Università
Ca' Foscari
Venezia

Computer Science Applications to Cultural Heritage

Relational Databases

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco
DAIS, Ca' Foscari University of Venice
Academic year 2018/2019

Databases & Cultural Heritage

One important application of computer science to cultural heritage concerns the selection of appropriate filing systems and data structures.

Ubiquitous usage:

- Storage of information regarding tangible and intangible cultural heritage content
- Storage of data to document the history of a specific content in a structured way
- Support the analysis of non-trivial relations between contents, to extract knowledge about our past

What is a database?

An organized system to **store**, **manage** and **retrieve** informations.

Characteristics:

- Must be efficient, especially in information retrieval
- Must be robust against data corruption
- Must be secure, allowing the storage and retrieval of information to different users with different privileges
- Operations typically falls into one of the four categories: **C**reate, **R**ead, **U**pdate, **D**elete (CRUD)

Types of databases

Depending on the underlying **data model**, different types of databases can be used:

Relational databases:

Data is structured in form of relations between entities that are physically represented as tables

Object-oriented databases:

Data is structured in form of objects, with different attributes and hierarchy relations between them

Spatial databases (GIS):

Optimized to store geographical and/or spatial informations and to retrieve informations based on the spatial relations between the elements

Data modeling

The most important aspect when working with data is being able to create a proper **data model** to represent the domain under study

What is a model?

- The representation of **facts** about a phenomenon **given in a formal language** made of the composition of **concepts** which are used to help people know, understand, or simulate a subject the model represents
- The **result of an interpretation process** driven by ideas and knowledge possessed by the interpreter

Data modeling

What do we model?

We model both concrete and abstract knowledge:

- **Concrete knowledge** is about the specific facts that we want to represent.
  - For example: in our context we have artists and paintings. Artist are characterized by a name, year of birth, year of death. Each painting has a creation date and belongs to an artist.
- **Abstract knowledge** is about:
  - **Structure** of the concrete knowledge
  - **Constraints** on the actual values of the concrete knowledge and how they evolve (data integrity)
  - **Rules** to derive new information from known facts

Data modeling

Examples of **Abstract knowledge:**

1. The name of the artist is a "string" made by a sequence of characters. Year of birth and year of death must be positive numbers.
2. The number representing the year of death must be greater than the year of birth.
3. If a painting belongs to an artist, than the creation date must fall between the year of birth and year of death of such artist

Before studying how to use specific kinds of databases (like the relational ones) we need to learn **how to model the knowledge of a specific domain**

Entities and properties

First step is to focus to the concrete knowledge (ie to the facts we want to represent).

To simplify the approach, we can assume that the domain we want to model is composed by **entities**

Each entity may have one or more **properties** that are interesting because they describe facts or characteristics of some entities

Each property has a name (called **attribute**) and a **domain** (the set of all the possible values it can assume)

Properties

A property can be:

Atomic:

if its value is not structured into multiple sub-values. Ex: the name of a person is atomic whereas the address is composed by (city, state, etc)

Unique:

if it has a single value. For example the telephone number of a person may not be unique

Total:

if every entity have a value set for that property (ex. Each person have a name). **Partial** if there exist some entities without that property (ex. Not all persons have a telephone number)

Properties

A property can also be:

Constant:

if its value do not change over time (Ex: The date of birth of a person is constant whereas the address is variable)

Computed:

If its value can be computed from some other properties. (Ex: the age of a person can be computed from the date of birth and the current date)

Classes

All the entities of our domain sharing the same **type** (ie. naturally belonging to the same concept and sharing the same set of properties) are grouped into collections named **classes**.

Definition:

A Class is a collection of homogeneous entities (ie. of all the entities, existing or possible, having the same type).

Synonym: entity set.

Classes

Example:

Mario, Luigi and Maria are all entities defining persons in this room. Such 3 entities have the following properties: a name, an age and an address.

We can naturally group the entities into a class named *Students* which collects all the entities (existing or possible) with the same characteristics (attributes) of Mario, Luigi and Maria.

We usually name classes with plural and capital letter

Class hierarchy

Usually classes can be organized hierarchically such that classes at a lower level specialize the entities at the upper levels.

Example:

- *Persons* and *Students* are classes.
- *Persons* class groups entities having two properties: name and a sex
- *Students* class groups entities that are persons (thus having a name and a sex) but also having a matriculation number
- *Students* is a **sub-class** of *Persons*
- *Persons* is a **super-class** of *Students*

Entity-relationship diagram

When we model the concrete knowledge into a database, we usually start by creating a **conceptual schema** using a graphical formalism called **entity-relationship diagram.**

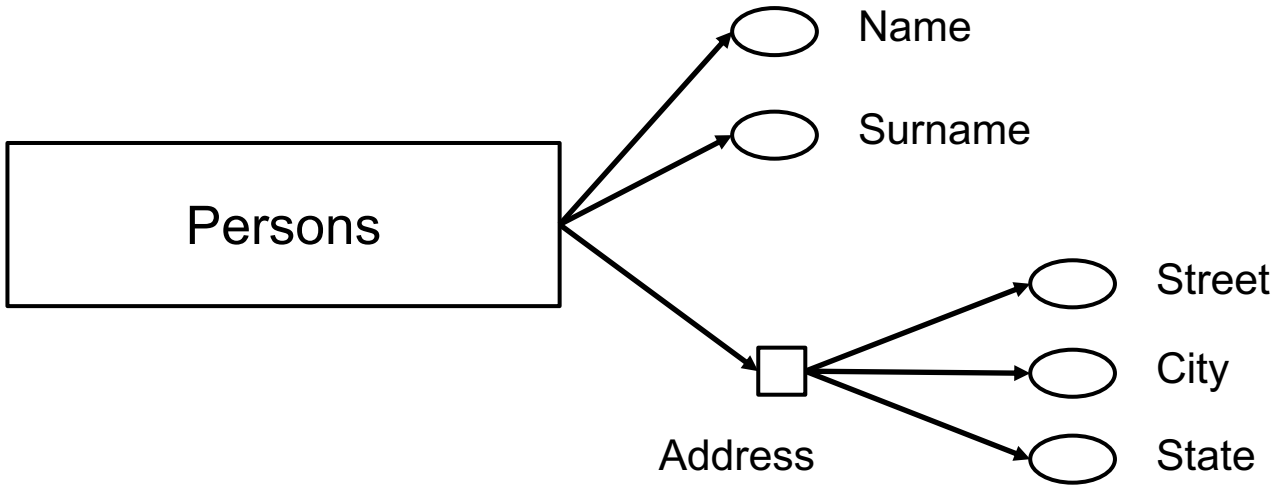
An entity-relationship diagram models groups of homogeneous entities (**classes**) together with their hierarchy and **relationships**

Classes (also known as **entity sets**) and relationships are the basic building blocks used to create the data model.

Università Ca' Foscari Venezia

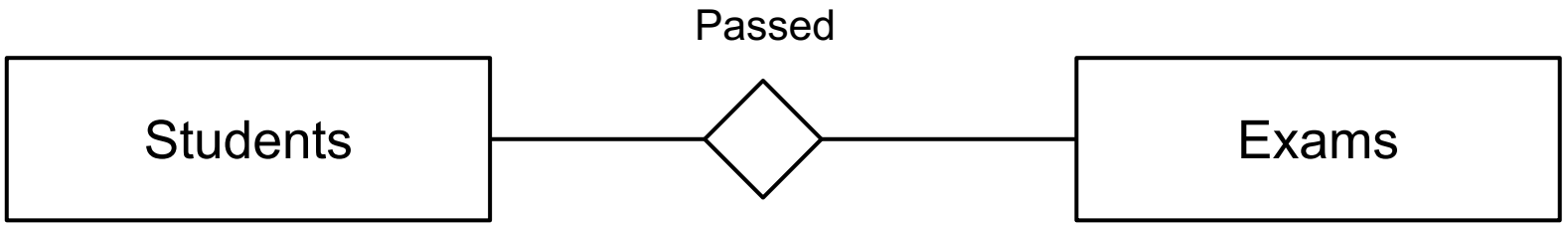
Entity-relationship diagram

- A class in an entity-relationship diagram is represented with a rectangle containing the name of the class.
- Attributes of a class are represented by ovals
- If an attribute is not atomic (ie. structured in multiple sub-attributes) we use a small square instead



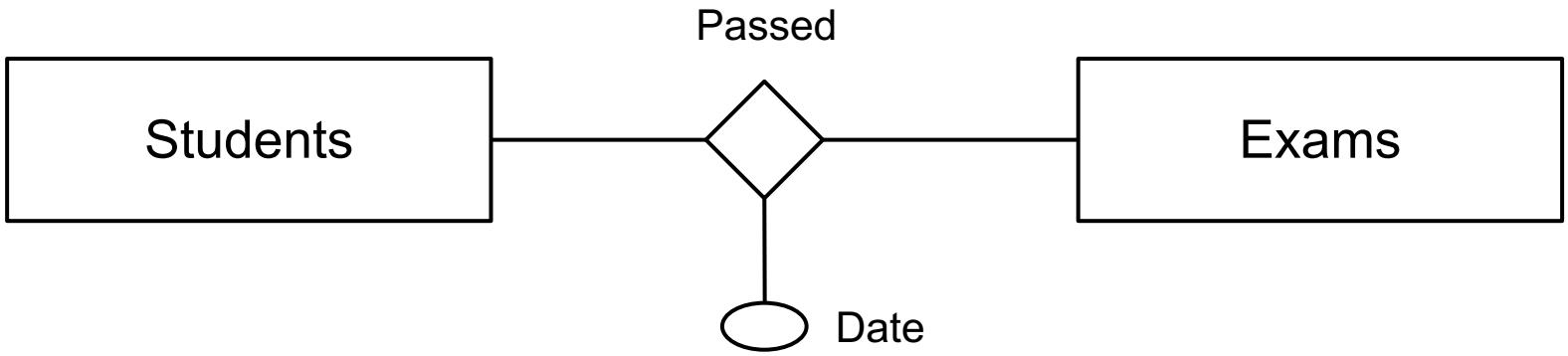
Entity-relationship diagram

- Relationships in an entity-relationship diagram are represented by diamonds, connected with arcs (arrows) to the associated classes
- Each diamond contains the name of the relation using a predicate such that it gives a meaning the sentence with the two connected classes acting as subject and complement
- Sentence is supposed to be read from left to right or from top to bottom



