Introduction of the relational model

- Introduced by Codd (*A Relational Model of Data for Large Shared Data Banks*, 1970)
- Basic idea is to represent data by tables
- Codd describes two languages, (1) a **procedural** language, called the relational algebra, it which you can describe how the computer can evaluate a query and (2) a **declarative** language, called the tuple calculus (which eventually became SQL), in which the user can specify what the result of the query should be, without saying how it should be computed
- Main result of his paper: Both languages are equivalent, and there are effective algorithms to convert from one to another
- How does this work? You may have seen other "declarative" languages, and they don't have nice properties like this
- Key idea: Restrict what the language can express. This is one of the few, languages that you will see, that is not Turing complete
- In fact, we also want very low complexity. Relational languages are usually linear (for those of you who have studied complexity, you might have seen the class called LOGSPACE)

Acceptance of the relational model

- Initial reaction from some: Very nice mathematics, but it will be too inefficient in practice. Also competed with IBM's IMS
- Prototype: System/R from IBM San Jose (*Access Path Selection in a Relational Database Management System*, 1979)
- Query language SEQUEL, later evolved to SQL
- A major project that addressed all the design issues: How to represent relations, expand the query language, and how to optimize queries. Many parts of this course are based on their techniques
- Slow acceptance, as most existing data was in other formats, and the benefits did not always justify converting the data
- Became the main DBMS with the introduction of Workstations and PCs

Brief history

- Early systems: DTBG, IMS, etc. required explicit navigation by the use
- Relational model: Becomes standard by the late 1980s
- Certain weaknesses are apparent. Not all data fits conveniently into tables, so there have been various proposals to extend the data model to accomodate these feature without losing the advantages of the relational mode
  - Nested relations
  - Object-oriented, Object-relational
  - XML
  - Extensions with abstract types such as data blades

