joseph Louis Lagrange
- Leonhard Euler
- Johann Bernoulli
- Jacob Bernoulli
- Peter Werenfels
- Theodor Zwinger, Jr.
- Sebastian Beck
- Johann Jacob Grynaeus
- Simon Sulzer
- Wolfgang Fabricius Capito
- Desiderious Erasmus
- Jan Standonck

One possible **partial** representation of this data is

```
CREATE TABLE PhDStudent(
    id INT,
    sin INT,
    name CHAR(20),
    advisorID INT);
```



id	sin	name	AdvisorID
1	Null	Jan Standonck	Null
2	Null	Desiderious Erasmus	1

Self Referencing Relations

Goal: have Advisor be foreign key reference for same table PhDstudent.

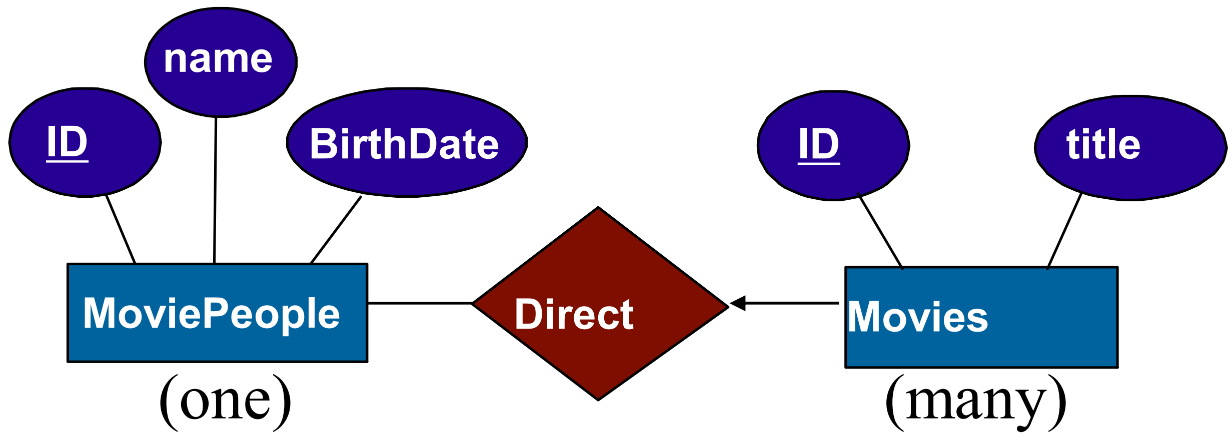
id	sin	name	AdvisorID
1	Null	Jan Standonck	Null
2	Null	Desiderious Erasmus	1

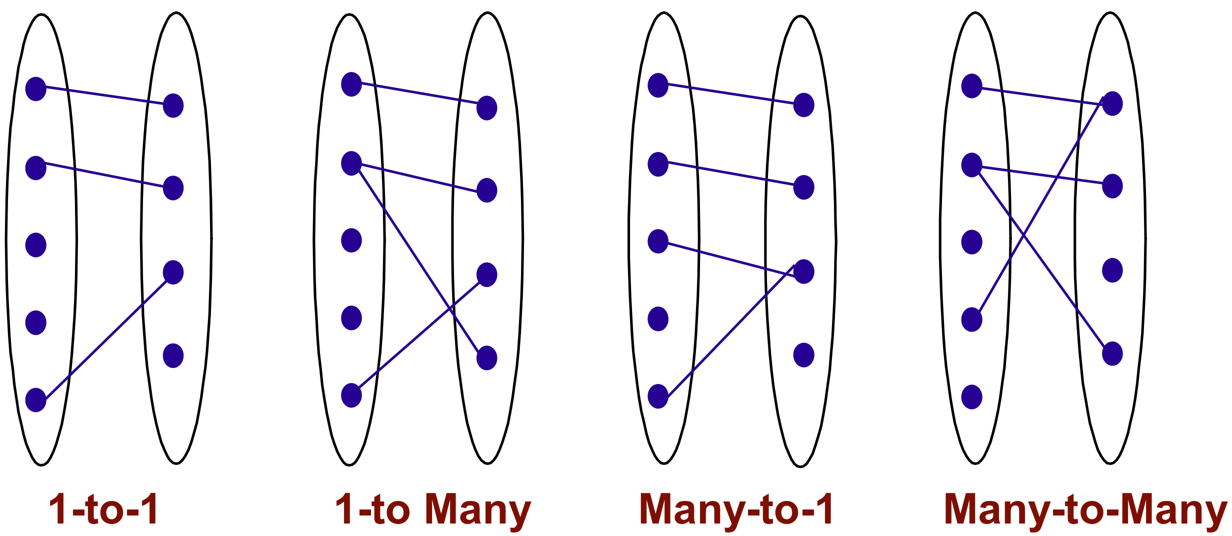
Could foreign key be null?

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.