Entity-relationship diagram

- We can add an attribute to the relation using ovals with the same graphical formalism we used for classes



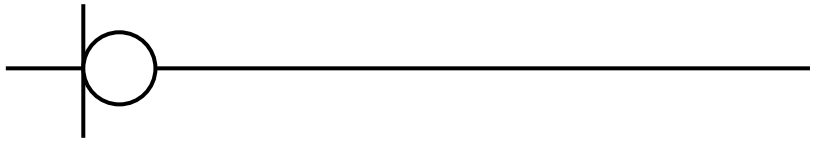
Multiplicity of relationships

- In general, a binary relationship can connect any member of one of its entity sets to any number of members of the other entity set
- It is common to specify the "multiplicity" of a relationship, that can be of 3 types:
  - One-to-one
  - One-to-many (or vice-versa)
  - Many-to-many
- We usually specify if the relation is mandatory (ie all the entities in the set have a relation with the other set) or optional (some entity can have no relation with the other set)

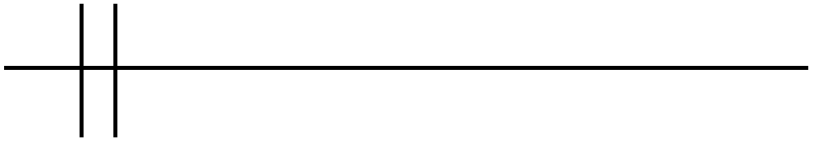
One-to-one

- A one-to-one relationship is represented by a single line.

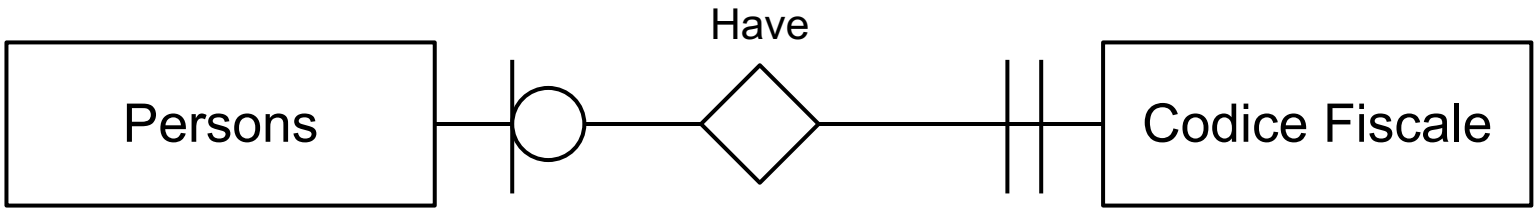
- If the relation is optional we use a circle and a vertical line (means zero or one)



- If the relation is mandatory we use two vertical lines



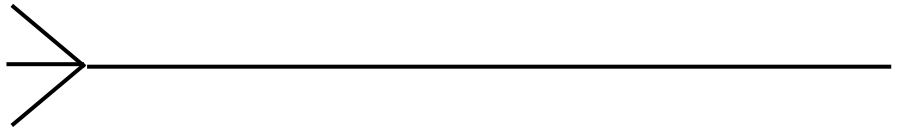
One-to-one



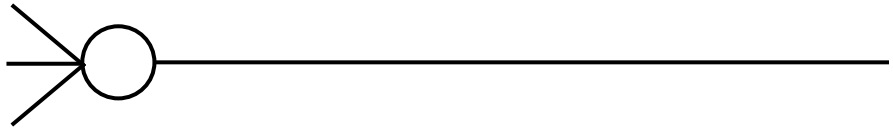
Example: Persons and "Codice Fiscale" (fiscal code) are in a one-to-one relationship. Each person have exactly one codice fiscale (is mandatory). A codice fiscale may have no person associated (refers to a non-existing person)

Many-to-one

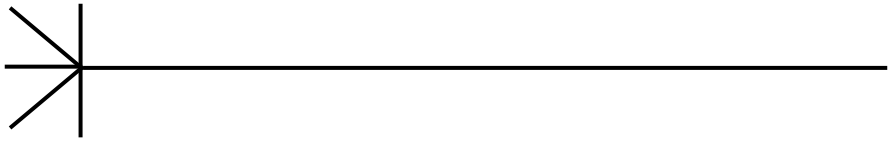
- A many-to-one relationship is represented by a line with multiple ends.



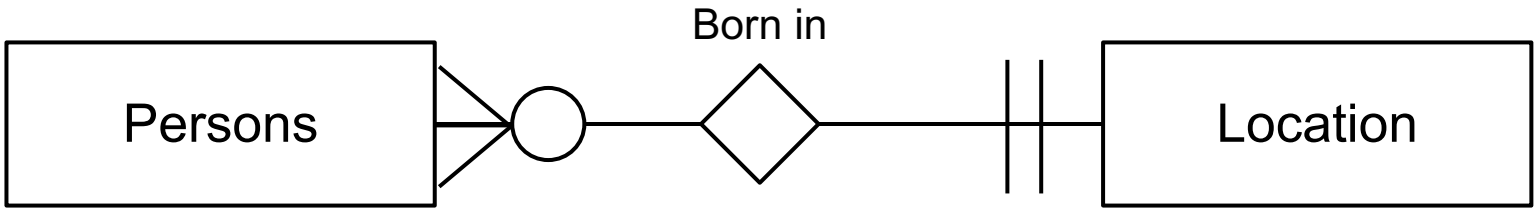
- If the relation is optional we add a circle



- If the relation is mandatory we use a vertical line



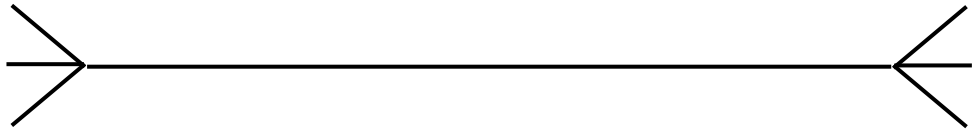
Many-to-one



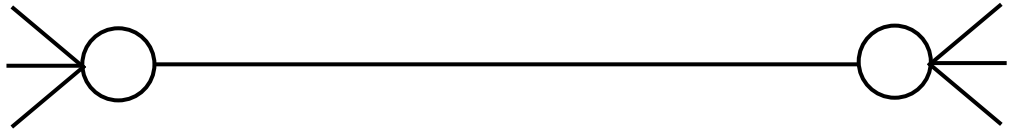
Example: Each person is born at a single unique location. Zero or many persons are born at a certain location

Many-to-many

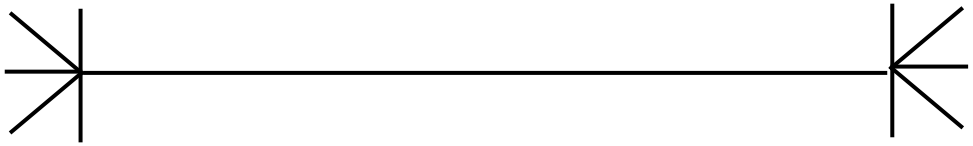
- A many-to-many relationship is represented by a line with multiple ends at both the sides



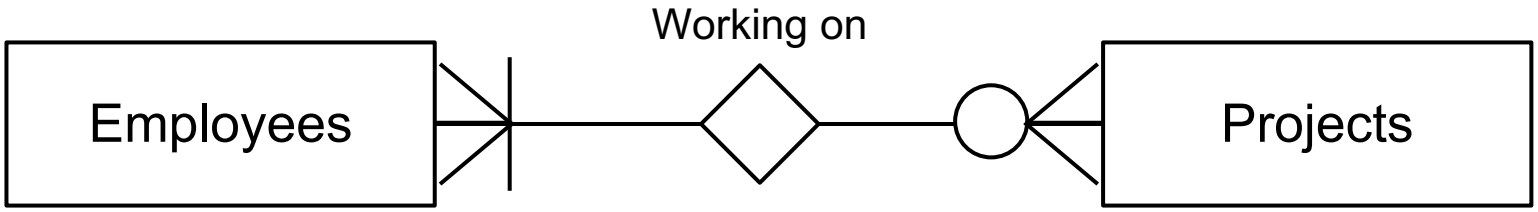
- If the relation is optional we add a circle



- If the relation is mandatory we use a vertical line



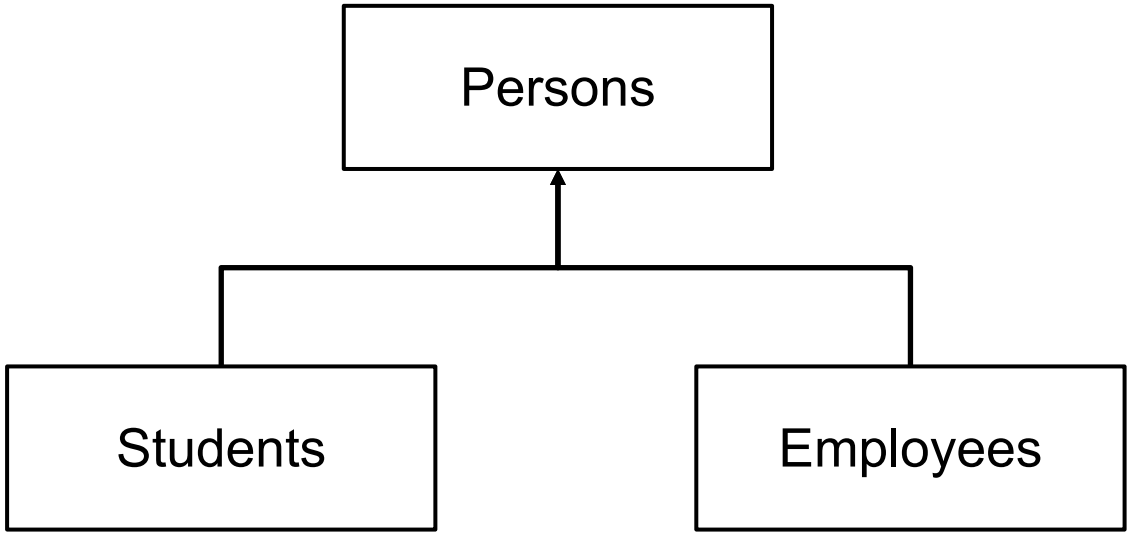
Many-to-many



Example: An employee can work on zero or many projects. Each project can have many employee working on it (at least one)

