A Quick Introduction to Database design with E/R (extracurricular to MAS 201)

Data Structure: Relational Model

- **Relational Databases:** Schema + Data
- **Schema:**
  - collection of *tables* (also called *relations*)
  - each table has a set of *attributes*
  - no repeating relation names, no repeating attributes in one table
- **Data** (also called *instance*):
  - set of *tuples*
  - tuples have one *value* for each attribute

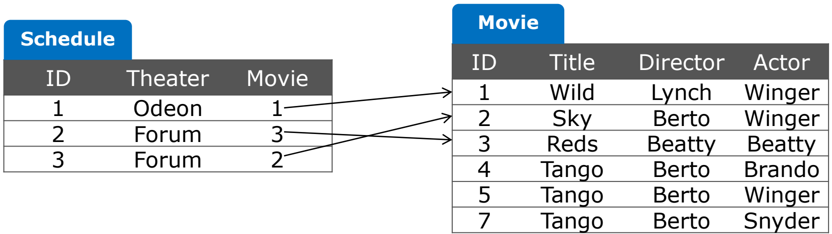
Movie

ID	Title	Director	Actor
1	Wild	Lynch	Winger
2	Sky	Berto	Winger
3	Reds	Beatty	Beatty
4	Tango	Berto	Brando
5	Tango	Berto	Winger
7	Tango	Berto	Snyder

Schedule

ID	Theater	Movie
1	Odeon	1
2	Forum	3
3	Forum	2

Data Structure: Primary Keys; Foreign Keys are value-based pointers



Schedule

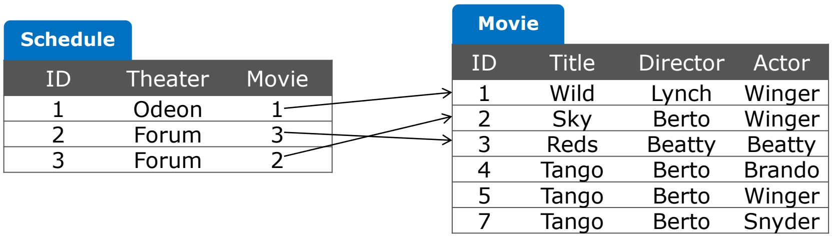
ID	Theater	Movie
1	Odeon	1
2	Forum	3
3	Forum	2

Movie

ID	Title	Director	Actor
1	Wild	Lynch	Winger
2	Sky	Berto	Winger
3	Reds	Beatty	Beatty
4	Tango	Berto	Brando
5	Tango	Berto	Winger
7	Tango	Berto	Snyder

- "`ID` is *primary key* of `Schedule`" => its value is unique in `Schedule.ID`
- "`Schedule.Movie` is foreign key (referring) to `Movie.ID`" means every `Movie` value of `Schedule` also appears as `Movie.ID`
- Intuitively, Schedule.Movie operates as pointer to Movie(s)

Schema design has its own intricacies



Schedule

ID	Theater	Movie
1	Odeon	1
2	Forum	3
3	Forum	2

Movie

ID	Title	Director	Actor
1	Wild	Lynch	Winger
2	Sky	Berto	Winger
3	Reds	Beatty	Beatty
4	Tango	Berto	Brando
5	Tango	Berto	Winger
7	Tango	Berto	Snyder

- This is a bad schema design!
- Problems
  - Change the name of a theater
  - Change the name of a movie's director
  - What about theaters that play no movie?

How to Design a Database and Avoid Bad Decisions

- With experience…
  - sweat, tears, etc etc
- Learning the normalization rules of database design
  - a well-developed mathematical theory about how to fix step by step a “bad” schema
- Think **entities and relationships** – then translate to relations
  - A practically useful way to come up with the good schema