Review: Key Constraints

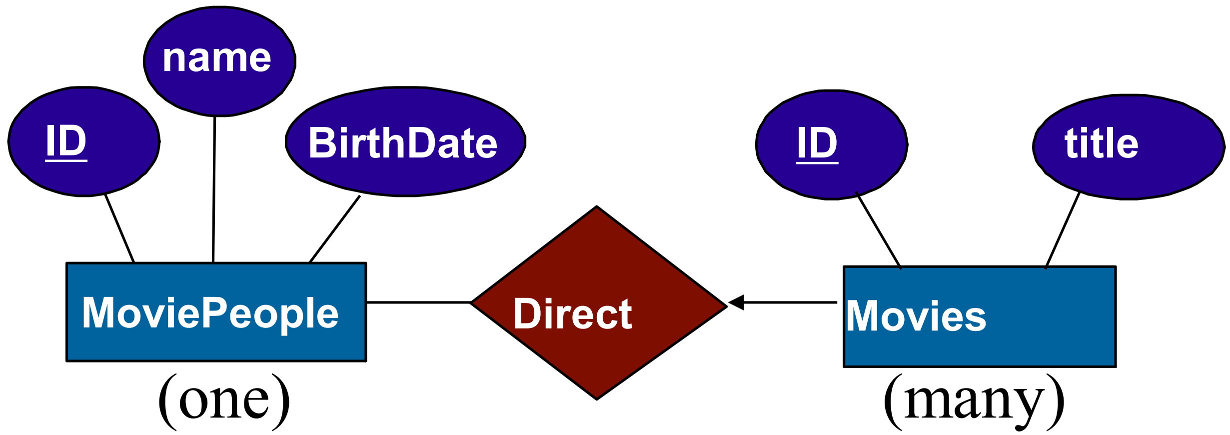
- Each movie has at most one director, according to the *key constraint* on Direct.





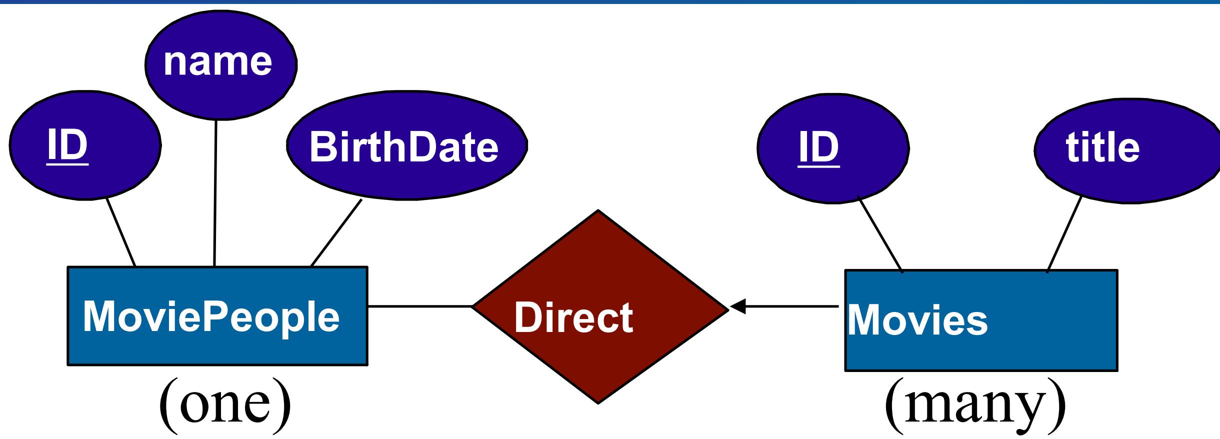
Translation to relational model?

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?

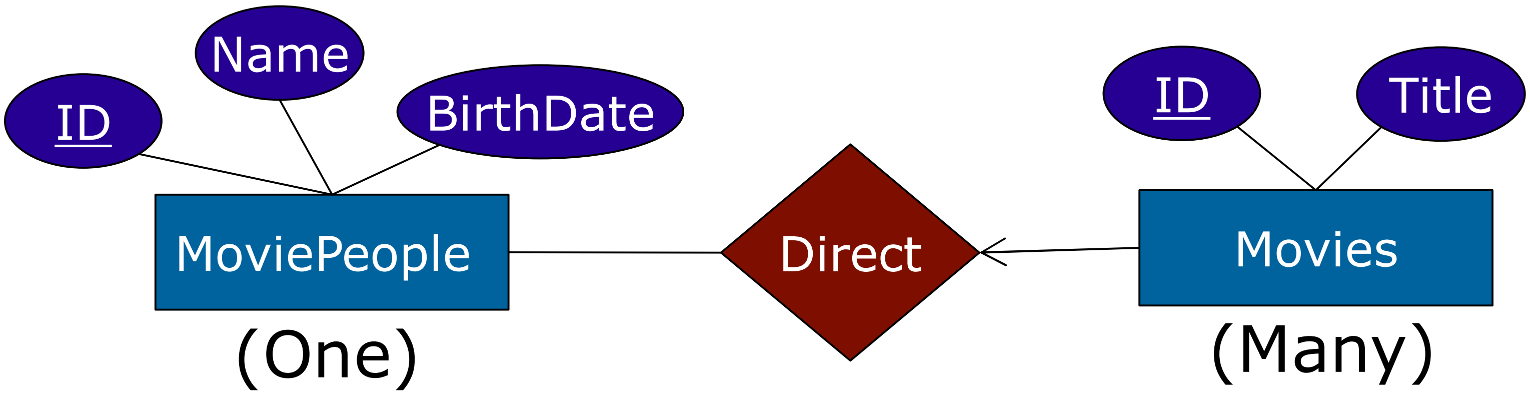
For the one to many case, combine the many side with the relationship