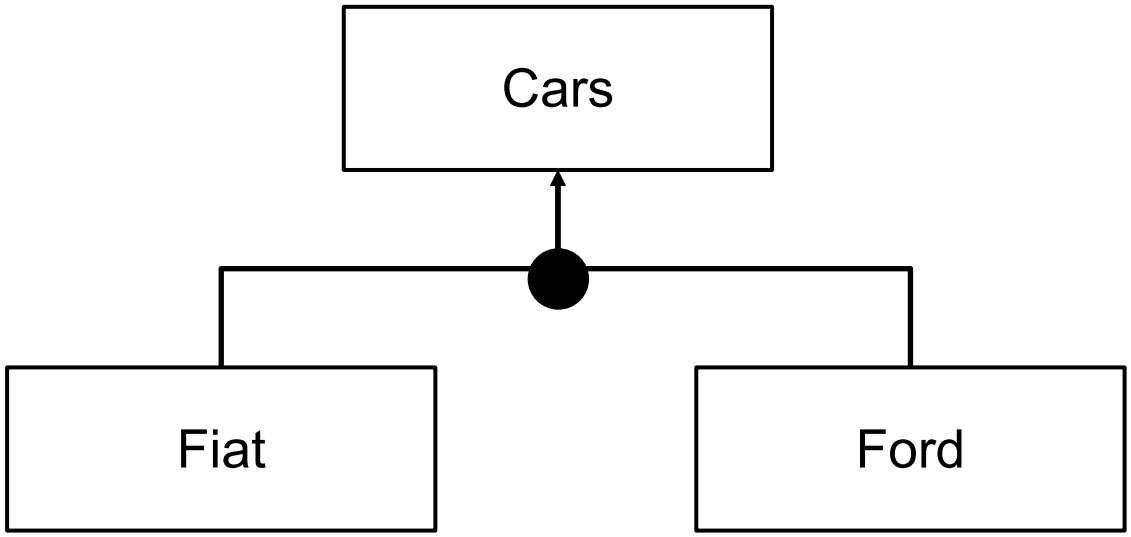
Class hierarchy

- Hierarchical relations between classes can be represented by arranging the classes vertically and using different types of arrows



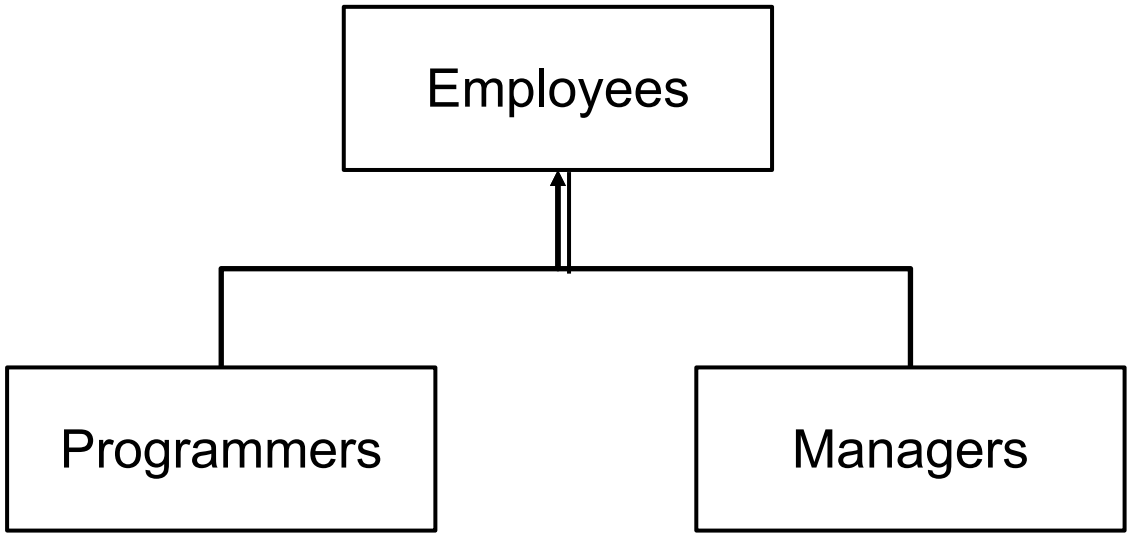
Students and employee are two sub-classes of Persons with no relations between them

Disjoint subclasses



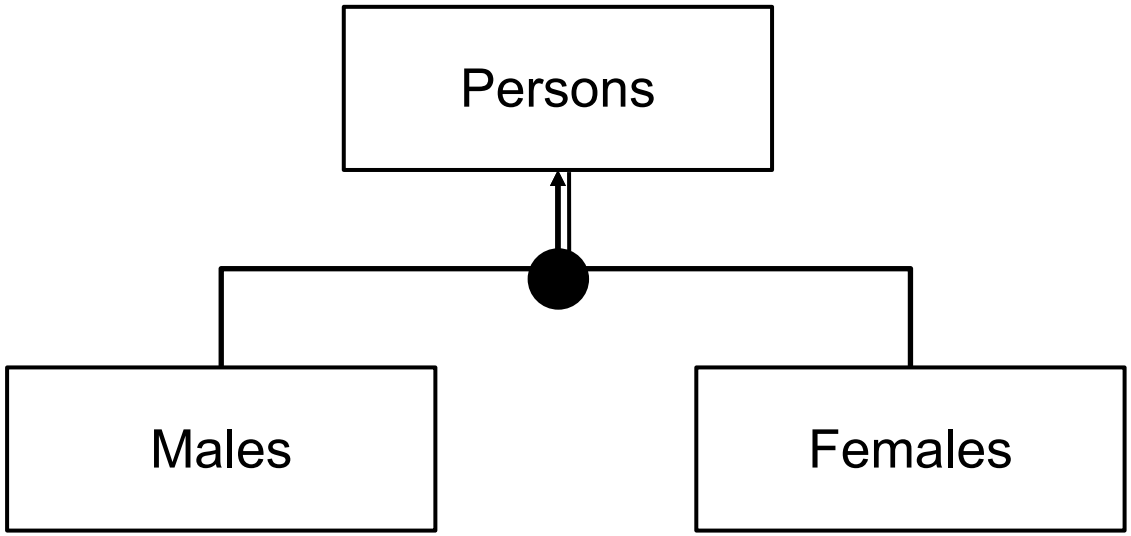
Fiat and Ford are sub-classes of Cars. Each car can be a Ford or a Fiat but not both. Some car may exist which is not a Fiat nor a Ford

Cover subclasses



Programmers and Managers are sub-classes of Employees. Each employee can be a Programmer or a Manager or both. There are no other employees other than programmers and managers

Partition (disjoint union)



Males and Females are sub-classes of Persons. Each person can be Male or Female but not both. A person must be either a male or a female (no other options)

Case study

Suppose that we want to create a data model for our departmental library.

We will define the concrete knowledge of our application domain using the graphical formalism of an entity-relationship diagram.

We will proceed in 3 phases:

1. Requirements analysis
2. Conceptual design
3. Logical design

Requirements analysis

With the requirement analysis we try to define the needs or conditions to meet of our product.

We need to:

- Acquire knowledge on the specific application domain
- Understand and define the terms usually associated to such domain
- List all the functionalities that the new product should have

This step is usually carried out in natural language

Università
Ca' Foscari
Venezia

Requirements analysis

In our case, the information system used by our departmental library should be able to manage a collection of multiple books written by different authors.

Members of the department, which can be either students or professors, are able to borrow paper copies of the books available in the library.

Each book can be borrowed for a limited period of time so the system must keep track of the date on which a book was lent together with any additional information needed to perform the operation.

Conceptual design

With the conceptual design we formalize the requirements expressed in the requirements analysis into a graphical formalism.

In this case we will use the **entity-relationships diagram**.

The conceptual design is carried out in 4 phases in which we define:

1. All the classes in our domain
2. The relations between classes
3. All the existing subclasses (if any)
4. Useful properties of each class

Conceptual design: defining classes

From the analysis we made before we can extract all the entities (classes) involved in our domain. Usually, the nouns are a good hint to identify such entities:

"In our case, the information system used by our departmental library should be able to manage a collection of multiple books written by different authors.

Members of the department, which can be either students or professors, are able to borrow paper copies of the books available in the library.

Each book can be borrowed for a limited period of time so the system must keep track of the date on which a book was lent together with any additional information needed to perform the operation."

Conceptual design: defining classes

From the analysis we made before we can extract all the entities (classes) involved in our domain. Usually, the nouns are a good hint to identify such entities:

*"In our case, the information system used by our departmental library should be able to manage a collection of multiple **books** written by different **authors**.*

***Members** of the department, which can be either students or professors, are able to borrow **paper copies** of the books available in the library.*

Each book can be borrowed for a limited period of time so the system must keep track of the date on which a book was lent together with any additional information needed to perform the operation."