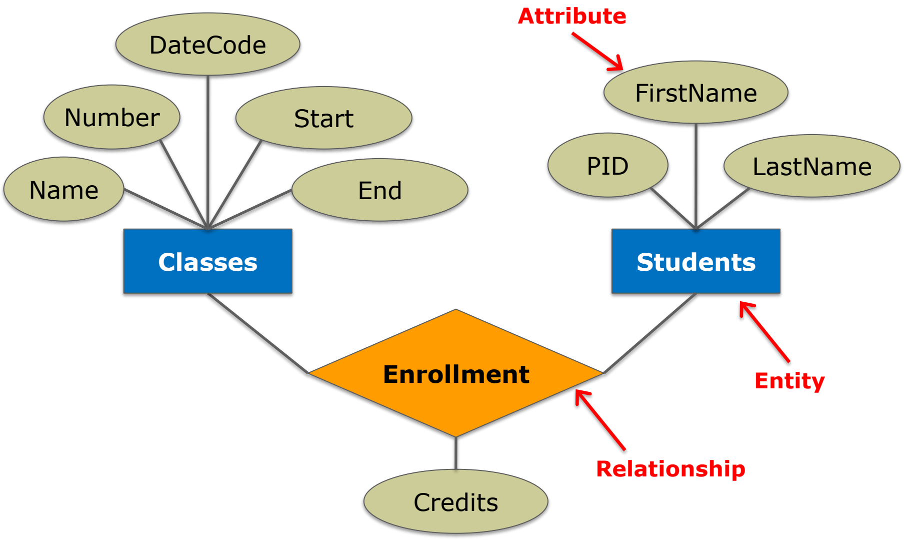
Data Structure: Relational Model

Example Problem:

- Represent the students and Spring classes of the CSE department, including the enrollment of students in classes.
- Students have pid, first name and last name.
- Classes have a name, a number, date code (TR, MW, MWF) and start/end time.
- A student enrolls for a number of credits in a class.

Solution:…

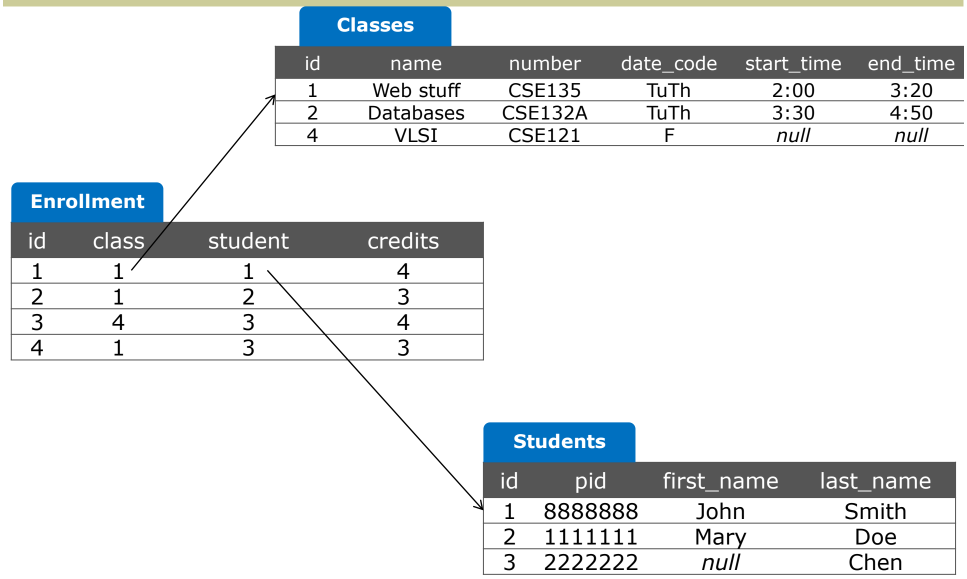
Example 1: E/R-Based Design



E/R → Relational Schema: Basic Translation

- For every entity
  - create corresponding table
  - For each attribute of the entity, add a corresponding attribute in the table
  - Include an ID attribute in the table even if not in E/R
- For every relationship
  - create corresponding table
  - For each attribute of the relationship, add a corresponding attribute in the table
  - For each referenced entity E_i include in the table a *required foreign key* attribute referencing ID of E_i

Sample relational database, per previous page’s algorithm



Classes

id	name	number	date_code	start_time	end_time
1	Web stuff	CSE135	TuTh	2:00	3:20
2	Databases	CSE132A	TuTh	3:30	4:50
4	VLSI	CSE121	F	*null*	*null*

Enrollment

id	class	student	credits
1	1	1	4
2	1	2	3
3	4	3	4
4	1	3	3

Students

id	pid	first_name	last_name
1	8888888	John	Smith
2	1111111	Mary	Doe
3	2222222	*null*	Chen

