DIUM, University of Udine

Basi di dati

Entity-relationship model

Andrea Brunello
andrea.brunello@uniud.it

The process of designing a database is typically articulated into different phases, that separate *what* to represent from *how* to represent it:

1. **Conceptual design**
2. **Logical design**
3. **Physical design**

Conceptual design

- The initial phase of database design is to characterize fully the data needs of the prospective database users
- Such needs are encoded into a conceptual schema, that acts as a connecting point between stakeholders and IT personnel
- In this course, for the development of the conceptual schema, we will consider the **Entity-Relationship** model, which makes use of *diagrams* to represent the overall logical structure of a database graphically
- Such diagrams are typically complemented by free text annotations, that describe the requirements that could not be encoded in the diagram

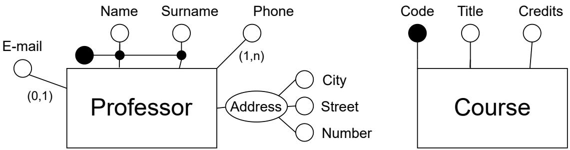
Entity-Relationship model

- The Entity-Relationship model describes a domain by means of:
  - Entity sets
  - Relationship sets
  - Attributes
- There are several ways in which entity sets, relationship sets and attributes can be represented in E-R diagrams
- We will consider the notation proposed in the book:

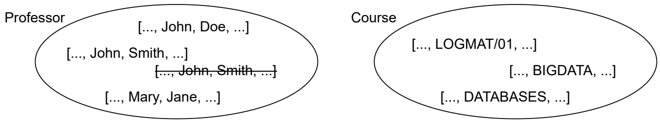
Database Systems - Concepts, Languages and Architectures, P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone

Entity sets

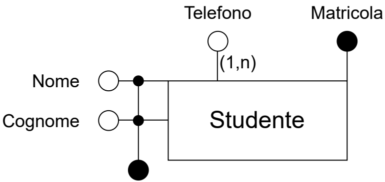
- An **entity** is an object that exists in the considered domain and is distinguishable from other objects
  - Example in the University domain: a specific professor, student, or course
- An **entity set** is a set of entities of the same type that share the same properties
  - Example: the set of all professors, each characterized by a name, a surname, and a salary
- Such properties, possessed by all members of an entity set, are represented by **attributes**
- Each attribute can take on values of a given domain
- A subset of the attributes forms a **key** of the entity set, i.e, knowing the values of such attributes, it is possible to uniquely identify a member (entity) of the entity set



- Entity set *Professor* has the attributes:
  - *E-mail*, optional
  - *Name* and *Surname*, that together make the key
  - *Phone*, multi-valued
  - *Address*, composite, and made of *City*, *Street*, *Number*
- Entity set *Course* has the attributes:
  - *Code*, that makes the key
  - *Title*
  - *Credits*

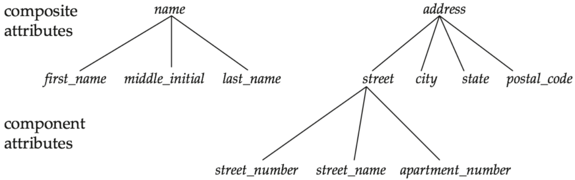


- An entity set cannot contain two entities with the same values on the key attributes
  - Here: name and surname for Professor, code for Course
- Observe that, being a set, each entity set has a *trivial* key composed of all the attributes
- Finally, note that, in general, an entity may have more than one key (it really depends on the considered domain)



- Entity set Student has two possible keys
- Spoiler: one key will be chosen as the primary key in the relational schema
- Thus, in our domain, we assume that *name* and *surname* are enough to discriminate between students (no homonyms are possible)

- Composite attributes can be nested further

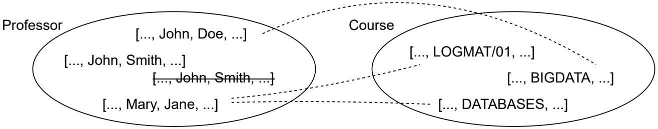


- The characteristics of attributes can be combined, e.g., there can be multi-valued composite attributes, as well as optional composite attributes