Conceptual design: defining classes

Authors

Books

Department Members

Paper copies

Conceptual design: defining relations

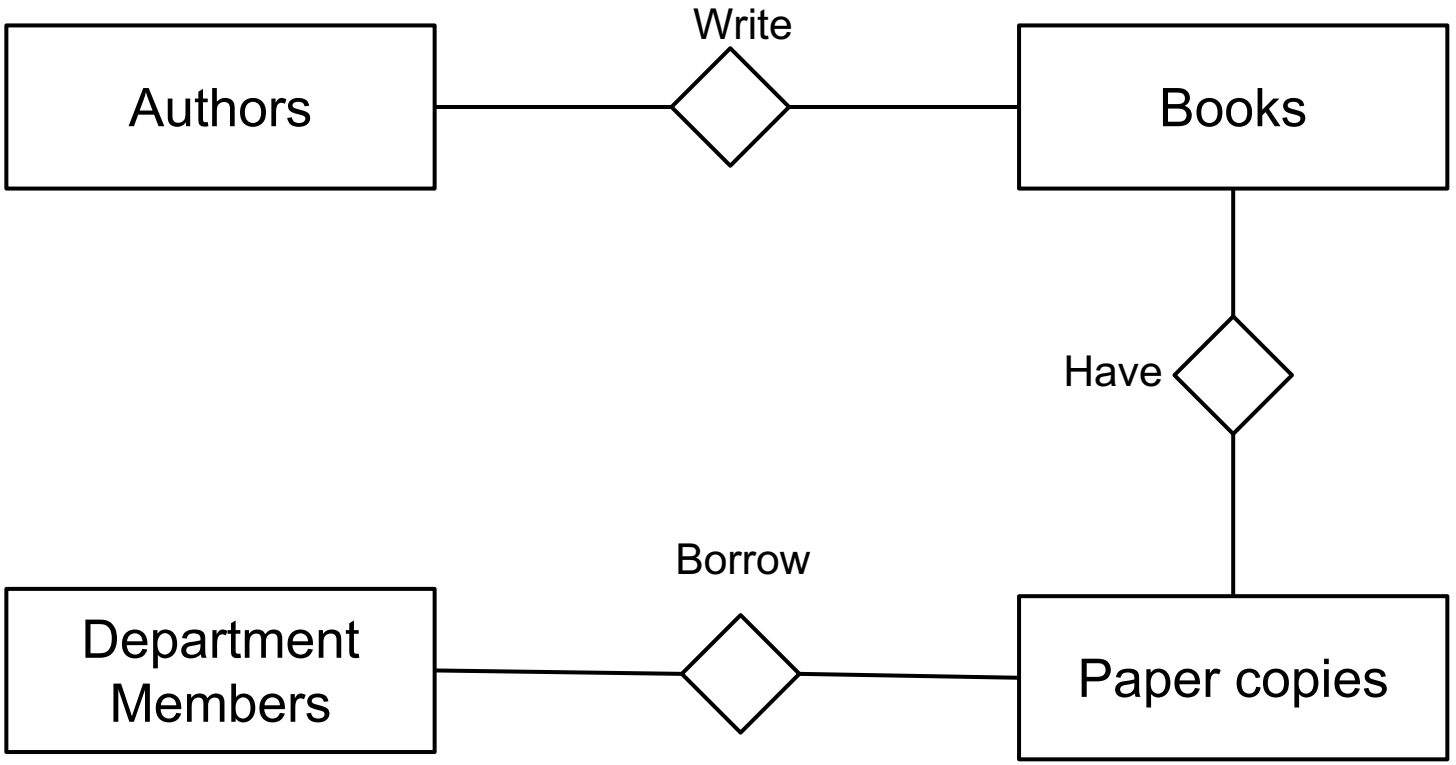
Once important classes (or the entity sets) have been defined, we can proceed to analyze the relations between them.

Steps:

- Give a name to each relation
- Analyze the multiplicity of the relations
- Specify if each relation is optional or mandatory
- Define properties of relations (if any)

Università Ca' Foscari Venezia

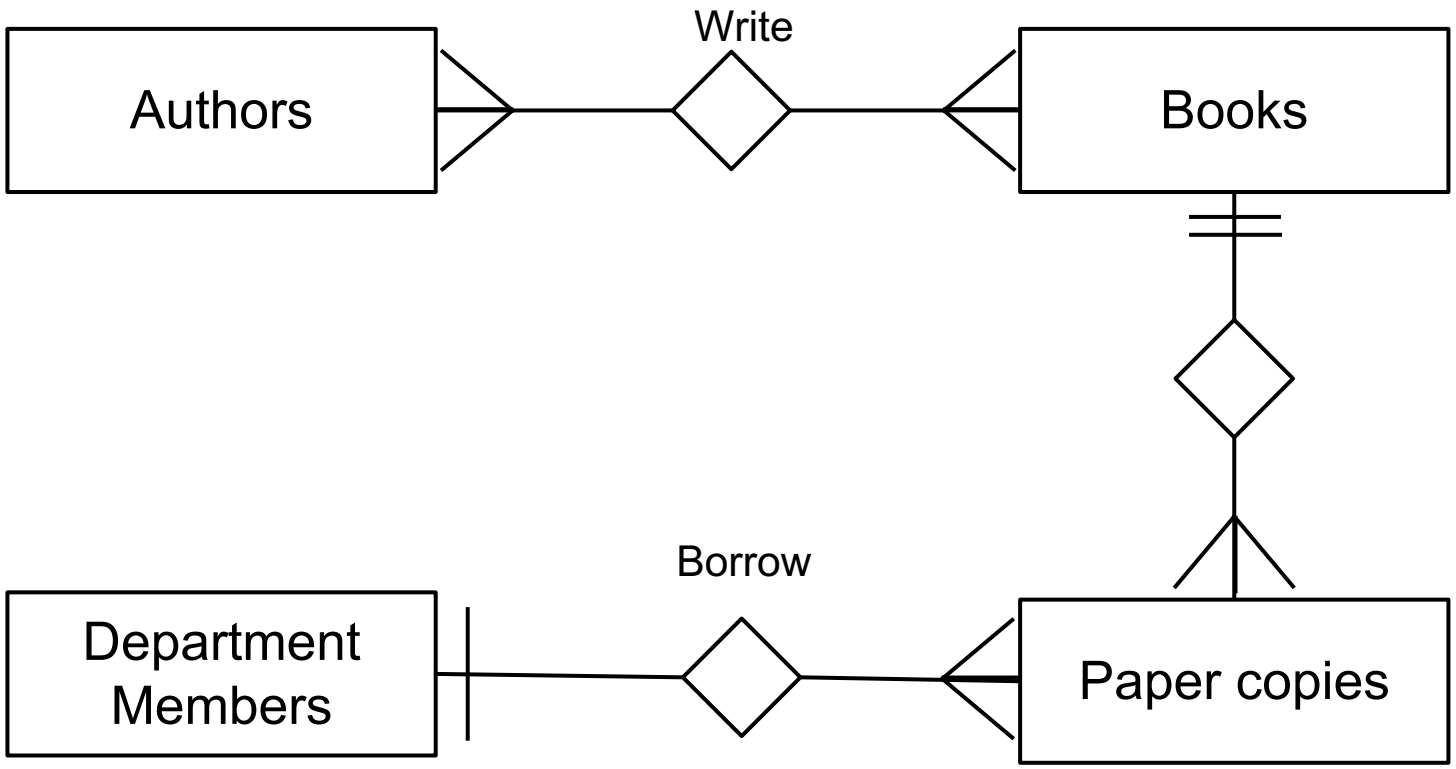
Conceptual design: defining relations



Then, we analyze the multiplicity of the 3 relations we have identified...

Università Ca' Foscari Venezia

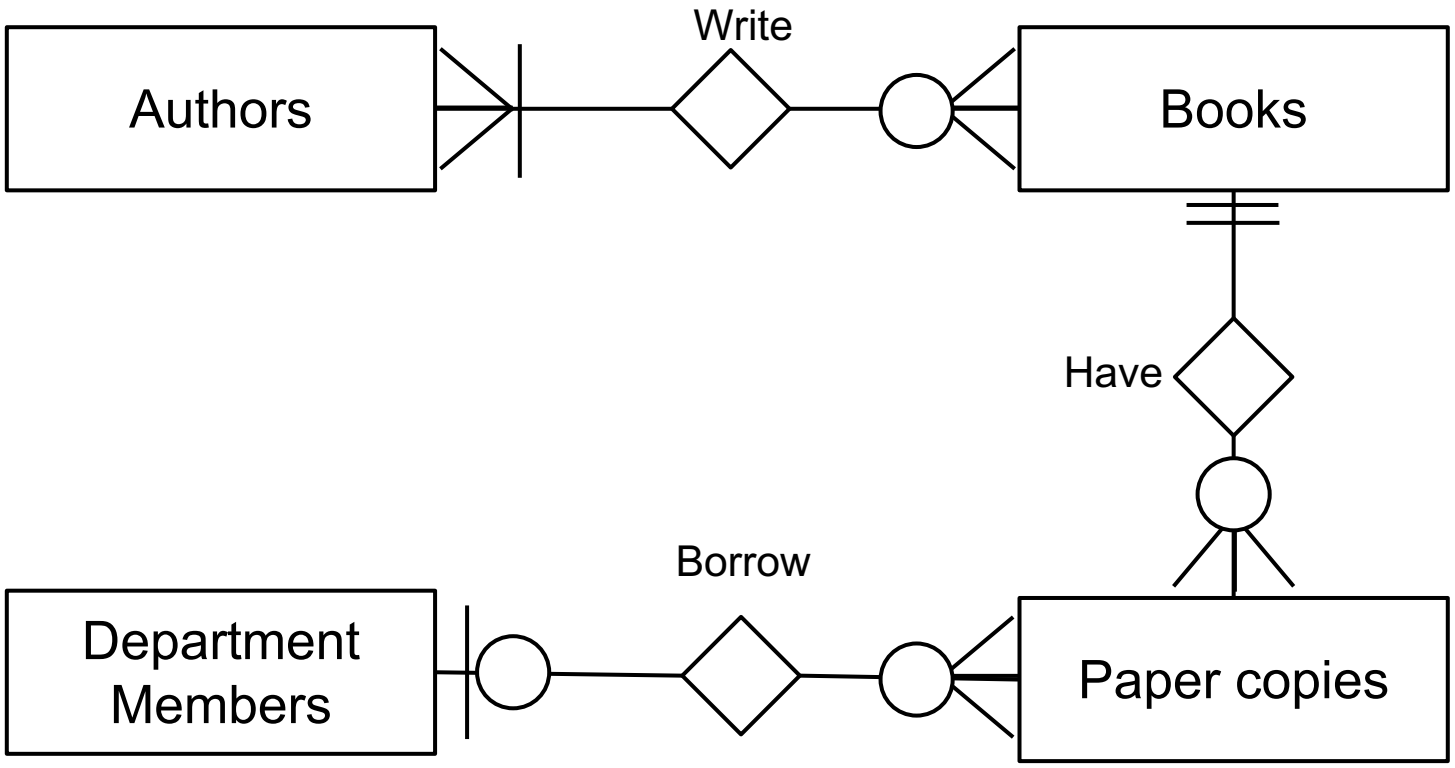
Conceptual design: defining relations



Then, we need to specify if each relation is optional or mandatory...