Declaration of schemas in SQL’s Data Definition Language

```
CREATE TABLE classes (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    number      TEXT,
    date_code   TEXT,
    start_time  TIME,
    end_time    TIME
)
CREATE TABLE students (
    ID          SERIAL PRIMARY KEY,
    pid         TEXT,
    first_name  TEXT,
    last_name   TEXT
)
CREATE TABLE enrollment (
    ID              SERIAL,
    class           INTEGER REFERENCES classes (ID) NOT NULL,
    student         INTEGER REFERENCES students (ID) NOT NULL,
    credits         INTEGER
)
```

If we had “ID **INTEGER** PRIMARY KEY” we would be responsible for coming up with ID values. **SERIAL** leads to a counter that automatically provides ID values upon insertion of new tuples

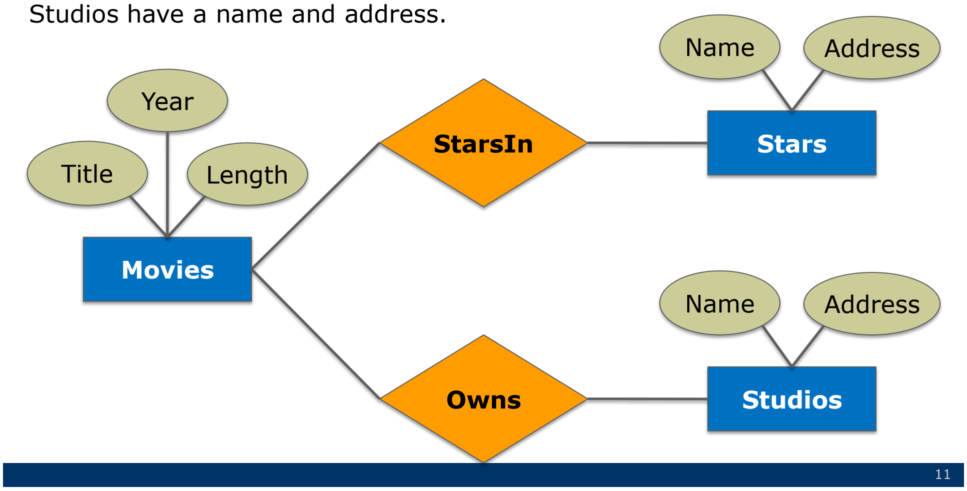
Changed name from “end” to “end_time” since “end” is reserved keyword

Foreign key declaration: Every value of **enrollment.class** must also appear as **classes.ID**

Declaration of “required” constraint: **enrollment.student** cannot be null (notice, it would make no sense to have an enrollment tuple without a student involved)

Example 2a

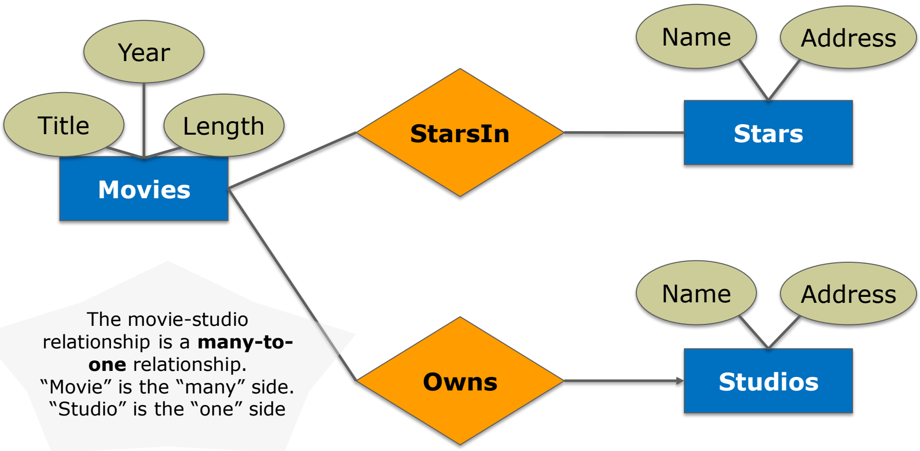
Movies have a title, a year of release and length (in minutes).
Actors have names and address.
Actors appear in movies.
A movie is (co-)owned by studios.
Studios have a name and address.



```
CREATE TABLE movies (
    ID          SERIAL PRIMARY KEY,
    title       TEXT,
    year        INTEGER,
    length      INTEGER,
)
CREATE TABLE stars (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    address     TEXT
)
CREATE TABLE studios (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    address     TEXT
)
CREATE TABLE starsin (
    ID              SERIAL,
    movie           INTEGER REFERENCES movies (ID) NOT NULL,
    star            INTEGER REFERENCES stars (ID) NOT NULL
)
CREATE TABLE ownership (
    ID              SERIAL,
    movie           INTEGER REFERENCES movies (ID) NOT NULL,
    owner           INTEGER REFERENCES studios (ID) NOT NULL
)
```

Example 2b: many-to-one relationship

Modification to Example 2a:
A movie is owned by **at most one** studio.