- A **relationship** is an association among several entities
  - E.g., *Professor: [John, Doe]* may be linked with *Course: [BIGDATA]* by means of a relationship *Teaches*
  - The relationship is an association between actual entities

- A **relationship set** $R \subseteq E_1 \times E_2 \times \cdots \times E_n$ is a mathematical relation among $n \geq 2$ entities (e_i), each taken from its entity set (E_i)
  - $R = \{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$
  - where $(e_1, e_2, \ldots, e_n)$ is a *relationship*
  - note that we cannot have repeated entries (being R a set)

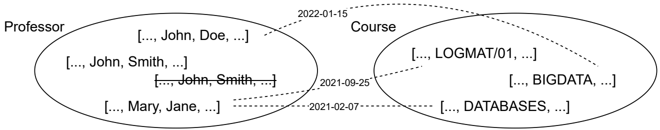
- For instance, $([John, Doe], [BIGDATA]) \in Teaches$ is a relationship belonging to relationship set *Teaches*



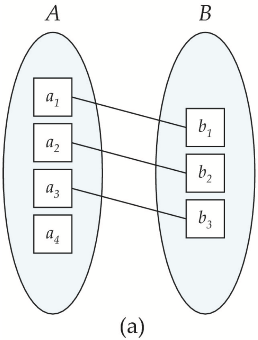
- The picture shows three relationships (dashed lines), which we assume belonging to relationship set *Teaches*:
  - *John Doe* is associated to *BIGDATA*, $([John, Doe], [BIGDATA])$
  - *Mary Jane* is associated to *LOGMAT/01*, $([Mary, Jane], [LOGMAT/01])$
  - *Mary Jane* is associated to *DATABASES*, $([Mary, Jane], [DATABASES])$

Relationships and attributes

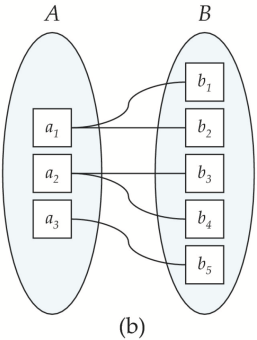
- Attributes may also be associated with relationships, e.g., to track when a professor started teaching a course
- Intuitively, a relationship thus becomes as follows:
  - $([John, Doe], [BIGDATA], 2022-01-15)$
- Still, the relationship is “defined” by the two entities that are linked, other attributes are “accessory” information. Cannot have both:
  - $([John, Doe], [BIGDATA], 2022-01-15)$
  - $([John, Doe], [BIGDATA], 2023-01-20)$



- Given entity sets E_1, E_2, and a relationship set R between them, entities may participate into relationships in different ways, for instance:
  - An entity of E_1 may not participate to any relationship with an entity of E_2 (a prof. may not be teaching any courses)
  - An entity of E_1 may participate to > 1 relationships with an entity of E_2 (a prof. may be teaching more than one course)
- In general, for a binary relationship set we can identify the following **cardinalities**:
  - One to one
  - One to many
  - Many to one
  - Many to Many
- Moreover, entity sets may have a *partial* or *total* **participation** to a relationship set

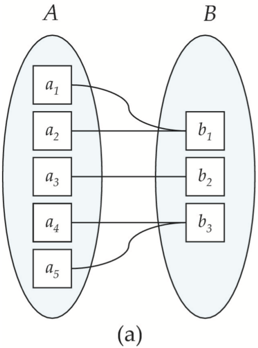


One to one

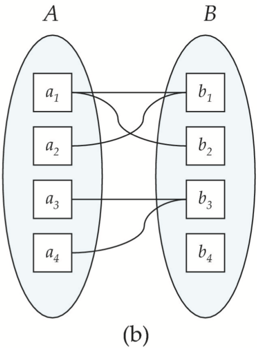


One to many

- Left: A has a partial participation, while B total
- Right: both A and B have a total participation