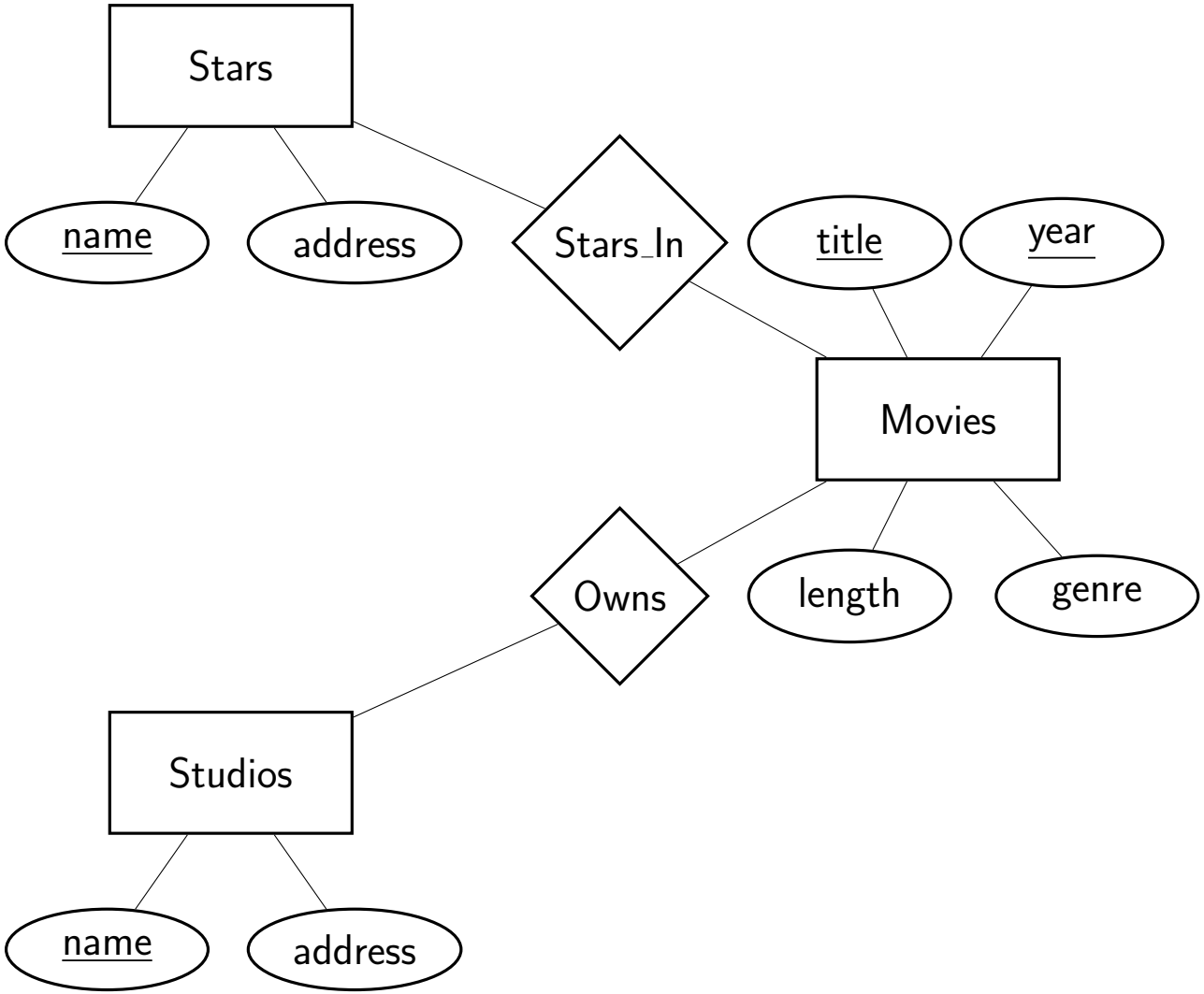
The relational model

title	director	actor
Chinatown	Polanski	Polanski

- First row: Column heads, attributes
- Other rows: Tuples. Each value in each column must correspond to the domain declared for the header
- For this lecture, we shall write $R(\underline{A}, B, C)$ to mean that R is a relation with attributes A,B,C, underlining A to show that it is a key.

Conversion from ER to Relational

- Convert each entity to a relation
- The attributes of an entity become attributes of the relation
- Keys of an entity become keys of the relation


Stars
name
address
Stars_In
title
year
Movies
Owns
length
genre
Studios
name
address

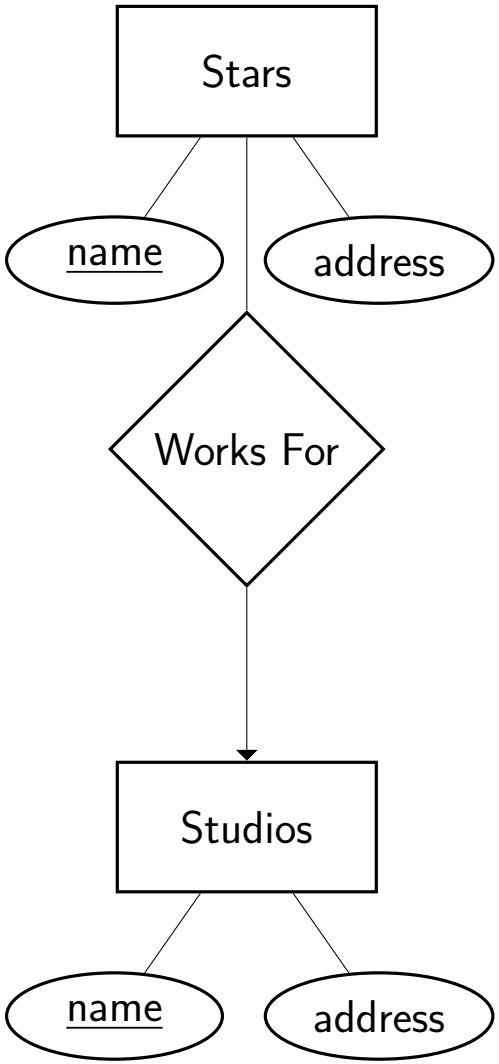
Entities

- Movie(<u>title</u>, <u>year</u>, length, genre)
- Stars(<u>name</u>, address)
- Studios(<u>name</u>, address)
- Note that 2 attributes are called "name"