Università Ca' Foscari Venezia

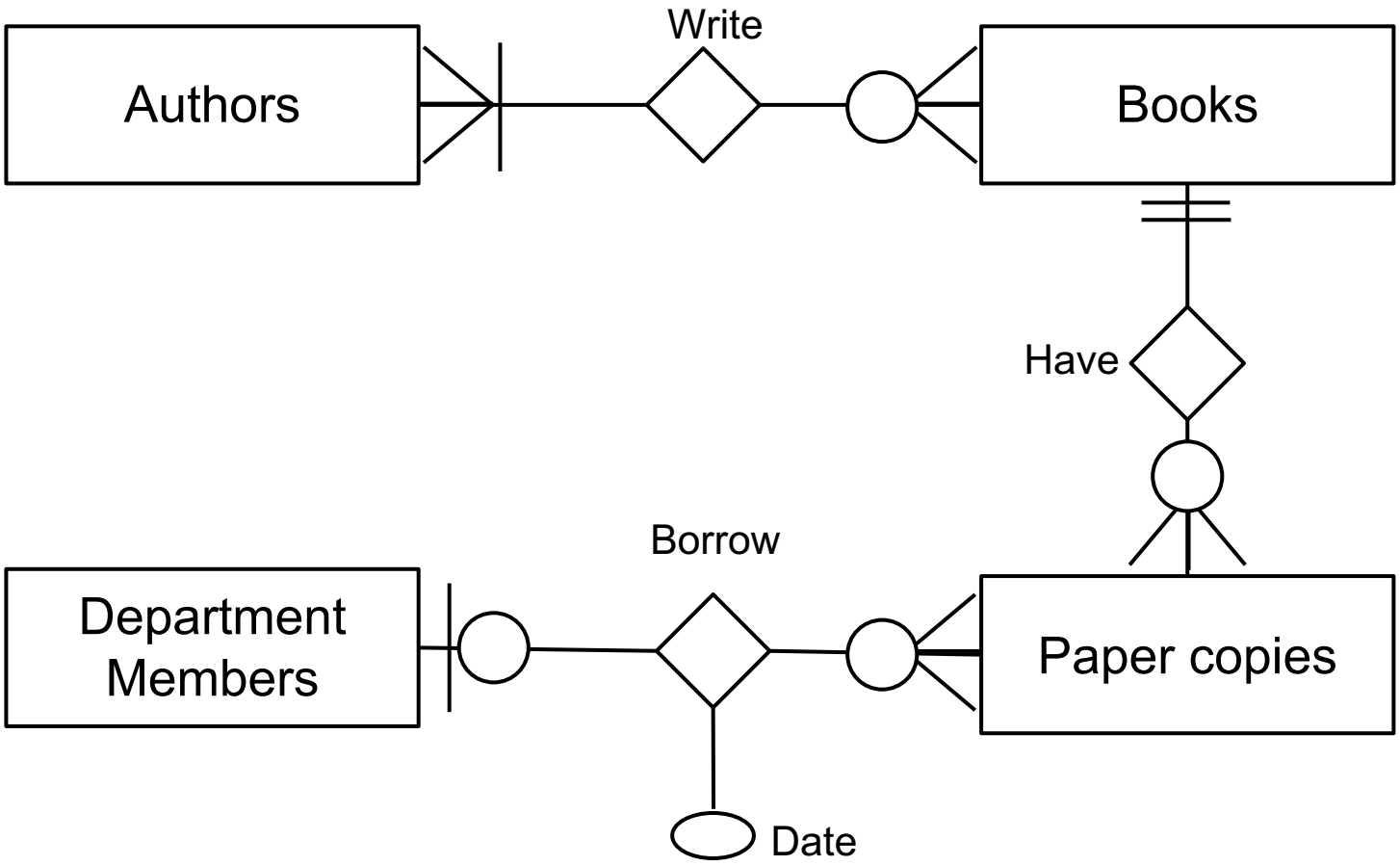
Conceptual design: defining relations



Finally, we look if some of the relations may have properties attached

Università Ca' Foscari Venezia

Conceptual design: defining relations



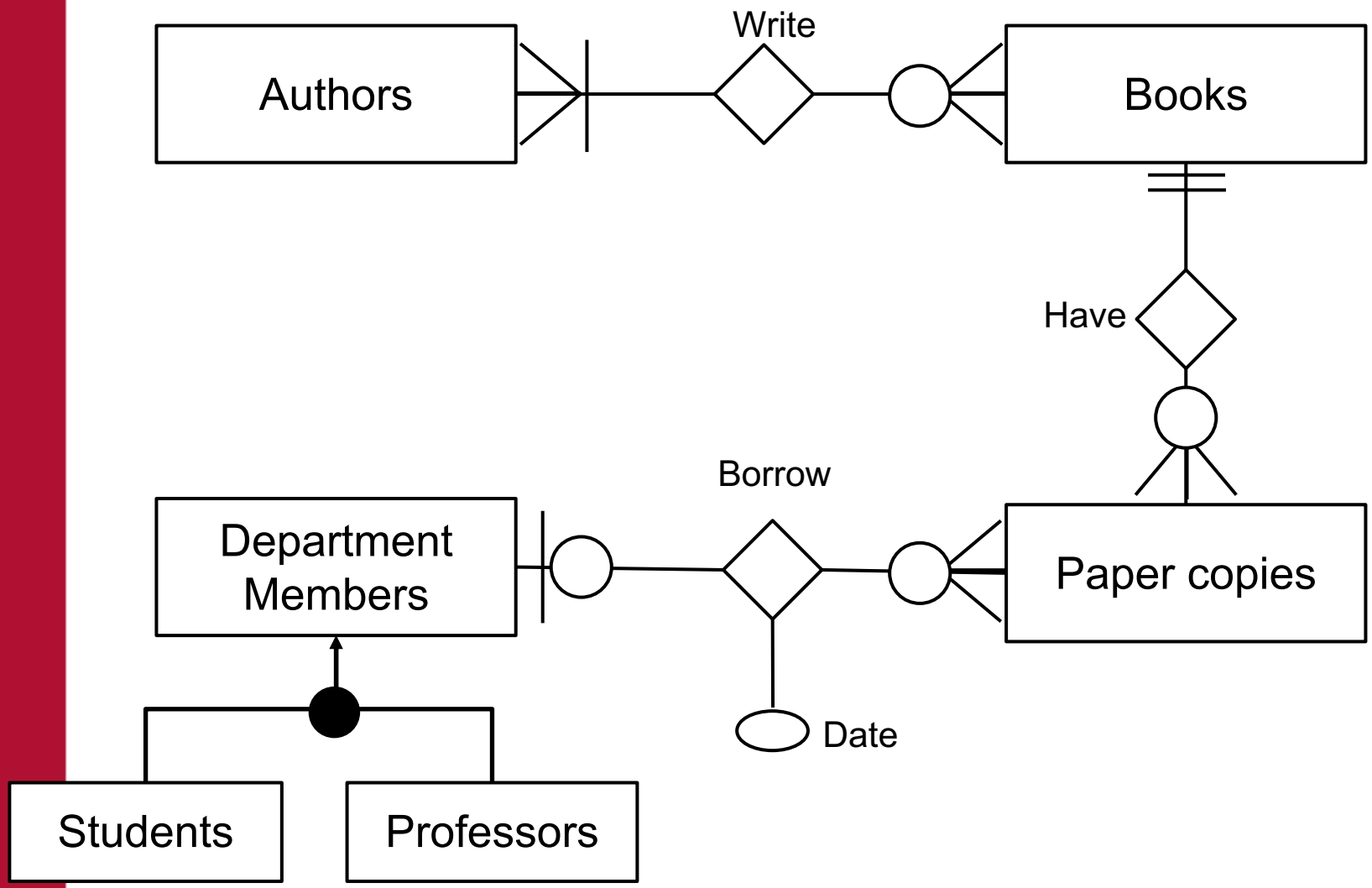
Conceptual design: defining subclasses

We now need to examine all the already defined classes to see if it may be useful to define some new classes in order to:

- Better characterize particular subsets of the existing ones
- Model entities that may change their status during the lifetime of the application (for example a person may change its role from student to professor and acquire different properties)

Università Ca' Foscari Venezia

Conceptual design: defining subclasses



Conceptual design: defining properties

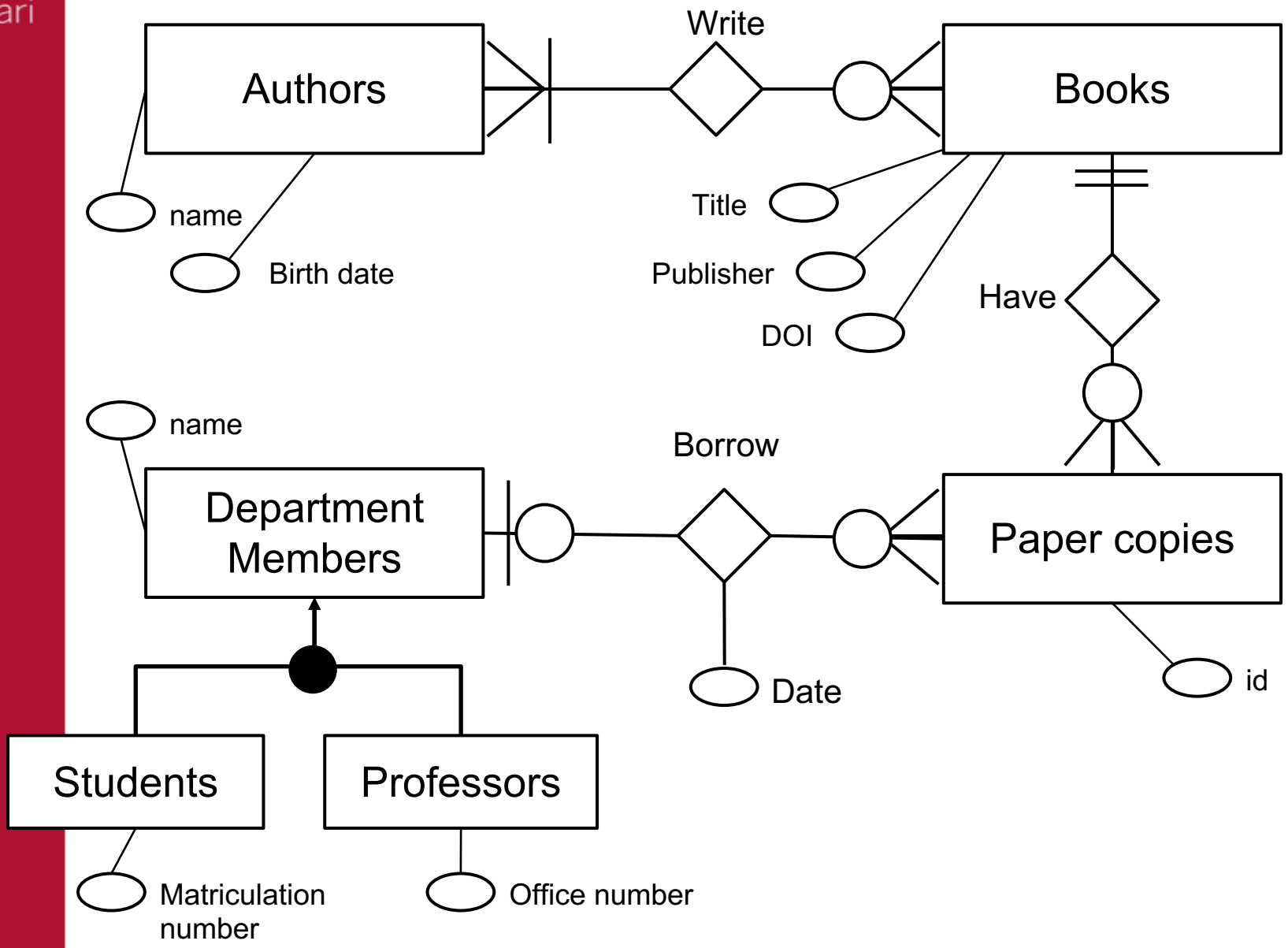
In the last step of our conceptual design we define the properties of all our classes.