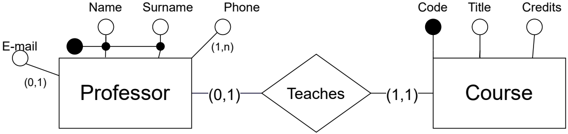


Many to one

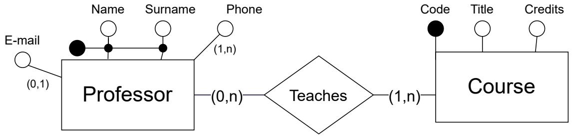


Many to many

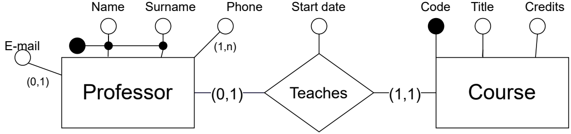
- Left: both *A* and *B* have a total participation
- Right: *A* has a total and *B* has a partial participation



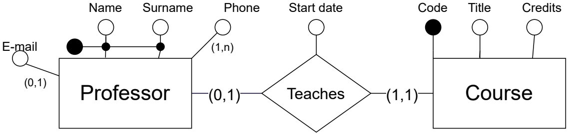
- A professor teaches in at most one course: (0,1)
- A course is taught by one and only one professor: (1,1)
- (1,1) is assumed by default, and can be omitted
- Concerning the relationship set *Teaches*, this means that:
  - A given entity $p_1 \in \mathit{Professor}$ can appear in at most one relationship $(p_1, c) \in \mathit{Teaches}$ where $c \in C$
  - A given entity $c_1 \in \mathit{Course}$ appears in exactly one relationship $(p, c_1) \in \mathit{Teaches}$ where $p \in P$



- A professor teaches in zero or more courses: (0,n)
- A course is taught by one or more professors: (1,n)
- Concerning the relationship set *Teaches*, this means that:
  - A given entity $p_1 \in Professor$ can appear in any number of relationships $(p_1, c) \in Teaches$ where $c \in C$
  - A given entity $c_1 \in Course$ appears in at least one relationship $(p, c_1) \in Teaches$ where $p \in P$
- Observe that, typically, just the values 0, 1, n are used. We do not specify a number, e.g., 4, with this notation

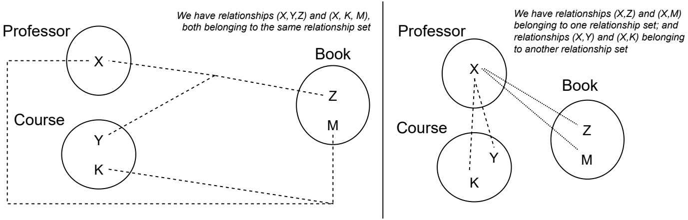


- Relationship sets can also have attributes
- This is quite natural when the attribute tells us something *regarding the specific entities involved* in the relationship
- For instance, here we track the date in which a given professor started teaching a given course
- Here, can we keep track of evidences of a professor teaching the same course in different years?

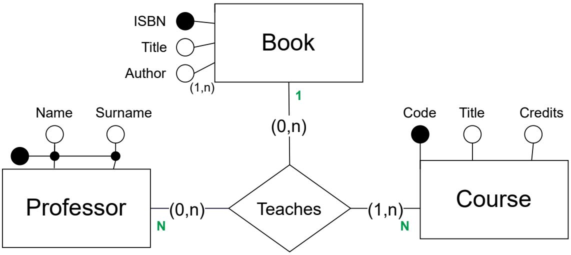


- Relationship sets can also have attributes
- This is quite natural when the attribute tells us something *regarding the specific entities involved* in the relationship
- For instance, here we track the date in which a given professor started teaching a given course
- Here, can we keep track of evidences of a professor teaching the same course in different years?
  - **No**. We are going to see the *reification* process.