- This is allowed, but one must be careful when writing queries to remember that they mean different things
- All of the attributes of a single relation **must** be different

Converting Relatonships

- A relationship between E and F is a set of pairs of entities in E and entities in F
- this can be represented by a relation whose attributes are the keys of the entities in E and F
- These are all keys for the relationship
- $\text{Stars\_In}(\underline{\text{name}}, \underline{\text{title}}, \underline{\text{year}})$
- $\text{Owns}(\underline{\text{name}}, \underline{\text{title}}, \underline{\text{year}})$

Another example



How to convert "Works For"?

Renaming attributes

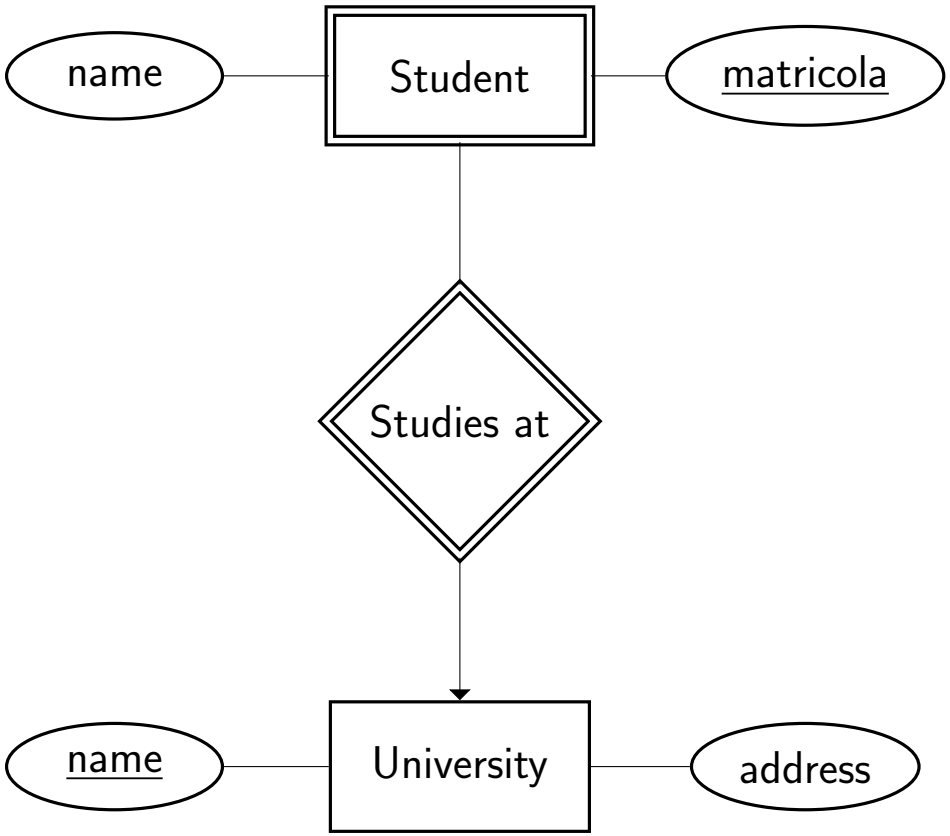
- Direct approach yields $\text{Works\_For}(\underline{\text{name}}, \underline{\text{name}})$
- But this is an illegal schema
- In such a case, we need to change some of the attribute names