Some suggestions:

- Only the relevant (with respect to our application) properties should be listed
- When adding properties, we may discover that a property is so important (with respect to other classes) that we may need go back to the class definition step and add it to our model
- Viceversa, we may discover that some classes may have been modeled as simple properties...

Università Ca' Foscari Venezia

Final e-r diagram for our case study



Keys

Keys are one or more attributes in an entity set such that, given any two entities in the set, they cannot have identical values for each of the attributes in the key. (ie. value of key attributes can be used to identify a specific entity)

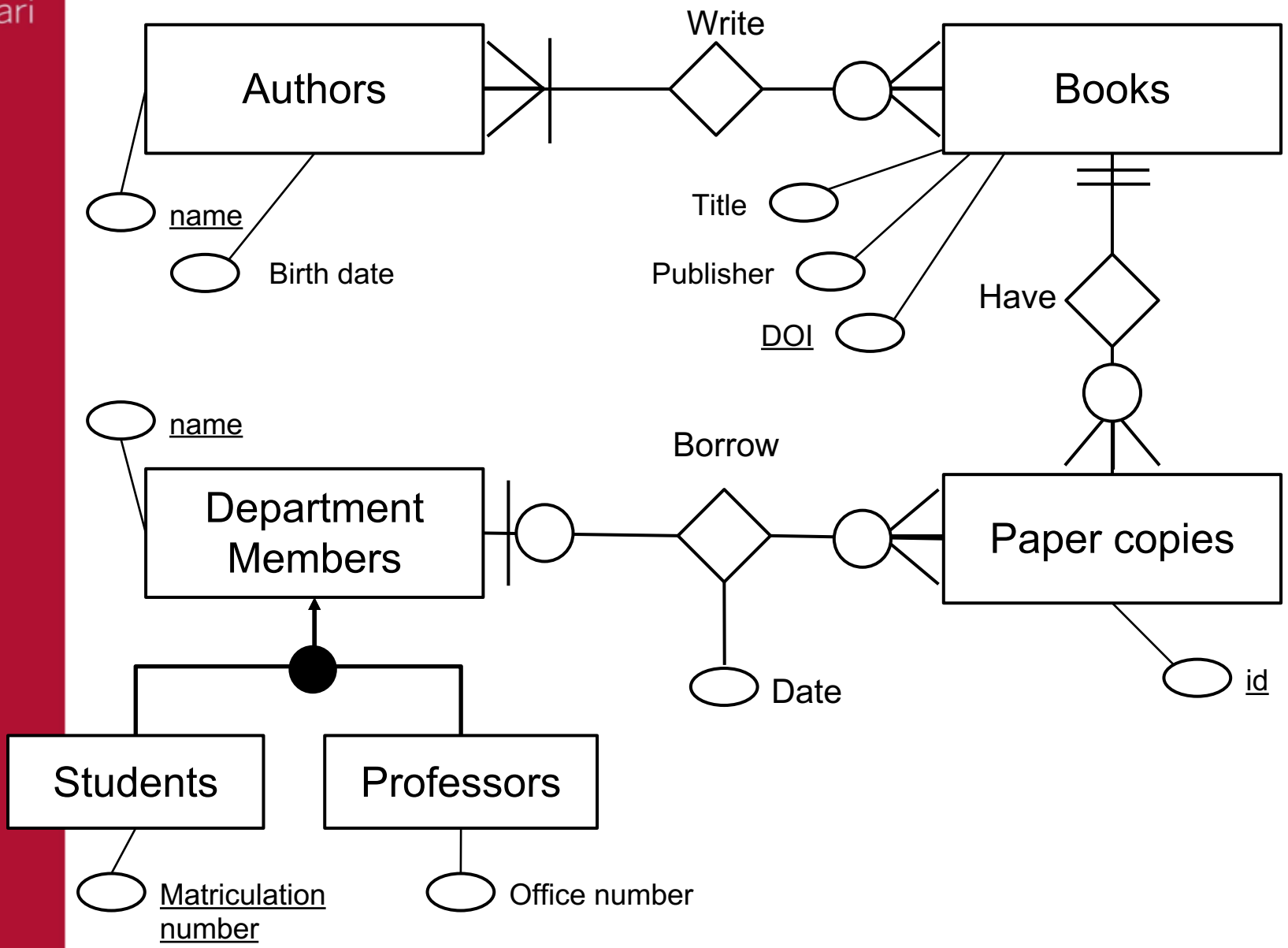
A key is called primary if the value of such attribute is unique among the entity set

In an e-r diagram, we underline key attributes

For example, "matriculation number" is a primary key attribute for the class "Students" because cannot exist two students with the same "matriculation number".

Università Ca' Foscari Venezia

Final e-r diagram for our case study



Logical design

With the logical design we translate our er-diagram to a logical schema expressed in a specific data model that can be implemented in a DBMS.

In particular, we will analyze the popular **relational model** which is implemented today in many **relational databases**

The fundamental building blocks of a relational model are:

- **Tuples**
- **Relations**

Tuples

A tuple is a finite set of pairs in the form

(attribute, value).

For example:

(name, “Filippo”), (surname, “Bergamasco”),
(birth_year, “1985”)

We use tuples to represent entities. In our example, the tuple could be used to represent an entity belonging to the class (entity set) “Persons”

Relations

A relation is a finite set of tuples. Each tuple must have the same "structure" (ie. it must be composed by the same set of attributes).

We use relations to represent classes.

A relation can be naturally represented with a **table**

Name	Surname	birth_year
Filippo	Bergamasco	1985
Mario	Rossi	1963
Giovanni	Verdi	2011

Relations

Name	Surname	birth_year
Filippo	Bergamasco	1985
Mario	Rossi	1963
Giovanni	Verdi	2011

- Each column represent a different attribute
- Each row represent an entity
- Each attribute must have an atomic value
- Primary keys are used to uniquely identify a specific row (entity) in the table

Relational database

In a relational database, all the data is stored by means of **tuples** and **relations**.

In other words, the data is stored in different tables containing all the entities acting in our domain.

How do we convert an e-r diagram into a set of relations (tables)?

- Each class is simply represented by a table, with the column composed by all the attributes of the class
- Each relationship is also represented in a table, in which the key of each entity involved become a new column (**foreign key**) of the table

Università Ca' Foscari Venezia

Relationships in a relational database

Primary key

Students

Matriculation #	Name	Surname	birth_year
071523	Filippo	Bergamasco	1985
067459	Mario	Rossi	1963
079856	Giovanni	Verdi	2011

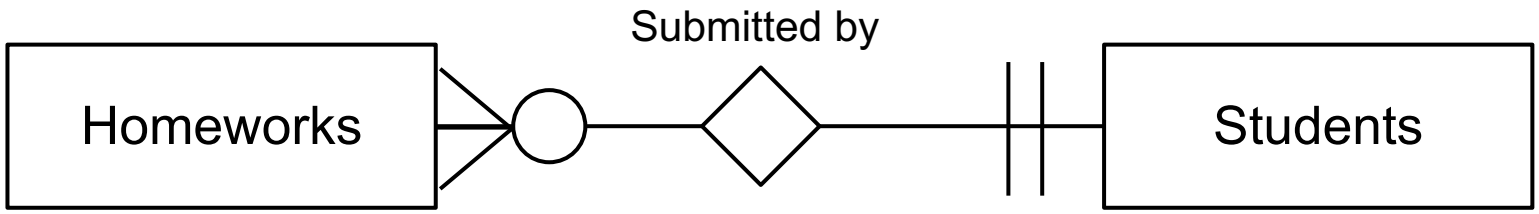
Homeworks

Student #	Date	Evaluation	Subject
071523	12/01/2016	27	Math
071523	04/03/2016	18	Chemistry
067459	04/03/2016	25	Chemistry
079856	21/11/2017	30L	Math

Foreign key

Relationships in a relational database

- One-to-many relations are represented in a relational database by adding a foreign key (referring to the other relation) to the relation for which the association is unique
  - (See the previous example)