- Sometimes, it is necessary to involve more than 2 entities in a relationship (though in a DB most of them are binary, and nearly always at most ternary)
- For instance:
  - *professor X teaches course Y using the book Z*
  - *professor X teaches course K using the book M*



- Observe how the relationships on the right convey less information than those on the left



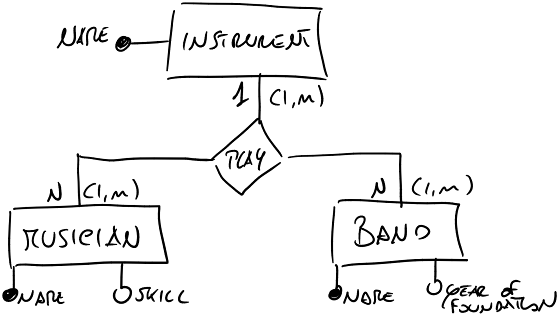
- Participation/cardinality constraints still apply to ternary relationships, although things complicate a little bit
- Here: a professor may teach zero or more courses, each course is taught by at least one professor, and a book is used within zero or more teaching activities, although a professor uses only a book within the same course

- If I have an entity of *Professor*, how many entities of *Book* and *Course* together can be associated to it?
  - Here, the answer is zero or more, so entity set *Professor* participates with $(0, n)$ to *Teaches*

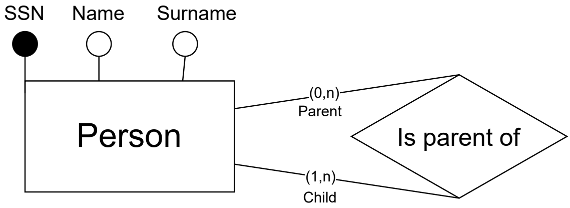
- As for the green numerosities, they allow us to solve some ambiguity. Without them, we wouldn't know if a professor may appear in more than one relationship because:
  - He may teach more than one course using the same book
  - He may teach only one course but using multiple books
  - He may teach multiple courses using multiple books

- Setting the green 1 close to *Book*, we say that, given a *Professor* and a *Course*, there can be only a *Book* associated to them both. On the contrary, given a *Professor* and a *Book*, we may have more courses (green *N* close to *Course*)

- Let us consider the following scenario about bands, musicians, and kind of instruments, that should be modeled using a ternary relationship
- A musician is identified by his/her name, and characterized by his/her skill level
- A band is identified by its name, and characterized by a genre and a year of foundation
- A kind of instrument is identified by its name (e.g., violin, piano, tenor saxophone, etc)
- Each musician can play in one or more bands, and a band has, in general, one or more musicians
- A musician can only play a single kind of instrument in a given band; however, in a band, there can be more than one musician playing the same kind of instrument



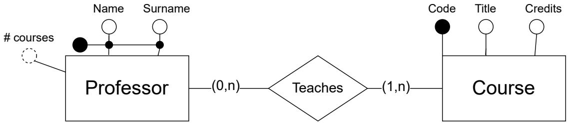
Note: if there is no necessity of modeling *Instrument* as an entity on its own, it is enough to employ just a many-to-many binary relationship *Play* between *Musician* and *Band*, with an attribute *Instrument* (cannot do that if the 1 below *Instrument* is replaced with an *N*, no multi-valued attributes on relationships)



- Entity sets of a relationship need not be distinct
  - For instance, consider the case in which we want to track the prerequisites of a given course
- Each occurrence of an entity set plays a “role” in the relationship
- In that case, it is useful to write down the **roles** as labels on the arcs

Derived attributes

- A derived attribute is an attribute whose value can be computed from other information
- For instance, the number of courses taught by a professor can be derived from the number of relationships of relationship set *Teaches* he/she participates to
- The derived attribute is depicted with a dashed circle
- The E-R diagram should include notes, written in natural language, on how to calculate the derived attributes contained in it
  - *"#courses of an entity of Professor can be calculated retrieving the number of relations Teaches in which such an entity participates"*



Weak entity sets

- Consider the following scenario:
  - We want to keep track of hotels, each characterized by a name
  - There may be multiple hotels with the same name in different cities, however, given a city, there cannot be two hotels with the same name
  - Each city is univocally identified by a code