$\text{Stars}(\underline{\text{starName}}, \text{address})$

$\text{Studios}(\underline{\text{studioName}}, \text{address})$

$\text{Works\_For}(\underline{\text{studioName}}, \underline{\text{starName}})$

What about weak entities?



Try the same approach

- Entity University: $\mathrm{University}(\underline{\mathrm{name}}, \mathrm{address})$
- Entity Student. Note:
  - We must use the full key, i.e., matricola and name (of University)
  - We must rename one of the two "name" attributes
- We get: $\mathrm{Student}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}}, \mathrm{studentName})$
- The relationship "Studies at" relates a key of "Student" to a key of "University". So we get:
- $\mathrm{StudiesAt}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}}, \underline{\mathrm{name}})$
- What is wrong?

- We have:

  $\mathrm{University}(\underline{\mathrm{name}}, \mathrm{address})$

  $\mathrm{Student}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}}, \mathrm{studentName})$

  $\mathrm{StudiesAt}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}}, \underline{\mathrm{name}})$

- Note that in "StudiesAt", the two "name" attributes will always be the same. We then have

  $\mathrm{University}(\underline{\mathrm{name}}, \mathrm{address})$

  $\mathrm{Student}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}}, \mathrm{studentName})$

  $\mathrm{StudiesAt}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}})$

- Note that the last schema is contained in the second. This does not always mean that the data is contained (think of the relation between "Star" and "Studio" with and without "Contract")

- Here, on the other we cannot have a pair of "matricola" and (university) "name" without having a corresponding "studentName". So we can delete the last relation, getting

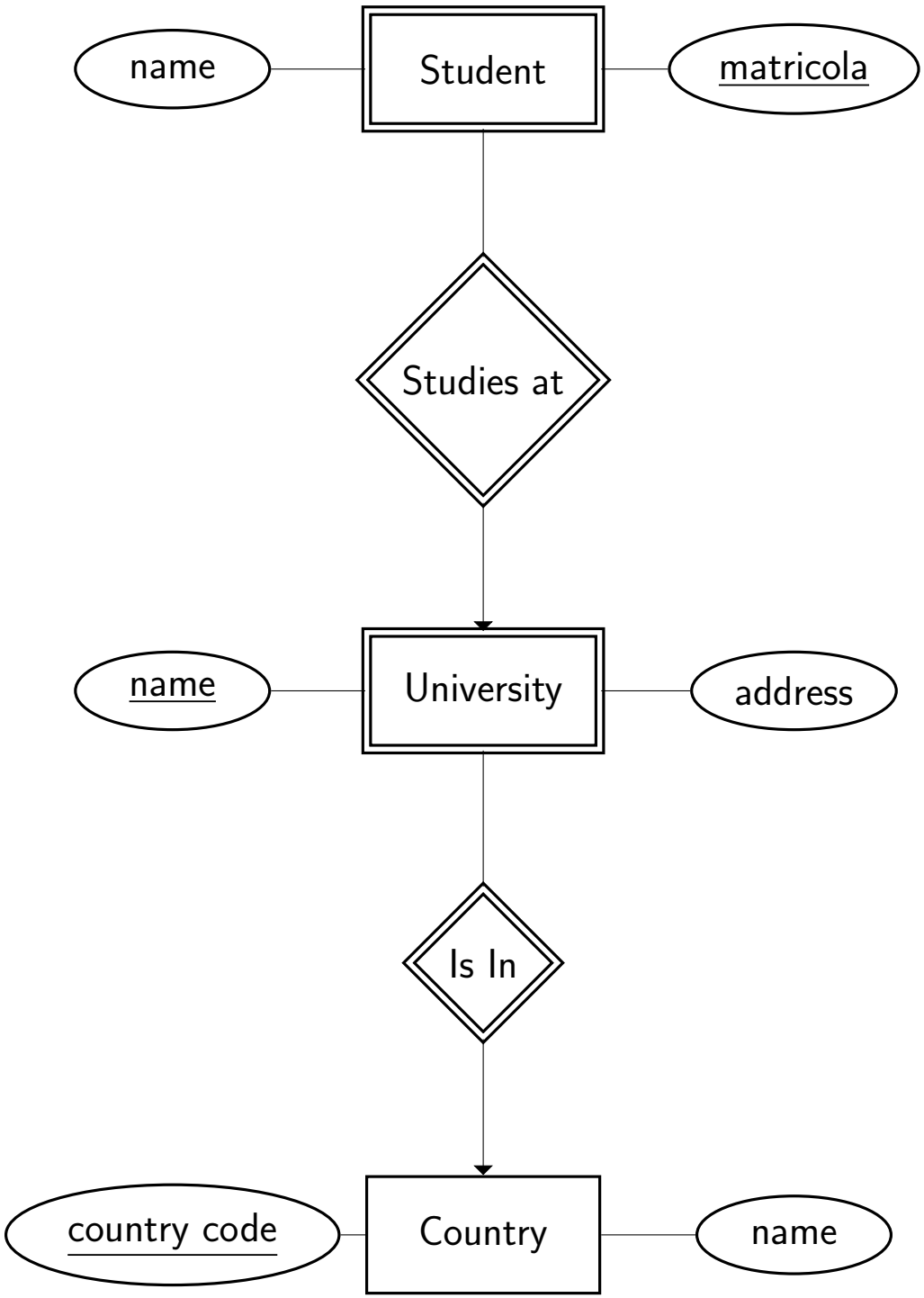
  $\mathrm{University}(\underline{\mathrm{name}}, \mathrm{address})$