The movie-studio relationship is a **many-to-one** relationship.
"Movie" is the "many" side.
"Studio" is the "one" side

E/R→ Relational: Basic Translation revisited for many-to-one relationship

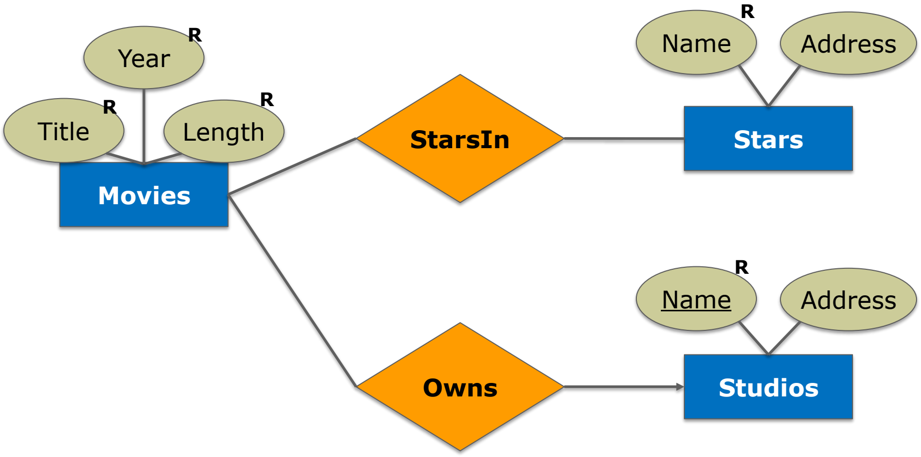
- For every entity, do the usual…
- For every **many-to-many** relationship, do the usual…
- For every **2-way many-to-one** relationship, where
  - E_m is the "many" side
  - E_o is the "one" side (pointed by the arrow)
  - **do not** create table, instead:
  - In the table corresponding to E_m add a (non-required) foreign key attribute referencing the ID of the table corresponding to E_o

```
CREATE TABLE movies (
    ID          SERIAL PRIMARY KEY,
    title       TEXT,
    year        INTEGER,
    length      INTEGER,
    owner       INTEGER  REFERENCES studios (ID)
)
CREATE TABLE stars (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    address     TEXT
)
CREATE TABLE studios (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    address     TEXT
)
CREATE TABLE starsin (
    ID              SERIAL,
    movie           INTEGER REFERENCES movies (ID) NOT NULL,
    star            INTEGER REFERENCES stars (ID) NOT NULL
)
```

Example 2c: Constraints: uniqueness; required attributes

In addition to Example 2b’s assumptions, let us also assume that:

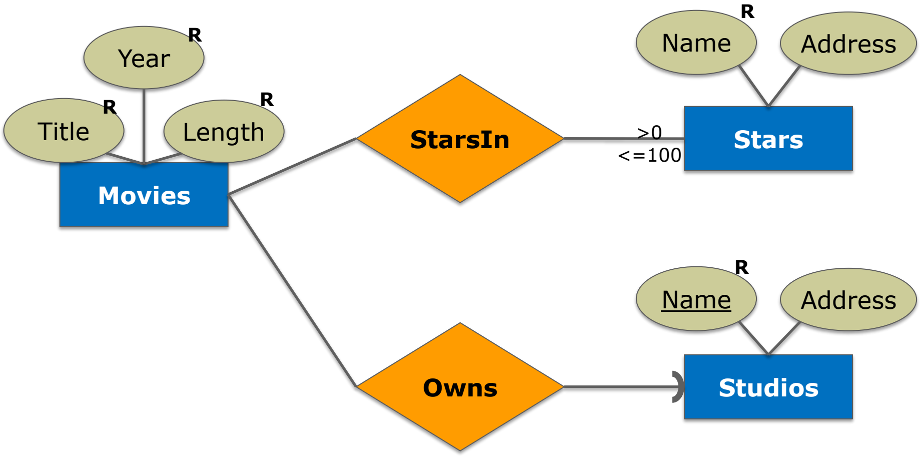
- title, year, length, star name and studio name are required attributes of the respective entities
  - default is that an attribute value may be **null**
- studios have unique names, i.e., no two studios may have the same name