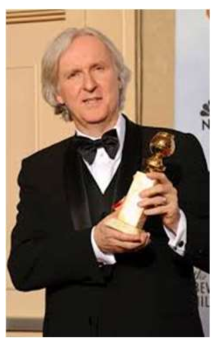
I know who the MoviePerson is if I know the movie!

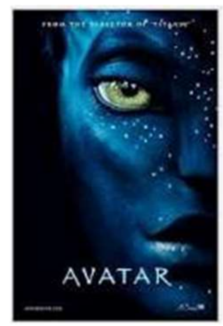
I can remove some redundancy in my design.

For the one to many case, combine the many side with the relationship

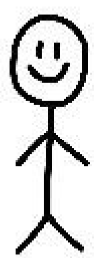


James Cameron





You



The Life of a CS Student

For the one to many case, combine the many side with the relationship

MoviePeople

ID	Name	Birthdate
1	James Cameron	…
2	You	…

Movies

ID	Title
1	Avatar
2	Titanic
3	The Life of a CS Student

Direct

MPID	MID
1	1
1	2
2	3

Can we reduce redundancy by integrating Direct into MoviePeople?

For the one to many case, combine the many side with the relationship

MoviePeople

ID	Name	Birthdate	~~Directed-MID~~
1	James Cameron	…	
2	You	…	

Movies

ID	Title
1	Avatar
2	Titanic
3	The Life of a CS Student

Direct

MPID	MID
1	1
1	2
2	3

Same issue as before.
We can't have
multiple values here.

For the one to many case, combine the many side with the relationship

MoviePeople

ID	Name	Birthdate
1	James Cameron	…
2	You	…

Movies

ID	Title
1	Avatar
2	Titanic
3	The Life of a CS Student

Direct

MPID	MID
1	1
1	2
2	3

Can we integrate Direct into Movies?

For the one to many case, combine the many side with the relationship

MoviePeople

ID	Name	Birthdate
1	James Cameron	…
2	You	…

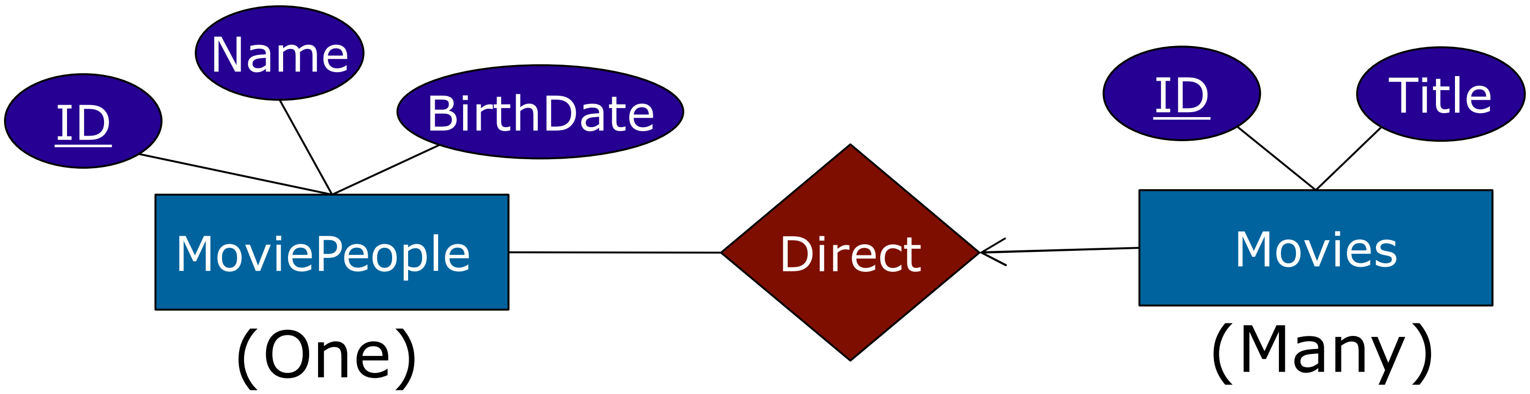
Movies

ID	Title	**Director-MPID**
1	Avatar	1
2	Titanic	1
3	The Life of a CS Student	2

Will there ever be two different directors for the same movie?