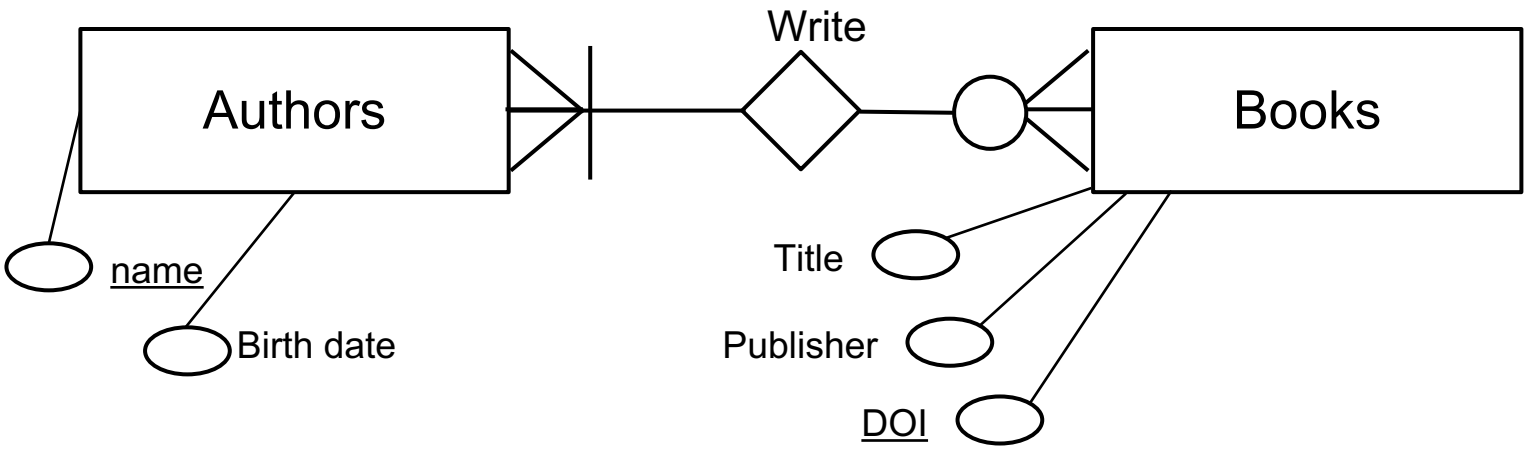
- In One-to-one relationships the foreign key can arbitrarily be put in any of the two relations

Università Ca' Foscari Venezia

Relationships in a relational database

- Many-to-many relations are represented in a relational database **by adding a new relation** containing 2 foreign keys (one for each relation). This new relation will contain a new tuple for each association.

Example:



Relationships in a relational database

Primary keys

Authors

Name	Birth date
Dan Brown	22/06/1964
J.r.r. Tolkien	03/01/1892
Tom Clancy	12/04/1947
Peter Lake	04/06/1964
Paul Crowther	13/11/1976

Books

ISBN	Title	Publisher
0-671-02735-2	Angels&Demons	Corgi Books
0-385-50420-9	The Da Vinci Code	Doubleday
0544003411	The Lord of the Rings	Mariner Books
978-1-4471-5601-7	Concise guide to databases	Springer

AuthorsBooks

Name	ISBN
Dan Brown	0-671-02735-2
J.r.r. Tolkien	0544003411
Dan Brown	0-385-50420-9
Peter Lake	978-1-4471-5601-7
Paul Crowther	978-1-4471-5601-7

Foreign keys

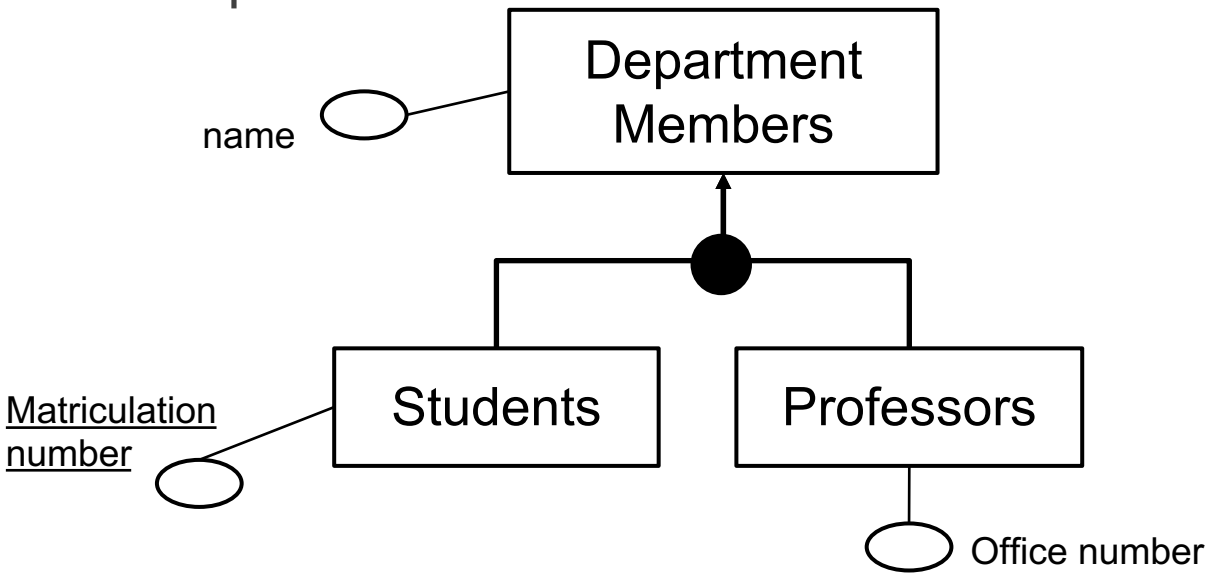
Università Ca' Foscari Venezia

Sub-classes in a relational database

Two alternative ways to represent sub-classes:

1. With a single relation containing both the attributes of each relation. An additional attribute is used to discriminate the class type
2. A new relation is created for each sub-class, with a foreign key pointing to the super-class.

Example:



Sub-classes in a relational database

First method: An attribute discriminates the class

(If an attribute does not exist for a specific class, its value will be "null")

Name	Birth date	Matriculation number	Office number	Is student
Mario rossi	12/03/85	057621	**null**	true
Luca Verdi	11/04/91	**null**	12	false
Giovanni Neri	14/11/90	7812319	**null**	true

Sub-classes in a relational database

Second method: A new relation is created for each sub-class

Department Members

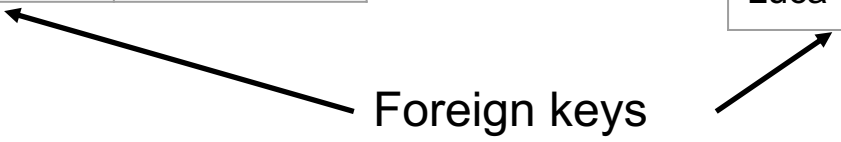
Name	Birth date
Mario rossi	12/03/85
Luca Verdi	11/04/91
Giovanni Neri	14/11/90

Students

Name	Matriculation number
Mario rossi	057621
Giovanni Neri	7812319

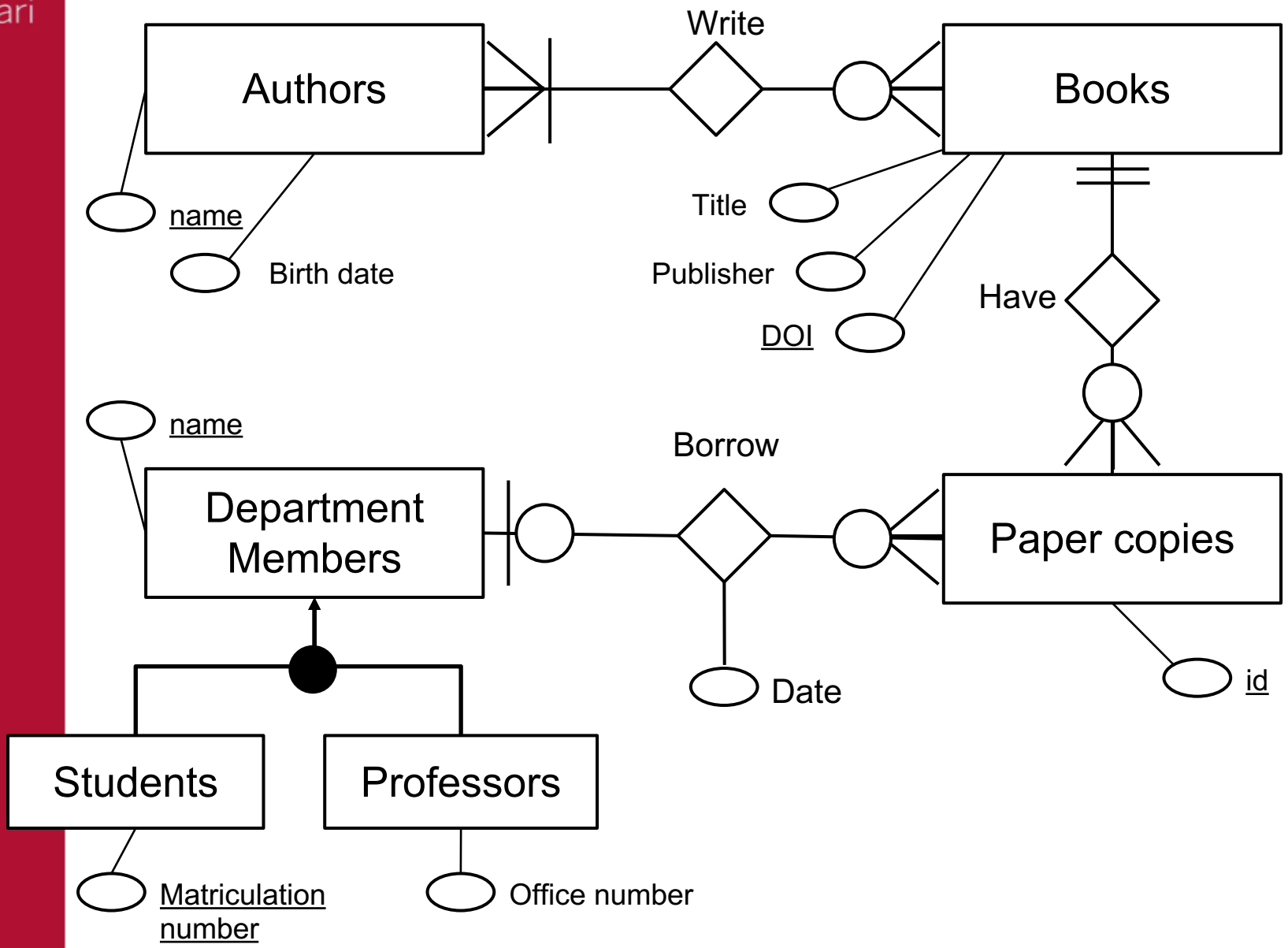
Professors

Name	Office number
Luca Verdi	12



Università Ca' Foscari Venezia

Back to our previous example...