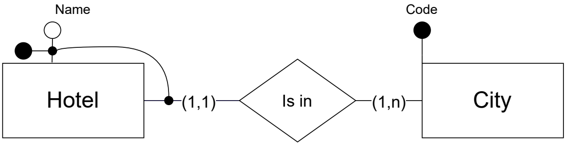
- Intuitively, to univocally identify a hotel, its name is not enough; we also need to know the city where it is

- Similar scenario, pertaining to a cinema franchise domain: cinema hall number within a given a multiplex cinema

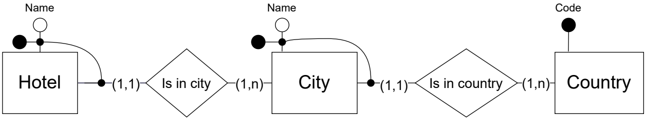
- The notion of **weak entity set** formalizes the above intuition. A weak entity set is one whose existence is dependent on another entity, called its **identifying entity**; instead of associating a key with a weak entity, to uniquely identify it we use the identifying entity, along with extra attributes called **partial keys**

- Each weak entity must be associated with *a corresponding* identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set. The identifying entity set is said to **own** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**

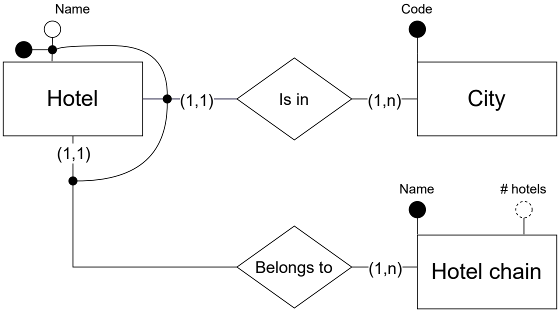
- Observe the $(1,1)$ cardinality on the *Hotel* side



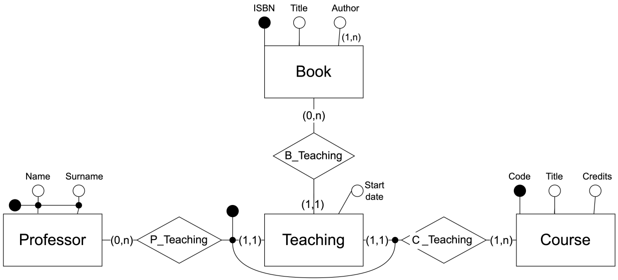
- We may also define chains of weak entity sets

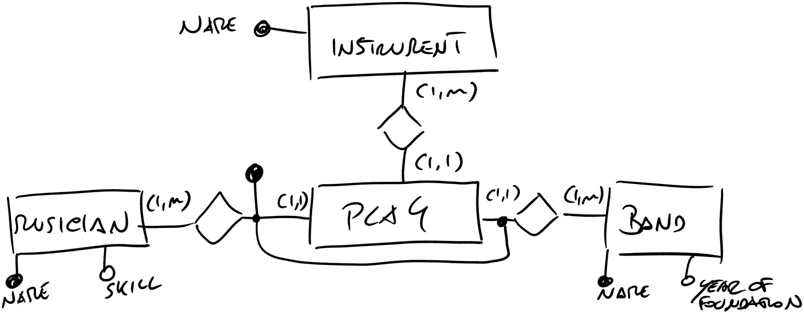


- A weak entity set may depend on more than one identifying entity



- Through a *reification* process we can replace a ternary relationship set with an entity set and 3 relationship sets
- The obtained representation is equivalent
- Observe that the introduced entity set is weak with respect to two entity sets: what if the professor can use more than one book in a course?