Example 2d: Constraints: Required relationship; cardinality ranges

In addition to Example 2c's assumptions, let us also assume that:

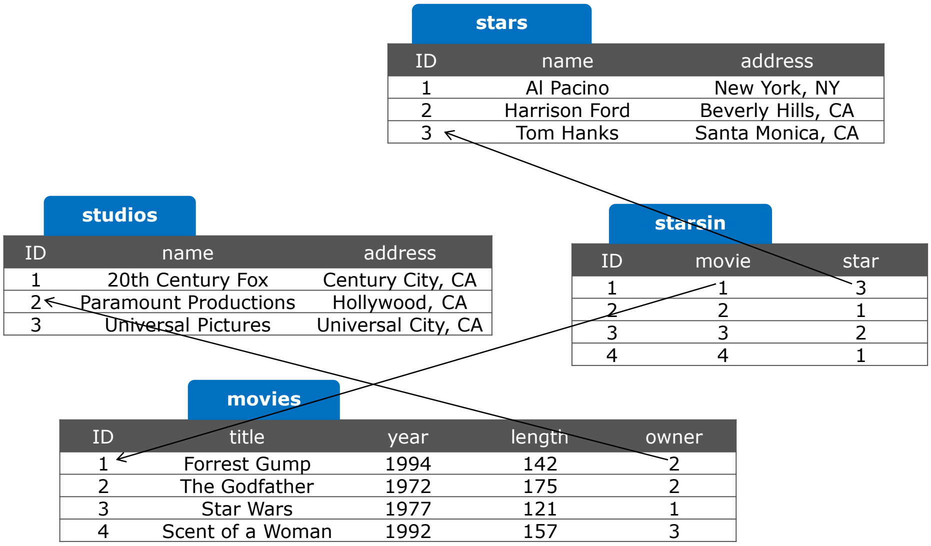
- a movie is owned by **exactly one** studio
  - so far we had not assumed that the owning studio has to be known (not null)
- a movie must have at least one actor and no more than 100



SQL Schema for Examples 2c, 2d

```
CREATE TABLE movies (
    ID          SERIAL PRIMARY KEY,
    title       TEXT NOT NULL,
    year        INTEGER NOT NULL,
    length      INTEGER NOT NULL,
    owner       INTEGER  REFERENCES studios (ID) NOT NULL
)
CREATE TABLE stars (
    ID          SERIAL PRIMARY KEY,
    name        TEXT NOT NULL,
    address     TEXT
)
CREATE TABLE studios (
    ID          SERIAL PRIMARY KEY,
    name        TEXT NOT NULL UNIQUE,
    address     TEXT
)
CREATE TABLE starsin (
    ID              SERIAL,
    movie           INTEGER REFERENCES movies (ID) NOT NULL,
    star            INTEGER REFERENCES stars (ID) NOT NULL
)
```

A sample database



stars

ID	name	address
1	Al Pacino	New York, NY
2	Harrison Ford	Beverly Hills, CA
3	Tom Hanks	Santa Monica, CA

studios

ID	name	address
1	20th Century Fox	Century City, CA
2	Paramount Productions	Hollywood, CA
3	Universal Pictures	Universal City, CA

starsin

ID	movie	star
1	1	3
2	2	1
3	3	2
4	4	1

movies

ID	title	year	length	owner
1	Forrest Gump	1994	142	2
2	The Godfather	1972	175	2
3	Star Wars	1977	121	1
4	Scent of a Woman	1992	157	3

Why do we want constraints? What happens when they are violated?

- Protect the database from erroneous data entry
- Prevent database states that are inconsistent with the rules of the business process you want to capture
- Whenever you attempt to change (insert, delete, update) the database in a way that violates a constraint the database will prevent the change
  - Try it out on the sample databases of the class page

Some constraints are not implemented by some SQL database systems

- Consider the cardinality constraint that a movie has between 1 and 100 actors.
- The SQL standard provides a way, named CHECK constraints, to declare such
  - its specifics will make more sense once we have seen SQL queries
- However, no open source database implements the CHECK constraints.