Tables in a relational database

Authors(name: pk, birth_date)

Books(doi: pk, title, publisher)

DepartmentMembers(name: pk)

Students(name: fk to DepartmentMembers, matriculation_number: pk)

Professors(name: fk to DepartmentMembers, office_number)

AuthorsBooks(name: fk to Authors, DOI: fk to Books)

PaperCopies(id: pk, name: fk to DepartmentMembers, doi: fk to Books)

Querying for data

How do we insert, update, delete and search (CRUD) data inside a relational database?

SQL: Structured Query Language

Introduced in 1986, SQL is based on relational algebra and tuple relational calculus.

- Declarative language
- Contains structures for data insert, query, update and delete, schema creation and modification, and data access control
- Advantages:
  - Can access multiple records with a single command
  - No need to specify "how to reach a record"

SQL: Some examples

Creating tables:

```
CREATE TABLE Students (
      Name  CHAR(30) PRIMARY KEY,
      Address VARCHAR(255),
      Birthdate  DATE )
```

Deleting tables:

```
DROP TABLE Students
```

Inserting values:

```
INSERT INTO Students VALUES ('Filippo',
'Via Falsa 123', '18/09/1985')
```

SQL: Retrieving data

The power of a relational database lies in the ability to easily search for data which meets a particular condition.

This condition may require data to be retrieved from one or more tables

The SELECT statement:

```
SELECT <attributes>
FROM <list of tables>
WHERE <conditions>
```

SQL: Retrieving data

The SELECT command will **return a new table** with the data extracted from the database according to the various clauses of the command

Examples:

```
SELECT *
FROM Students
```

Returns a new table containing all the data of the table Students (all the rows and all the attributes)

SQL: Retrieving data

```
SELECT Name, Matriculation_number
FROM Students
```

Returns a new table containing all the rows of the table Students but only the columns “Name” and “Matriculation_number”

In relational algebra this operation is called **projection**:

Produce from a relation R a new relation that has only some of R's columns

SQL: Retrieving data

```
SELECT Name, Matriculation_number
FROM Students
WHERE BirthYear > 1985
```

Returns a new table containing all the Students having the BirthYear attribute greater than 1985 and keep only the Name and Matriculation_number columns

In relational algebra this operation is called **selection**:

Produce a new relation with a subset of R's tuples according to some true/false condition

SQL: Joins

Suppose that we have the following tables we have seen before.

Students

Matr_num	Name	Surname	birth_year
071523	Filippo	Bergamasco	1985
067459	Mario	Rossi	1963
079856	Giovanni	Verdi	2011

Homeworks

Student_num	Date	Evaluation	Subject
071523	12/01/2016	27	Math
079856	04/03/2016	18	Chemistry
071523	21/11/2017	30L	Chemistry

How do we enumerate the homeworks made by each student?

SQL: Joins

Two operations:

1. Compute the cartesian product between Students and Homeworks
   - Ie. Create a table having the union of the attributes of Students and Homeworks as columns and every combination of Students and Homeworks as rows

1. Eliminate the rows for which the foreign key does not match the corresponding primary key
   - This is performed with the where clause

Università Ca' Foscari Venezia

SQL: Joins

1. Compute the cartesian product:

Matr_num	Name	Surname	Student_num	Date	Evaluation	Subject
071523	Filippo	Bergamasco	071523	12/01/2016	27	Math
071523	Filippo	Bergamasco	079856	04/03/2016	18	Chemistry
071523	Filippo	Bergamasco	071523	21/11/2017	30L	Chemistry
067459	Mario	Rossi	071523	12/01/2016	27	Math
067459	Mario	Rossi	079856	04/03/2016	18	Chemistry
067459	Mario	Rossi	071523	21/11/2017	30L	Chemistry
079856	Giovanni	Verdi	071523	12/01/2016	27	Math
079856	Giovanni	Verdi	079856	04/03/2016	18	Chemistry
079856	Giovanni	Verdi	071523	21/11/2017	30L	Chemistry

SQL: Joins

2. Eliminate the lines for which the foreign key does not match the corresponding primary key

Matr_num	Name	Surname	Student_num	Date	Evaluation	Subject
071523	Filippo	Bergamasco	071523	12/01/2016	27	Math
071523	Filippo	Bergamasco	079856	04/03/2016	18	Chemistry
071523	Filippo	Bergamasco	071523	21/11/2017	30L	Chemistry
067459	Mario	Rossi	071523	12/01/2016	27	Math
067459	Mario	Rossi	079856	04/03/2016	18	Chemistry
067459	Mario	Rossi	071523	21/11/2017	30L	Chemistry
079856	Giovanni	Verdi	071523	12/01/2016	27	Math
079856	Giovanni	Verdi	079856	04/03/2016	18	Chemistry
079856	Giovanni	Verdi	071523	21/11/2017	30L	Chemistry

SQL: Joins

```
SELECT Name,Surname,Subject,Evaluation
FROM Students, Homeworks
WHERE Students.matr_num =
Homeworks.student_num
```

In the relational algebra the operation is called “JOIN” and is realized as a combination of cartesian product and selection

So frequent that there exist dedicated keywords to automatically perform the JOIN operation without explicitly match the keys in the WHERE clause

SQL: Joins

```
SELECT Name,Surname,Subject,Evaluation
FROM Students, Homeworks
WHERE Students.matr_num =
Homeworks.student_num
```

Name	Surname	Subject	Evaluation
Filippo	Bergamasco	Math	27
Filippo	Bergamasco	Chemistry	30L
Giovanni	Verdi	Chemistry	18

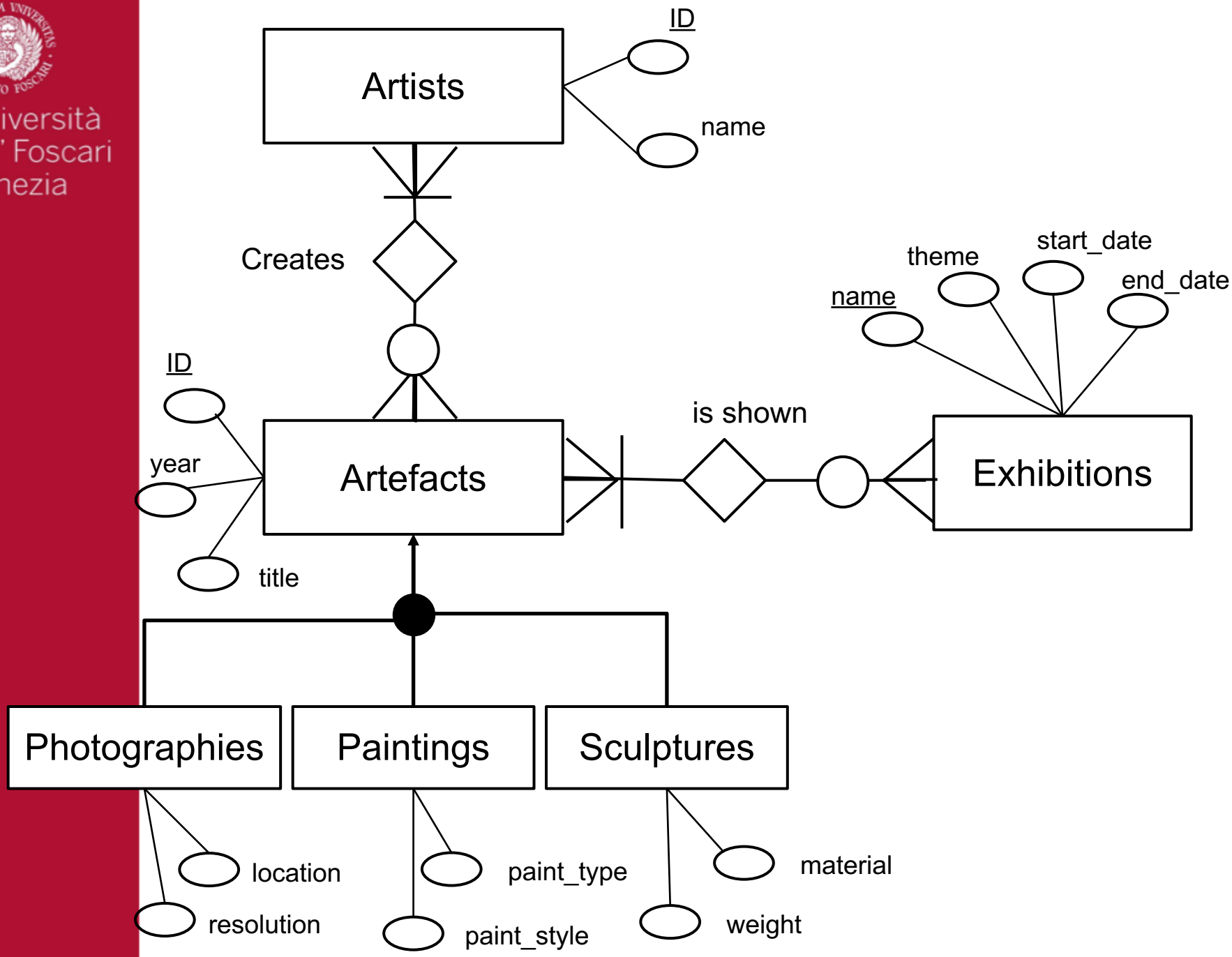
Exercise

We want to create the data model of a typical **museum** and implement it using a relational database

Requirement analysis:

The information system to be developed should manage all the artefacts of our museum. An artefact is a piece of art created by some artist. Each artefact is characterized by a title and the production year. Three kind of artifacts are managed by the museum: sculpture, painting and photography.

Sometimes, the museum organizes an exhibition with a specific theme to show a collection of artefacts to the general public


Artists
ID
name
Creates
Artefacts
ID
year
title
is shown
Exhibitions
name
theme
start_date
end_date
Photographies
location
resolution
Paintings
paint_type
paint_style
Sculptures
material
weight

Tables in a relational database

Artists(id: pk, name)

Artefacts(id: pk, year, title)

Exhibitions(name: pk, theme, start_date, end_date)

Photographies(id: fk to Artefacts,location, resolution)

Paintings(id: fk to Artefacts, paint_type, paint_style)

Sculptures(id: fk to Artefacts, material, weight)

ArtefactsExhibitions(id: fk to Artefacts, name: fk to Exhibitions)

ArtefactsArtists(id_artefact: fk to Artefacts, id_artist: fk to Artists)