No! (because of our one to many relationship)

For the one to many case, combine the many side with the relationship



MoviePeople

ID	Name	Birthdate
1	James Cameron	…
2	You	…

Movies

ID	Title	**Director-MPID**
1	Avatar	1
2	Titanic	1
3	The Life of a CS Student	2

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.

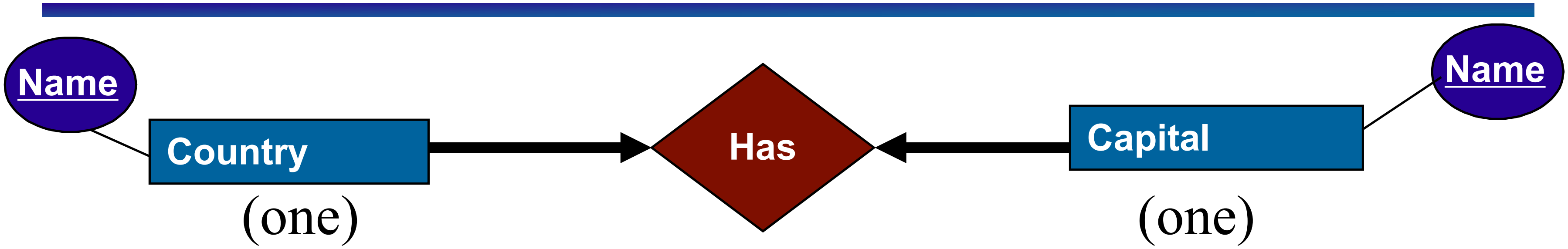
```
CREATE TABL[illegible]ect(
  MPID    CHAR[illegible]
  MID   INTEGER[illegible]
  PRIMARY KEY  ([illegible]
  FOREIGN KEY[illegible]FERENCES
MoviePeople,
  FOREIGN K[illegible]IID) [illegible]RENCES Movies)
```

- Method 2 (better)
  - Since each movie has a unique director, we can **combine Direct and Movies into one table.**
  - Create another table for MoviePeople
  - Must have on delete and on update in this case!

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

Oracle does not support “on update” – still use method 2

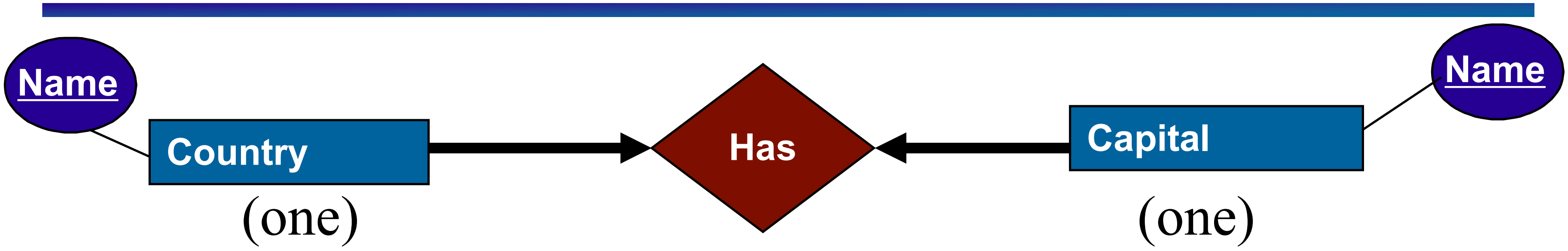
Details, details



Assume you went with Country(<u>coName</u>, 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(
        country-name CHAR(20)  PRIMARY KEY,
        capital-name CHAR(20),
        UNIQUE capital-name ← needed for one-to-one constraint
)
```

Details, details

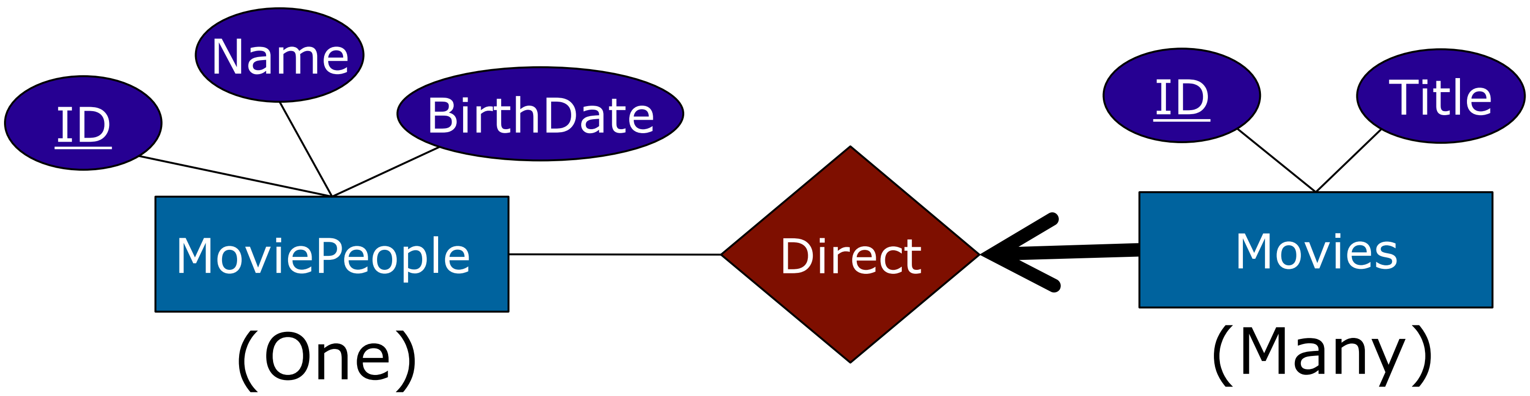


Assume Country(coName, caName) and all attributes of type Char(20) and we're not creating a separate relation for Capital.