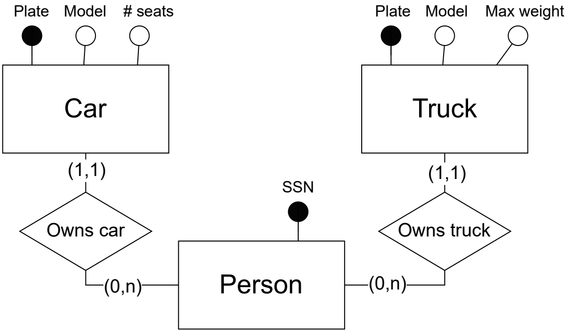




What if. . .

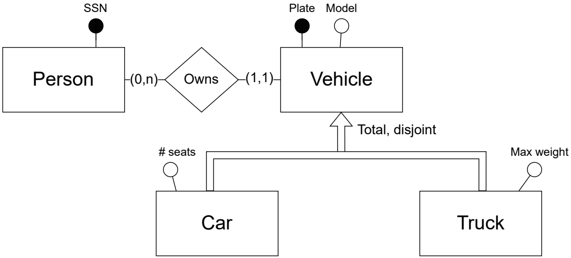
- *Play* is weak w.r.t. *Instrument*, *Musician*, *Band*?
- *Play* is weak w.r.t. *Instrument* and *Musician*?
- *Play* is weak w.r.t. *Instrument*, *Band*?

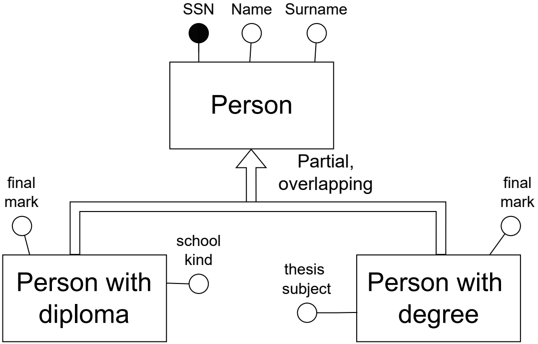
- It may happen that different entity sets share some commonalities regarding their attributes or relationships
  - E.g., a *Truck* and a *Car* may both be considered as a *Vehicle*



Specialization (cont.)

- The **specialization** construct allows us to handle such a scenario
- A “child” entity set inherits all the attributes and relationship participations of the “parent” entity set which it is linked to
- Each occurrence of a child entity is also an occurrence of the parent entity





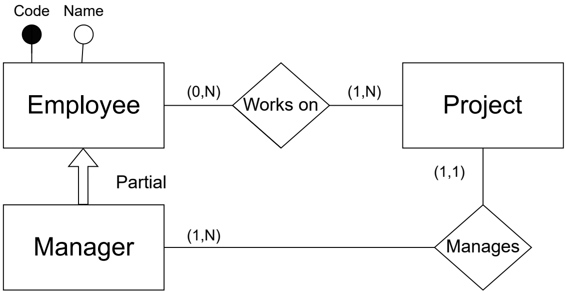
SSN
Name
Surname
Person
Partial,
overlapping
final mark
school kind
Person with diploma
thesis subject
Person with degree
final mark

Specialization
(cont.)

- A specialization may be:
  - **Total** or **Partial**
    - **Total**: each higher-level entity corresponds to at least one of the lower-level entities (e.g., a *Vehicle* is a *Car* or a *Truck*)
    - **Partial**: there may be higher-level entities that do not correspond to any lower-level entity (e.g., a *Person* may not hold a diploma nor a degree)
  - **Disjoint** or **Overlapping**
    - **Disjoint**: a higher-level entity corresponds to at most one of the lower-level entities (e.g., a *Vehicle* can be a *Car* or a *Truck*, but not both)
    - **Overlapping**: a higher-level entity may correspond to more lower-level entities (e.g., a *Person* may hold both a diploma as well as a degree)
- Of course, each lower-level entity corresponds to a higher-level entity
- **Always write** in the diagram the kind of specialization

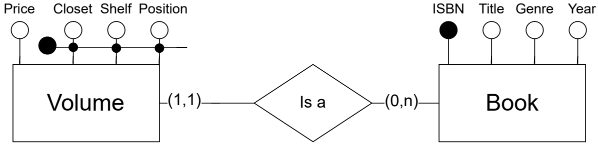
Specialization
(cont.)