Vice versa: SQL allows some constraints that are not in plain E/R

Notable cases:

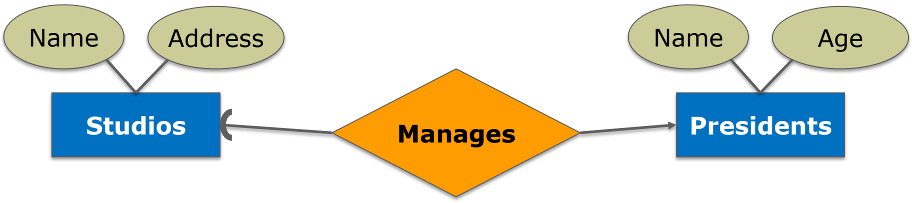
- Attribute value ranges
  - Example: Declare that the year of movies is after 1900
- Multi-attribute UNIQUE
  - Example: Declare that the (title, year) attribute value combination is unique

Added constraints of previous slide to schema of Example 2d

```
CREATE TABLE movies (
    ID          SERIAL PRIMARY KEY,
    title       TEXT NOT NULL,
    year        INTEGER NOT NULL CHECK (year > 1900),
    length      INTEGER NOT NULL,
    owner       INTEGER  REFERENCES studios (ID) NOT NULL,
    UNIQUE (title, year)
)
CREATE TABLE stars (
    ID          SERIAL PRIMARY KEY,
    name        TEXT NOT NULL,
    address     TEXT
)
CREATE TABLE studios (
    ID          SERIAL PRIMARY KEY,
    name        TEXT NOT NULL UNIQUE,
    address     TEXT
)
CREATE TABLE starsin (
    ID              SERIAL,
    movie           INTEGER REFERENCES movies (ID) NOT NULL,
    star            INTEGER REFERENCES stars (ID) NOT NULL
)
```

Example 3: one-to-one relationships

Assume that a president manages exactly one studio and
a studio may have at most one president.
Notice: a studio may not have a president but
in order to be a president one has to manage a studio.



1st candidate

```
CREATE TABLE presidents (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    age         INTEGER,
    manages     INTEGER  REFERENCES studios (ID) NOT NULL UNIQUE
)

CREATE TABLE studios (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    address     TEXT
)
```

Guarantees that in order to be president, one has to manage a studio

Guarantees that no two presidents may manage the same studio

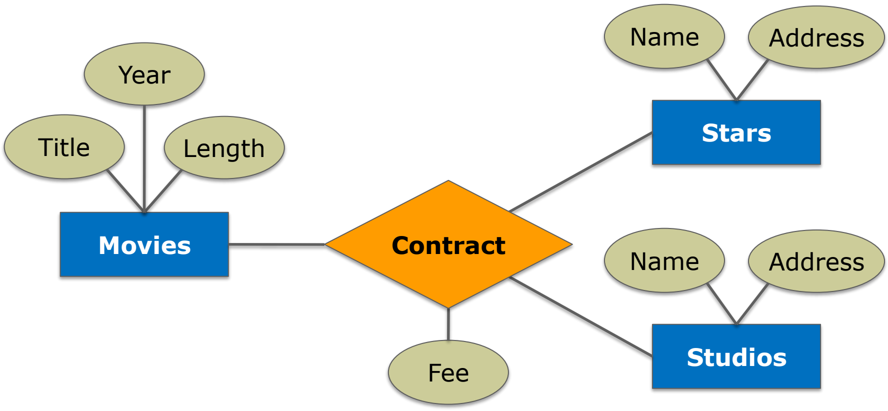
2nd candidate

2nd candidate is not preferred. Why? What constraint it misses?

```
CREATE TABLE presidents (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    age         INTEGER
)

CREATE TABLE studios (
    ID          SERIAL PRIMARY KEY,
    name        TEXT,
    address     TEXT,
    managedBy   INTEGER  REFERENCES presidents (ID) UNIQUE
)
```