```
CREATE TABLE Country(
        country-name CHAR(20)  PRIMARY KEY,
        capital-name CHAR(20),
        UNIQUE capital-name); ← needed for one-to-one constraint)
```

coName	caName
Canada	Ottawa
United States	Ottawa
Mexico	Ottawa

Translating Participation Constraints



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

Participation Constraints in SQL

- Using method 2 (add Directs relation in the Movie table), we can capture participation constraints by
  - ensuring that each **MID** is associated with a **MPID** that is not null
  - not allowing deletion of a director before the director is replaced

```
CREATE TABLE  Directed_Movie(
  MID        INTEGER,
  title      CHAR(20),
  MPID          CHAR(11) NOT NULL,
  PRIMARY KEY  (MID),
  FOREIGN KEY (MPID) 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

MoviePeople

<u>ID</u>	Name	Birthdate
1	Robet Pattinson	1986/05/13
2	James Cameron	1954/08/16

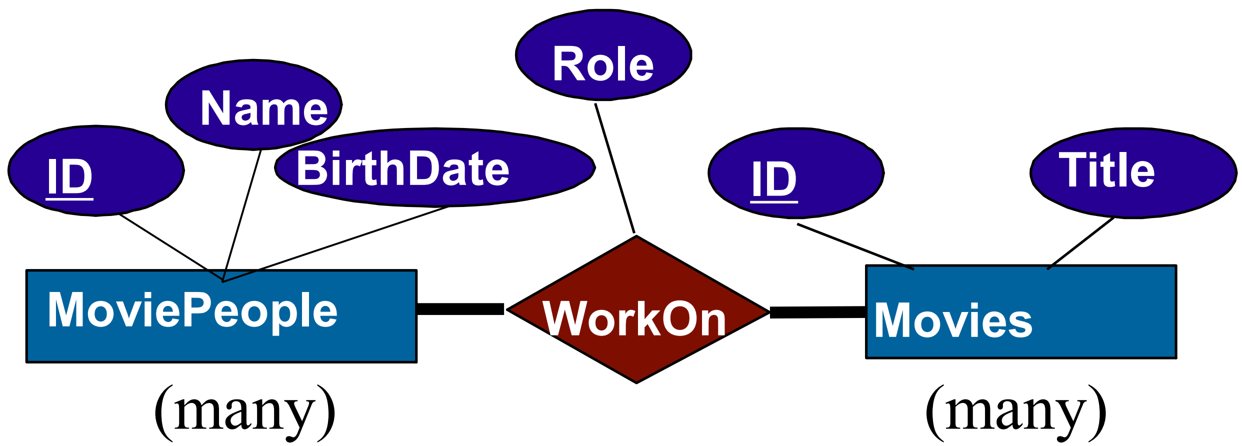
DirectedMovie

MID	Title	MPID is not null
1	Avatar	2
~~2~~	~~The Dark Knight~~	~~null~~

Not legal!→

Because we can't have a tuple in the DirectedMovie table with a null MPID, we can't insert any movies without directors. Therefore, all movies must participate in the Direct relationship

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 (covered later in the SQL module)