  $\mathrm{Student}(\underline{\mathrm{matricola}}, \underline{\mathrm{name}}, \mathrm{studentName})$

Summary

To convert a weak entity into the relation model, simply convert both entities, remembering to use the full key for the weak entity.

Do not convert the supporting relationship


name
Student
matricola
Studies at
name
University
address
Is In
country code
Country
name

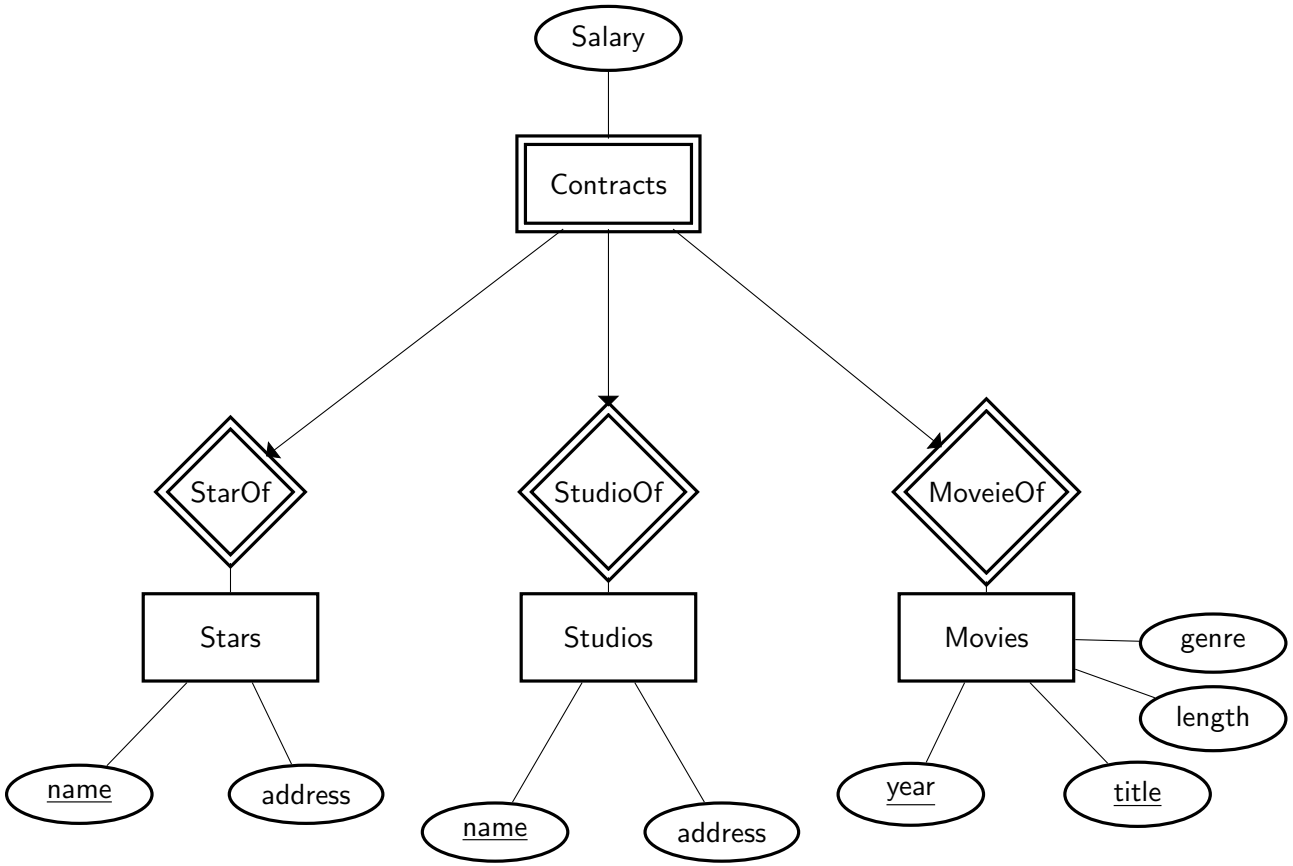
Relational equivalent

Country(<u>countryCode</u>, countryName)

University(<u>universityName</u>, <u>countyCode</u>, universityAddress)

Student(<u>matricola</u>, <u>universityName</u>, <u>countryCode</u>, studentName)

Another example



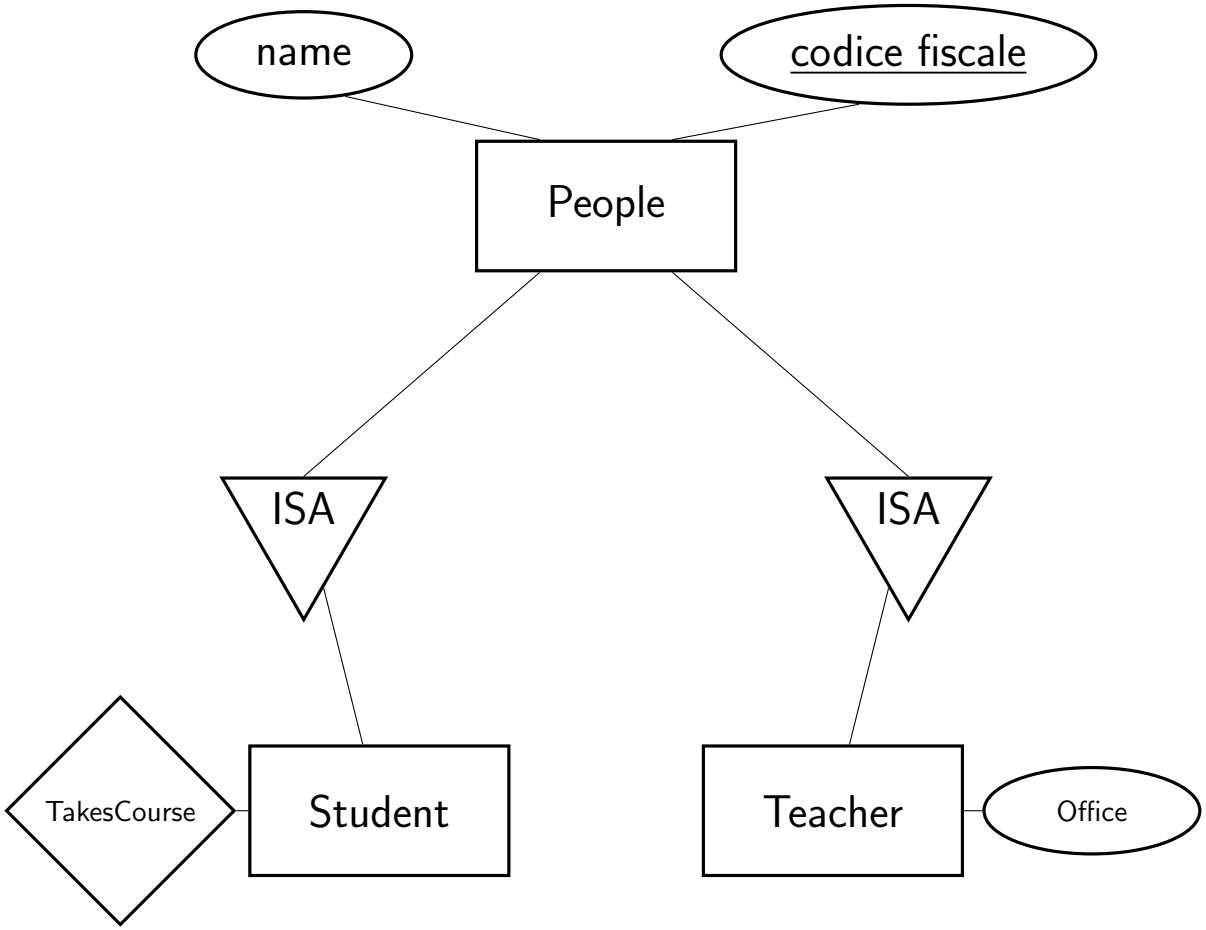
Relational model

- Contracts(<u>StarName</u>, <u>StudioName</u>, <u>title</u>, <u>year</u>, salary)
- Stars(<u>StarName</u>, address)
- Studios(<u>StudioName</u>, address)
- Movies(<u>title</u>, <u>year</u>, genre, length)

Note that we could have used

- Stars(<u>name</u>, address)
- Studios(<u>name</u>, address)