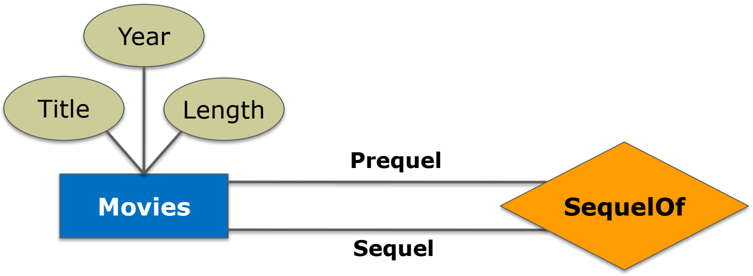
Example 4: 3-Way Relationship



- A studio has contracted with a particular star to act in a particular movie
  - No ownership of movies by studios

```
CREATE TABLE contract (
    ID               SERIAL,
    movie            INTEGER REFERENCES movies (ID) NOT NULL,
    star             INTEGER REFERENCES stars (ID) NOT NULL,
    owner            INTEGER REFERENCES studios (ID) NOT NULL,
    fee              INTEGER
)
```

Example 5a : Self-Relationships with Roles

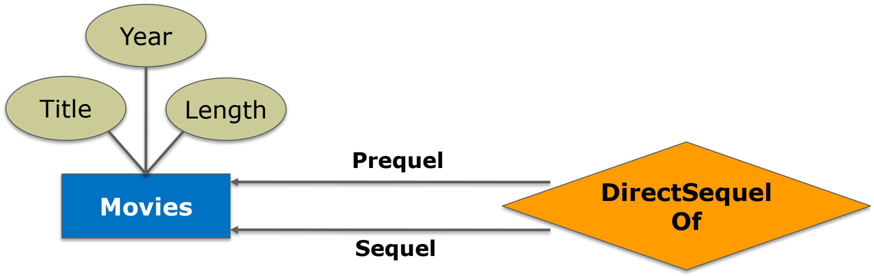


```
CREATE TABLE movies (
    ID          SERIAL PRIMARY KEY,
    …
)

CREATE TABLE sequelof (
    ID              SERIAL,
    prequel         INTEGER REFERENCES movies (ID) NOT NULL,
    sequel          INTEGER REFERENCES movies (ID) NOT NULL
)
```

Notice the use of roles as attributes names for the foreign keys

Example 5b : Combo: One-to-one Self-Relationship

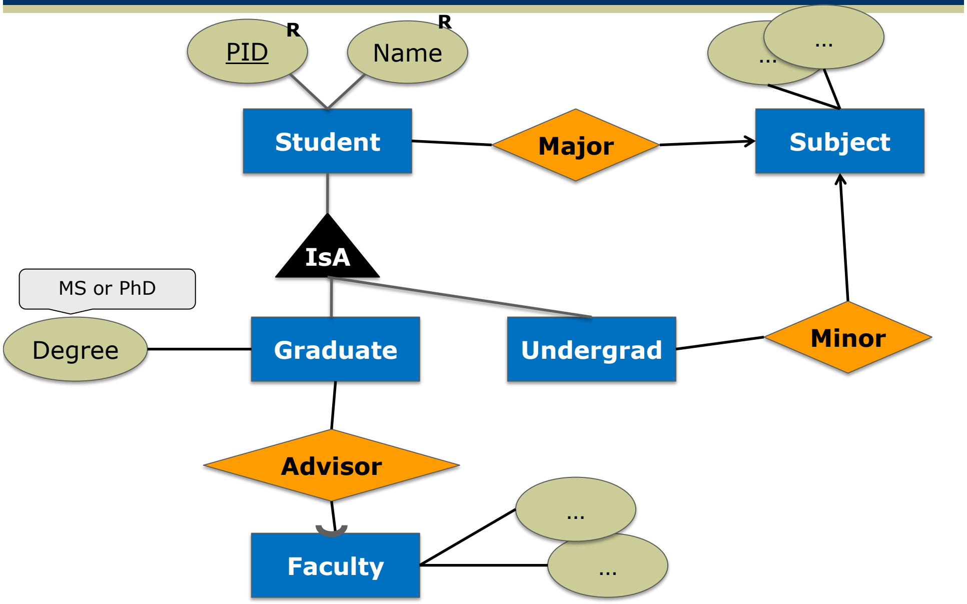


A movie has at most one direct "prequel" and at most one direct "sequel"

Modeling movie sequels by "DirectSequelOf" is preferable to using "SequelOf" of previous slide

A lesson about database design:
- Good designs avoid redundancy.
- No stored piece of data should be inferable from other stored pieces of data