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

MoviePeople

ID	Name	Birthdate
1	Robert Pattinson	1986/05/13
2	James Cameron	1954/08/16

Movie

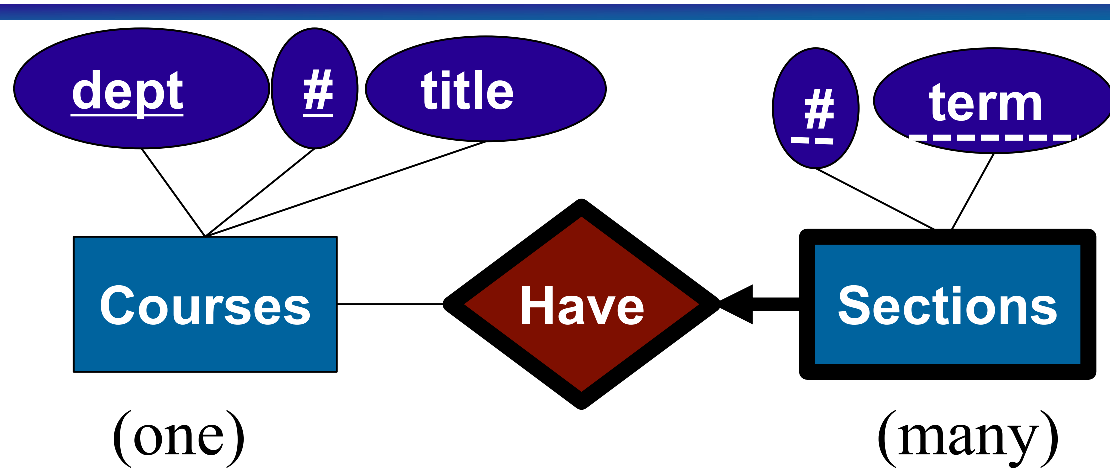
ID	Title
1	Oppenheimer
2	Avatar

Workon

MPID	MID	Role
2	2	Director

No nulls, but Robert Pattinson does not work on a movie and Oppenheimer has no one working on it

Translating Weak Entity Sets



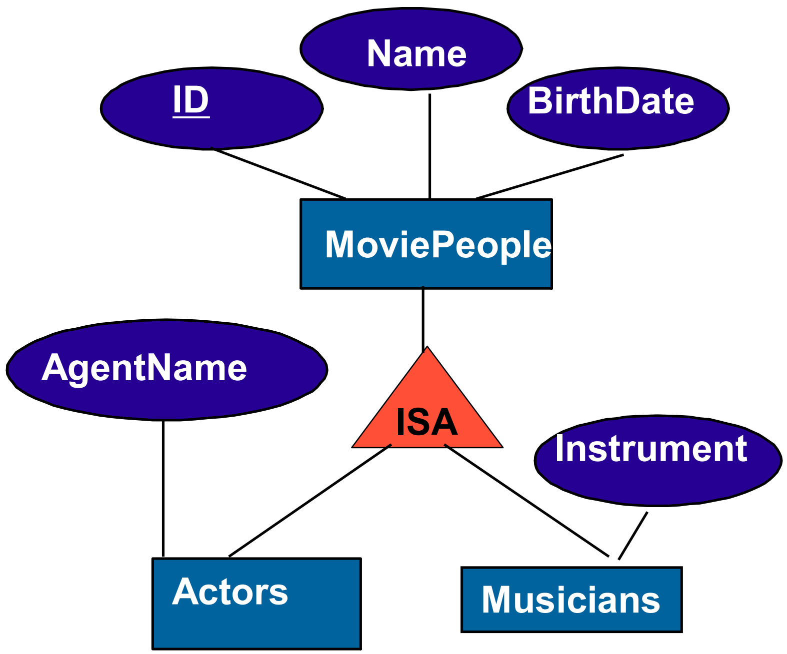
- A ***weak entity*** is identified by considering the primary key of the *owner* (strong) entity.
  - Owner entity set and weak entity set participate in a one-to-many identifying relationship set.
  - Weak entity set has total participation.
- What is the best way to translate it?

Translating Weak Entity Sets(cont')

- Weak entity set and its identifying relationship set are translated into a single table (like many to one anyway)
  - 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.

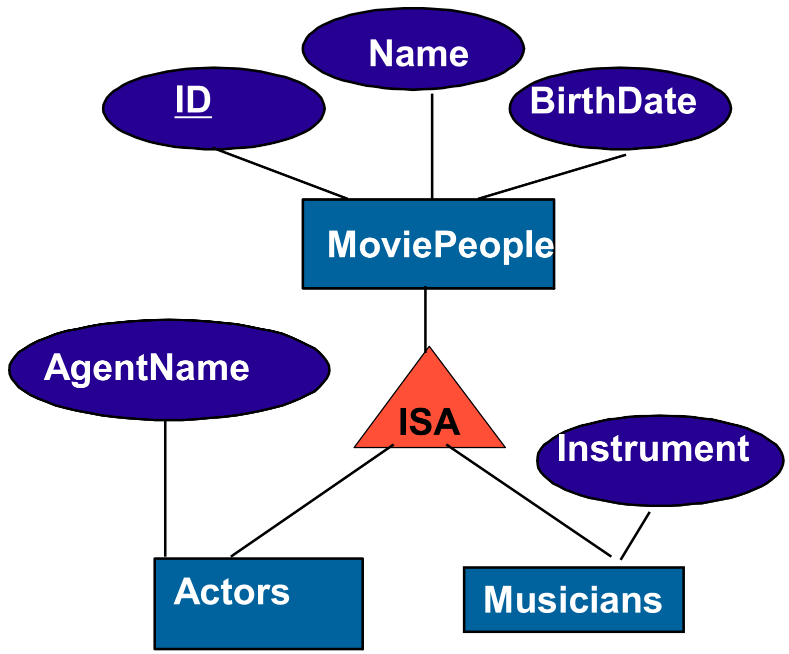
```
CREATE TABLE  Course_Section (
  dept            CHAR(4),
  course_num INTEGER,
  section_num INTEGER,
  term            CHAR(6)
  PRIMARY KEY  (dept, course_num, section_num, term),
  FOREIGN KEY  (dept, course_num) REFERENCES
                  Courses(dept, num),
                  ON DELETE CASCADE)
```

