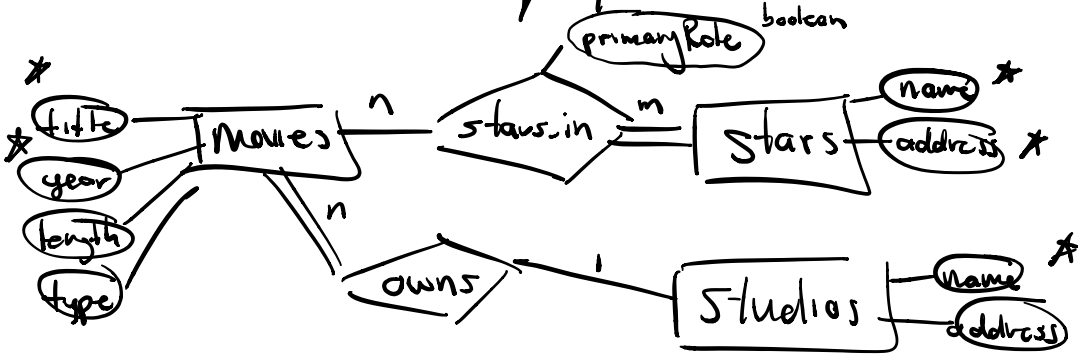
Transforming E/R → Relational Schema



Step 1: identify the important attributes of each entity set

relational schema

$$\Rightarrow \begin{cases} \text{movies}(\underline{\text{title}}, \underline{\text{year}}, \text{length}, \text{type}) \\ \text{stars}(\underline{\text{name}}, \underline{\text{address}}) \\ \text{studios}(\underline{\text{name}}, \text{address}) \end{cases}$$

now create tables for these entity sets:

```
CREATE TABLE studios (
    name varchar(40),
    address varchar(255),
    PRIMARY KEY (name));
```

Step 2: Relationships → Relations

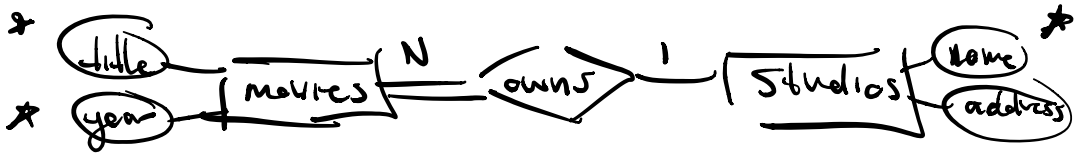
a) for each entity set involved in the relationship, take its key attributes

b) add any of the relationship's own attributes

$\Rightarrow$ starsIn(title, year, name, address, primaryRole)

```
CREATE TABLE starsIn(
  movie_title varchar(30),
  movie_year int,
  star_name varchar(30),
  star_address varchar(255),
  primaryRole boolean,
  PRIMARY KEY (movie_title, movie_year, star_name, star_address),
  FOREIGN KEY (movie_title, movie_year)
    REFERENCES movies(title, year),
  FOREIGN KEY (star_name, star_address)
    REFERENES stars(name, address));
```

* optimization can happen for N:1, 1:N, or 1:1 relationships
  - in general just add the primary key from the "1" side as an attribute (foreign key) on the "N" side




```
CREATE TABLE movies(
  title varchar(40)
  year int,
  length int
```

```
type varchar(30),
studio_name varchar(40),
PRIMARY KEY (title, year),
FOREIGN KEY (studio_name) REFERENCES
    studio (name));
```

SQL Data Types:

Text/String data types:

CHAR(n) - fixed length string
max n (length) is 255

VARCHAR(n) - variable length string
max n (length) is 255

TEXT - variable length up to
65,535 characters

Number data types:

INT - integers using 32 bits
- defaults to SIGNED
- you can specify UNSIGNED
to get 1 extra bit worth of values

BIGINT – integers using 64 bits
- same deal with SIGNED/UNSIGNED

DECIMAL(s,p) – decimal point number with max length 's' and max digits following decimal 'p'
max s = 38

Time data types:

DATE - format YYYY-MM-DD

TIME - format HH:MM:SS

DATETIME - format YYYY-MM-DD HH:MM:SS

True/False data:

BOOLEAN – alias TINYINT(1)
- you can use TRUE and FALSE keywords

Binary data type(s):

BLOB
- this is used for storing raw binary
- this cannot really be used in queries because it is not considered a scalar value in MySQL so you can't do searching or comparison in BLOBs
- if you're trying to store files, probably better to store them in an actual filesystem, and then store file paths as strings in a DB

NULL
- used to represent empty values
- similar to the empty set
- in MySQL any data type can be NULLable
- any column can be declared NOT NULL which disallows empty values
  - PRIMARY KEYS must be NOT NULL
- FOREIGN KEYS can be and often are NOT NULL which is how we implement participation constraints (NOT NULL = total participation)
- all columns are NULLable by default, except PRIMARY KEYS

final example:

```
CREATE TABLE persons (
    id INT,
    name VARCHAR(60) NOT NULL,
    age UNSIGNED INT,
  friend_id INT,
  PRIMARY KEY (id),
 FOREIGN KEY (friend_id) REFERENCES
        friend (id));
```

Relational Algebra

- a query language
  - a formal, procedural query language

- this language is composed of 5 fundamental operators:

$$\{ \sigma, \pi, \times, \cup, - \}$$

Assume the following schema:

Student (name, <u>sid</u>, ssn, status)
Faculty (name, <u>fid</u>, ssn, salary)
Course (name, credits, dept, <u>cnum</u>, fid)
Grade (<u>sid</u>, <u>cnum</u>, <u>semester</u>, grade)
Enrolled (<u>sid</u>, <u>cnum</u>)

σ (Selection operator)
- selects tuples that satisfy a given predicate

$$\sigma_p(R)$$

eg. $\sigma_{name="Park"}(Faculty)$

↳ select tuples from the Faculty relation that satisfy name = "Park"

↳ {(Park, 3, 452, 105)}

$\sigma_{name = "Bob"}(Students)$

↳ {(Bob, 99, 121, 4)}

$\sigma_{status > 3}(Students)$

↳ {(Bob, 99, 121, 4), (Mary, 73, 395, 4)}

$\sigma_{credits > 3 \wedge dept = "CS"}(Courses)$

$\wedge$ - and operator, same as && in programming

$\vee$ - or operator, same as ||

↳ {(simulation, 5, CS, 426, 7)}

$\sigma_{cnum > 500}(Enrolled)$

↳ { }

π (Projection operator)

- copy relation with only the attributes given as arguments

$$\pi_{a_1, a_2 \ldots}(R)$$

e.g: $\pi_{name, sid}(Students)$

↳ copy the Student relation with only the attributes 'name' and 'sid'

↳ {(Bob, 99),
(Sue, 19),
(John, 12),
(Mary, 73)}

$\pi_{salary}(Faculty)$

↳ {(1),
(81),
(105)}

$\pi_{name}(\sigma_{status > 3}(Students))$

↳ {(Bob),
(Mary)}

- Say now we want to get the names of all students enrolled in course 321 ....?
  - we need an operator that takes two relations, Student and Enrolled, and combines them....

X (Cartesian product operator)
- takes the cross product of two relations, i.e. each tuple in the second relation is concatenated to each tuple in the first

$$R = A \times B$$

- the schema of the resulting relation, R, is the combined attributes of both participating relations, A and B

$$R = \text{Student} \times \text{Enrolled}$$

schema of R
↳ (student.name, student.sid, student.ssn, student.status, enrolled.sid, enrolled.cnum)

Now that we have this huge table from the cross product, how do we get the names of students enrolled in 321?

$\Pi_{student.name}(\sigma_{enrolled.cnum=321}(Student \times Enrolled))$

Does this work?

(Bob, Bob, Bob, Bob, Sue, Sue, Sue, Sue, John, John, John, John, Mary, Mary, Mary, Mary)

(Bob, Sue, John, Mary) ... But John isn't in 321

$\Pi_{student.name}(\sigma_{student.sid=enrolled.sid}(\sigma_{enrolled.cnum=321}(Student \times Enrolled)))$

this is the correct query