Example 6: Subclassing



Schemas for subclassing: Candidate 1

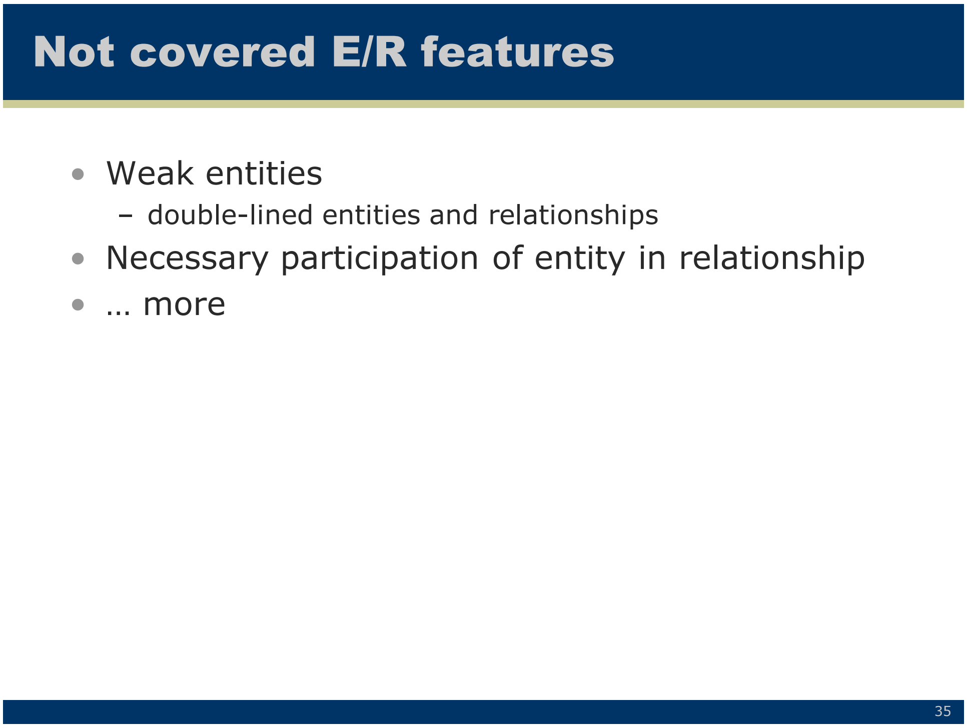
```
CREATE TABLE student(
        ID      SERIAL PRIMARY KEY,
        pid     TEXT NOT NULL UNIQUE,
        name    TEXT NOT NULL,
        major   INTEGER REFERENCES subject(ID)
)
CREATE TABLE undergrad(
        studentid       INTEGER REFERENCES student(ID) NOT NULL,
        minor           INTEGER REFERENCES subject(ID)
)
CREATE TABLE graduate(
        studentid      INTEGER REFERENCES student(ID) NOT NULL,
        degree  TEXT NOT NULL CHECK (degree=“PhD” OR degree=“MS”),
        advisor         INTEGER REFERENCES faculty(ID) NOT NULL
)
CREATE TABLE subject(
        ID      SERIAL PRIMARY KEY,
        …
)
CREATE TABLE faculty(
        ID      SERIAL PRIMARY KEY,
        …
)
```

+ captures constraints
- Information about graduates is spread on two tables
- Creating a report about students is a tricky query
To appreciate the above wait till we discuss SQL

Schemas for subclassing: Candidate 2

```
CREATE TABLE student(
        ID      SERIAL PRIMARY KEY,
        pid     TEXT NOT NULL UNIQUE,
        name    TEXT NOT NULL,
        kind    CHAR(1) CHECK (kind=‘U’ OR kind=‘S’),
        major   INTEGER REFERENCES subject(ID),
        minor   INTEGER REFERENCES subject(ID),
        degree          TEXT CHECK (degree=“PhD” OR degree=“MS”),
        advisor         INTEGER REFERENCES faculty(ID)
)
CREATE TABLE subject(
        ID      SERIAL PRIMARY KEY,
        …
)
CREATE TABLE faculty(
        ID      SERIAL PRIMARY KEY,
        …
)
```

-misses constraints
E.g., notice that it does not capture that a graduate student must have an advisor since we had to make the advisor attribute non-required in order to accommodate having undergraduates in the same table



Not covered E/R features

- Weak entities
  - double-lined entities and relationships
- Necessary participation of entity in relationship
- … more