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)



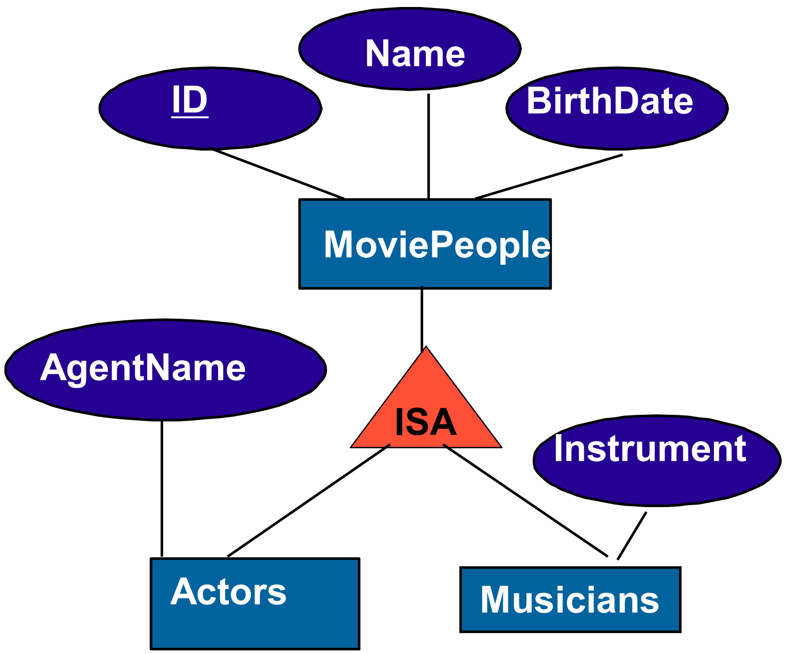
One table per entity. Each has *all* attributes:

MoviePeople(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 (not in book)



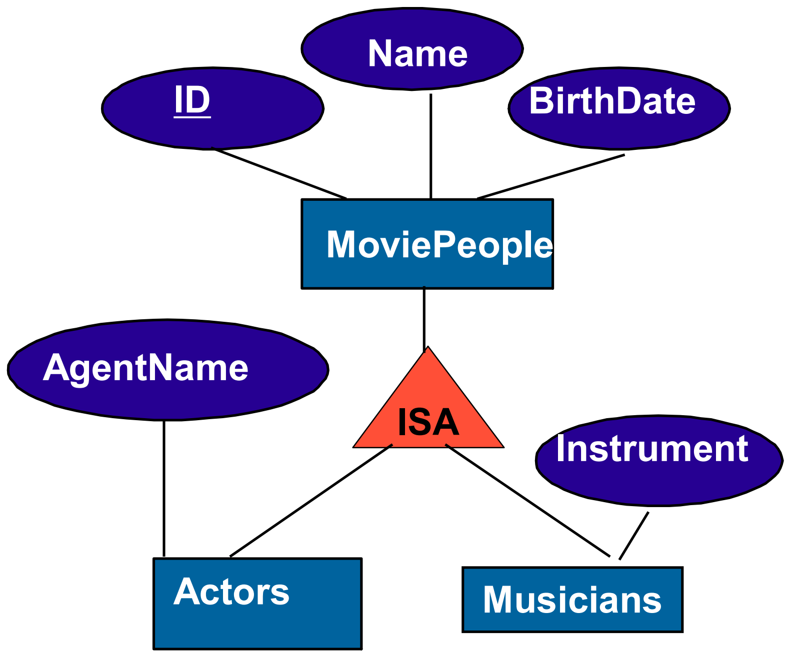
MoviePeople(ID, Name, BirthDate, AgentName, Instrument)

~~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

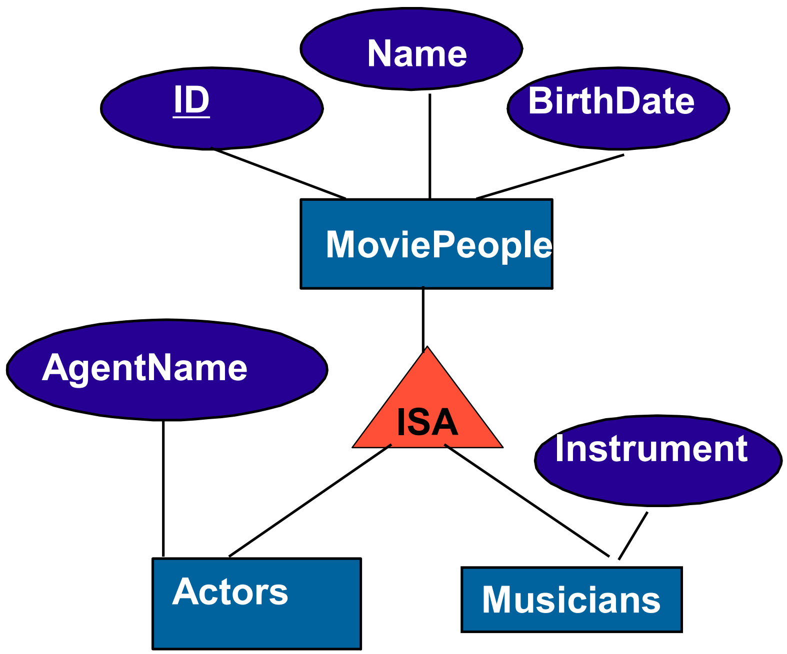
MoviePeople(<u>ID</u>, Name, BirthDate, ~~AgentName, Instrument~~)