Subclasses


name
codice fiscale
People
ISA
ISA
TakesCourse
Student
Teacher
Office

Main points

- `Student` and `Teacher` are subclasses of `People`
- Every entity in each of these entity classes are also entities in `People`
- They *inherit* all the properties (attributes, relationships) of their parents
- Each Student and Teacher has name and codice fiscale as attributes
- Both have codice fiscale as key
- `Teacher` *also* has an attribute `Office`
- `Student` *may also* participate in the relationship `TakesCourse` with an entity `Courses` (not shown)

Conversion to relational: First method

- One relation with many attributes
- `People(codiceFiscale,name,office,...)`
- What happens with `office` when the person is not a Teacher?
- Value is missing (`NULL`)
- Only some tuples participate in the relationship `TakesCourse`
- Disadvantages
  - Tuples can have many attributes, most of them missing
  - Difficulty in enforcing constraints ("every student must take at least one course")

Another approach

- One relation for each entity
- `People(name,codiceFiscale)`
- Subclasses inherit attributes from parent
- `Student(name,codiceFiscale)`
- `Teacher(name,codiceFiscale,office`
- Main disadvantage: lots of data is repeated (and must all be updated)
- Why not delete `People`?

The relational model

- Our standard example: Movie database, similar to our ER model
- We now describe the schema of our database
- Example uses features that are used in many applications
- The specific combination of features is used for didactic purposes, and is sometimes a bit artificial

The schema

```
CREATE TABLE Movies(
    title          CHAR(100),
    year           INT,
    length         INT,
    genre          CHAR(10),
    studioName     CHAR(30),
    producerC#     INT,
    PRIMARY KEY (title,year)
)
```

- First four attributes as in ER model
- `primary key` will be explained later
- `studioName` included as attribute instead of modelling the relationship separately
- `producerC#` identifies the producer of the move (see `MovieExec` below)

```
CREATE TABLE MovieStar(
    name         CHAR(30)  PRIMARY KEY,
    address      VARCHAR (255),
    gender       CHAR (1),
    birthdate    DATE
)
```

We could also have used

```
PRIMARY KEY (name)
```

- We have added two new attributes
- `char` single character (`M` or `F`)
- `date` special type for dates, with builtin operations on dates

```
CREATE TABLE StarsIn(
    movieTitle   CHAR(100),
    movieYear    INT,
    starName     CHAR(30),
    PRIMARY KEY (movieTitle,movieYear,starName)
)
```

- All three attributes are the only key
- Use of `movieTitle` and `movieYear` is only to illustrate certain aspects of SQL later

```
CREATE TABLE MovieExec(
    name       CHAR(30),
    address    VARCHAR(255),
    cert#      INT  PRIMARY KEY,,
    netWorth   INT
)
```

- `cert#` number (such as codice fiscale but system determined) identifying an executive
- Why not also for `MovieStar`? Only for didatic reasons...
- Movie Executive could be movie producer (as above) or president of a studio

```
CREATE TABLE Studio(
    name      CHAR(30) PRIMARY KEY,
    address   VARCHAR(255),
    presC#    INT
)
```

The relational algebra

- Our first query language
- A procedural language. Describe how to compute the result we are looking for
- We study:
  - The original language of Codd. Relations are *sets*
  - A second language, closer to how relations are implemented. Relations are *bags*
  - Additional operators not in the orginal algebra
- In reality, data in a computer is always ordered, but this is ignored in Codd's algebra
- Lack of order lets the query processor change the order of data for efficiency

Basic relational algebra

- Corresponds to the algebra of Codd
- Three standard set-theoretic operations: union ($\cup$), intersection (cap), difference (). All are binary operations
- Two simple unary operations: selection (σ) and projection (π) and one binary one: cross product ($\times$)
- A much more complicated binary operation: natural join ($\bowtie$)
- A trivial unary operator: renaming (ρ)

Schemas of operators

- Each operator is descibed by two parts: the permitted schemas of the inputs, the schema of the output, and the data in the output
- Schema $R(A, B, C)$ means that relation R has three attributes, with names A, B, and C
- Union, intersection, and difference can only be applied to relations with the same schema (not just the same number of columns). The output has the same schema as the input
- Selection has the form $\sigma_C(R)$ where C is a "condition" (to be discussed later) and R contains all the atributes used in C. $\sigma_C(R)$ has the same schema as R
- Projection $\pi_{A1,...,Ak}(R)$, where $A1$, ..., Ak are attributes of R, results in a relation with schema $(A1, \ldots, Ak)$.
- Cross product is defined when R and S have disjoint schemas. If R has schema $(A1, ..., Am)$ and S has schema $(B1, ..., Bn)$, then $R \times S$ has schema $(A1, ..., Am, B1, ..., Bn)$

Union

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$S(A, B, C)$:

A	B	C
1	1	1
2	2	2
3	4	5

$R \cup S$ has schema (A, B, C) and values

A	B	C
1	1	1
2	2	2
1	2	3
3	4	5

Intersection

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$S(A, B, C)$:

A	B	C
1	1	1
2	2	2
3	4	5

$R \cap S$ has schema (A, B, C) and values

A	B	C
1	1	1
2	2	2

Difference

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$S(A, B, C)$:

A	B	C
1	1	1
2	2	2
3	4	5

$R - S$ has schema (A, B, C) and values

A	B	C
1	2	3

Difference, continued

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$S(A, B, C)$:

A	B	C
1	1	1
2	2	2
3	4	5

On the other hand $S - R$ has schema (A, B, C) and values

A	B	C
3	4	5

$R(A, B) \cup S(A, B, C)$, $R(A, B, C) \cap S(A, B, D)$ etc. are not defined

Selection

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$\sigma_C(R)$ has the same schema as R

$\sigma_{A=1}(R)$:

A	B	C
1	1	1
1	2	3

$\sigma_{A=B}(R)$:

A	B	C
1	1	1
2	2	2

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$\sigma_{B<C}(R)$:

A	B	C
1	2	3

$\sigma_{A=B \texttt{ AND } B=C}(R)$:

A	B	C
1	1	1
2	2	2

We also have `OR`, `NOT`, <> (not equal), as well as arithmetic for numeric domains

Projection

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$\pi_{A,B}(R)$:

A	B
1	1
2	2
1	2

$\pi_A(R)$:

A
1
2

Renaming

$R(A, B, C)$:

A	B	C
1	1	1
2	2	2
1	2	3

$\rho_{R(X,B,A)}(R)$ has schema $R(X, B, A)$ and data

X	B	A
1	1	1
2	2	2
1	2	3

Cross product

$R(A, B)$:

A	B
1	1
3	1
4	4

$S(C, D)$:

C	D
1	5
1	6
4	7

$R \times S$:

A	B	C	D
1	1	1	5
1	1	1	6
1	1	4	7
3	1	1	5
3	1	1	6
3	1	4	7
4	4	1	5
4	4	1	6
4	4	4	7

Natural Join

$R(A, B)$:

A	B
1	1
3	1
4	4

$S(B, C)$:

B	C
1	5
1	6
4	7

$R \times S$:

A	B	C
1	1	5
1	1	6
3	1	5
3	1	6
4	4	7

- Combine tuples from R and S that agree on the common attribute B
- For example

$$(1,1),(1,5) \Rightarrow (1,1,5)$$

- or

$$(3,1),(1,6) \Rightarrow (3,1,6)$$

- but

$$(3,1),(4,7) \not\Rightarrow$$

Another view of the natural join

- Take the cross product $R(A, B) \times S(B, C)$ (for now, ignore the fact that two attributes are the same)

A	B	B	C
1	1	1	5
1	1	1	6
1	1	4	7
3	1	1	5
3	1	1	6
3	1	4	7
4	4	1	5
4	4	1	6
4	4	4	7

Another view of the natural join, continued

- Next, select those tuples for which the two values of B are equal

A	B	B	C
1	1	1	5
1	1	1	6
3	1	1	5
3	1	1	6
4	4	4	7

- Finally, project onto A, one of the Bs, and C:

A	B	C
1	1	5
1	1	6
3	1	5
3	1	6
4	4	7

- Note that the choice of which B to use doesn't matter, as the values are always the same

More formally

- Given $R(A, B)$ and $S(B, C)$,

  $$U = R \bowtie S = \rho_{U(A,B,C)}\pi_{R.B}\sigma_{R.B=S.B}(\rho_{R(A,R.B)}(R) \times \rho_{S(S.B,C)}(S))$$

- It is often better to write relational algebra expressions in several stages
- $R1 := \rho_{R(A,R.B)}(R)$
- $S1 := \rho_{S(S.B,C)}(S)$
- $T = \sigma_{R.B=S.B}(R1 \times S1)$
- $U = \rho_{U(A,B,C)}\pi_{R.B}(T)$
- We could use any names for the new attribute values; our choice of $R.B$ and $S.B$ anticipates the usage of SQL

The general case

- Given $R(A1, ..., Ak, B1..., Bn)$ and $S(A1, ..., Ak, C1, ..., Cm)$ with common attributes $A1$,. . . , Ak, where $k \geq 0$
- $R \bowtie S$ is defined as follows (we omit the renaming rules)
- $U = \sigma_{\texttt{R.A1=S.A1 AND ... AND R.Ak=S.Ak}}(R \times S)$
- $R \bowtie S = \pi_{R.A1,...,R.Ak,B1,...,Bn,C1,...,Cm}(U)$
- What if $k = 0$? The selection clause is empty, so $R \bowtie S = R \times S$

Another Example

$U(A, B, C)$:

A	B	C
1	2	3
6	7	8
9	7	8

$V(B, C, D)$:

B	C	D
2	3	4
2	3	5
7	8	10

$U \bowtie V$:

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

Theta-Joins

- Join under more general conditions
- "Theta-join": Name for historical reasons (people used to write θ instead of C for the condition)
- Notation $R \bowtie_C S$
- Meaning
  - Take the product of R and S
  - Select the tuples that satisfy C
- Note: $R \times S$ may have repeated attributes
- Since their values may be different (unlike the natural join) we cannot eliminate one of them