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

Musicians(<u>**ID**</u>, ~~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

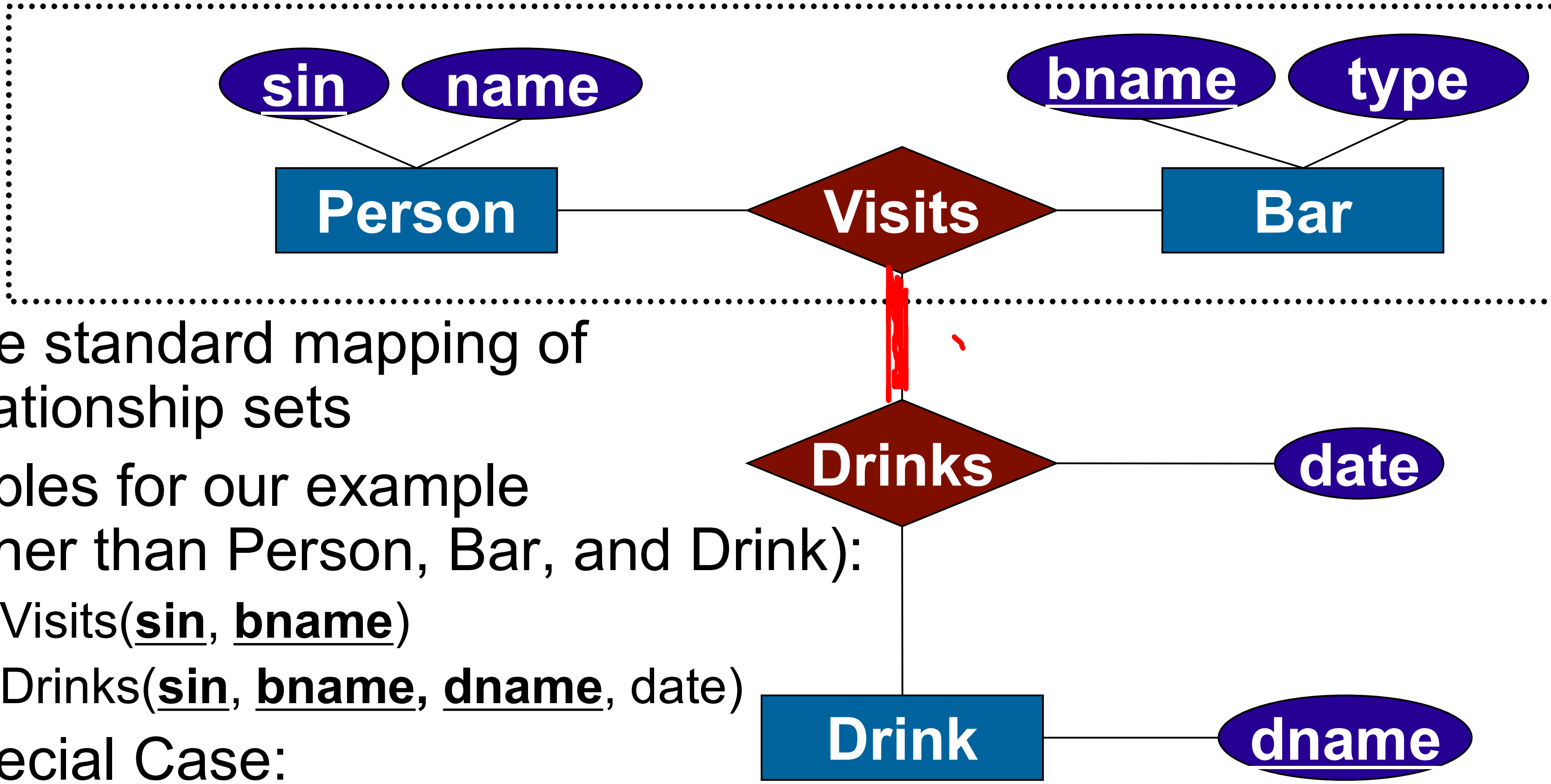
~~MoviePeople(ID, Name, BirthDate, AgentName, Instrument)~~

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).

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.