Example

$R(A, B)$:

A	B
1	1
6	1
6	4

$S(B, C)$:

B	C
1	5
1	6
4	7

$R \bowtie_{A=C} S$:

A	R.B	S.B	C
6	1	1	5
6	4	1	6

- For conditions on B, we must use $R.B$ notation

  $R \bowtie_{A=S.B} S$:

A	R.B	S.B	C
1	1	1	5
1	1	1	6

- "Almost" $R \bowtie S$

  $R \bowtie_{R.B=S.B} S$:

A	R.B	S.B	C
1	1	1	5
1	1	1	6
6	1	1	5
6	1	1	6

$U(A, B, C)$:

A	B	C
1	2	3
6	7	8
9	7	8

$V(B, C, D)$:

B	C	D
2	3	4
2	3	5
7	8	10

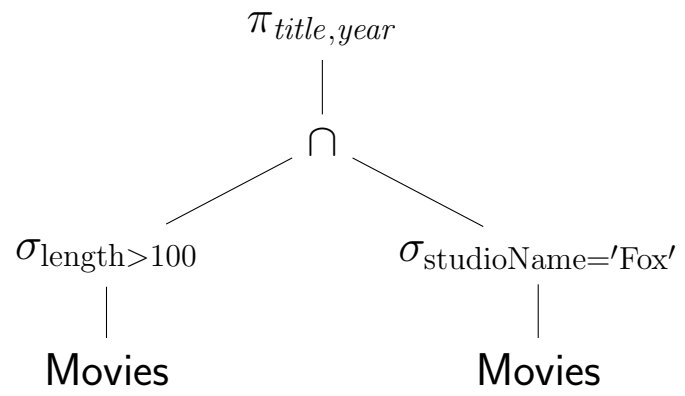
$U \bowtie_{A<D} V$:

A	U.B	U.C	V.B	V.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Queries: Example with real relations

`Movies(title,year,genre,length,studioName)`

- Query: Find the films made by Fox which are longer than 100 minutes
- One possibility:
  1. Select the `Movie` tuples for which `length>100`
  2. Select the `Movie` tuples for which `studioName=Fox`
  3. Compute the intersection of (1) and (2)
  4. Project onto `title` and `year`



$$\pi_{\texttt{title,year}}(\sigma_{\texttt{length}>100}(\texttt{Movies}) \cap \sigma_{\texttt{studioName}=\texttt{'Fox'}}(\texttt{Movies}))$$

- Or

$$\pi_{\texttt{title,year}}\sigma_{\texttt{length}>100\ \texttt{AND}\ \texttt{studioName}=\texttt{'Fox'}}(\texttt{Movies})$$

Relations between the operators

- $R \cap S = R - (R - S)$
- Proof:

$$\begin{aligned} t \in R - (R - S) &\Leftrightarrow t \in R \wedge t \notin (RS) \\ &\Leftrightarrow t \in R \wedge (t \in R \vee t \in S) \\ &\Leftrightarrow (t \in R \wedge t \notin R) \vee (t \in R \wedge t \in S) \\ &\Leftrightarrow (t \in R \wedge t \in S) \\ &\Leftrightarrow (t \in R \cap S) \end{aligned}$$

- Another rule

$$R \bowtie_C S = \sigma_C(R \times S)$$

- We have already seen that natural join can be expressed with $\times$, σ, and π

Algebraic Rules

- There are many identities about the relational algebra. These include trivial identities such as

$$R \cap R = R$$

- Rules similar to multiplication and addition such as

$$R \cup (S \cup T) = (R \cup S) \cup T$$

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

- For join, we have

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Selection and Join

- If C only uses attributes of R

$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$$

- For example, given $R(A, B)$, $S(B, C)$

$$\sigma_{A=1}(R \bowtie S) = \sigma_{A=1}(R) \bowtie S$$

$$\sigma_{B=1}(R \bowtie S) = \sigma_{B=1}(R) \bowtie S = R \bowtie \sigma_{B=1}(S)$$

$$\sigma_{A=B}(R \bowtie S) = \sigma_{A=B}(R) \bowtie S$$

$$\sigma_{C=1}(R \bowtie S) \neq \sigma_{C=1}(R) \bowtie S$$

(in fact, the right-hand side is meaningless)

A few words on optimization

- This rule is very important for optimization
- Suppose that R and S each have 1.000.000 tuples.
- $R \bowtie S$ may be enormous up to 1.000.000.000.000 tuples
- But $\sigma_{A=1}(R)$ will often be much smaller
- $\sigma_{A=1}(R \bowtie S)$: Evaluating this means creating this huge relation, and then looking for the tuples with $A = 1$

Alternative approach

$$\sigma_{A=1}(R) \bowtie S$$

- First evaluate $\sigma_{A=1}(R)$. At worst, this means examining 1.000.000 tuples, and obtaining only a few tuples (1 if A is a key)
- Then examine the 1.000.000 tuples in S, looking for those than join with R
- Much less work: examine only 2.000.000 tuples
- Basic rule of optimization: Do selections as soon as possible