A specialization can also have a single child

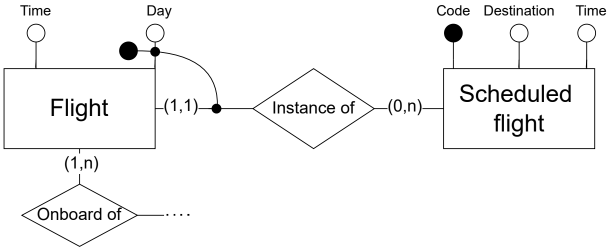


Note how the (natural) domain constraint *"a manager can only manage projects he works on"* is not enforced by the schema

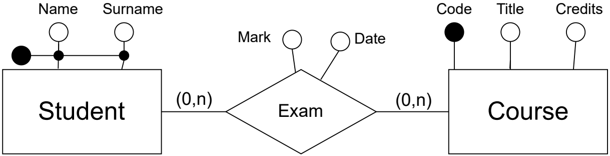
- Consider the following scenario:
  - A library maintains a catalogue of published books. For each book, it records: ISBN, title, genre, and year of publication
  - Not all books are available at the library. Of the available ones, we record the selling price and the location of each copy: closet, shelf, position in the shelf
- We may notice the presence of two different entities:
  - *Book*: a "virtual" book, with its ISBN
  - *Copy*: the actual, "physical" volume, which can be considered as an instance of a book



- An airport wants to keep track of flights
- A flight is uniquely identified by a code, and characterized by a destination and a daily scheduled departure time (e.g., the AZ123 flight of 3:50 PM to Rome)
- For each occurrence of a flight, we also want to keep track of the actual departure time, its crew, [. . .]

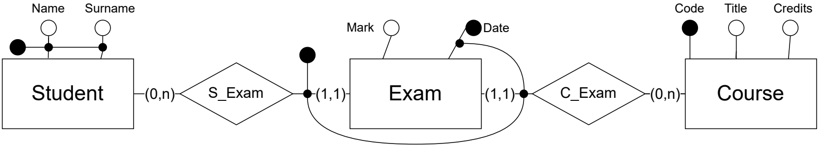


The following solution is valid if and only if a given student can only take the exam for a course *once*



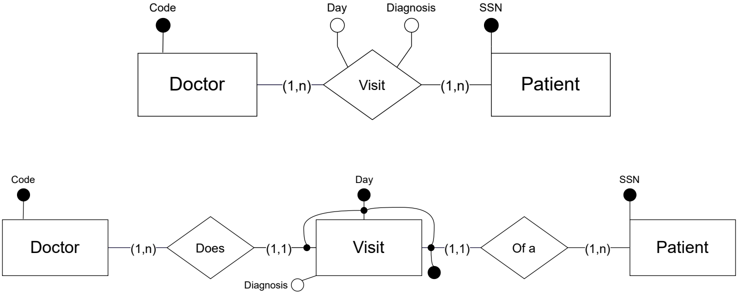
This is because the relationship set *Exam* is composed of unique pairs *Student–Course* (the fact that the relationships has some attributes does not change that)

To allow a student taking more than once the exam for a course, we must employ a reification process

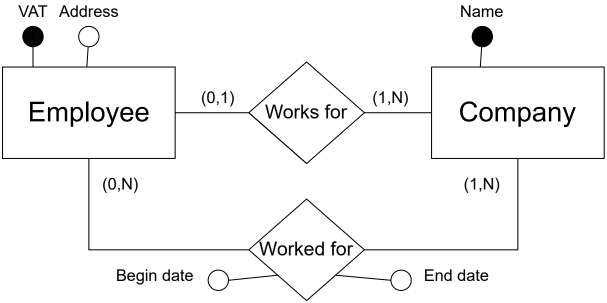


Note that the attribute *Date* belongs now to the key of the entity *Exam*

Assume that we want to model, within a hospital domain, information about doctors, patients, and visits. . . can you spot any differences between the two solutions?



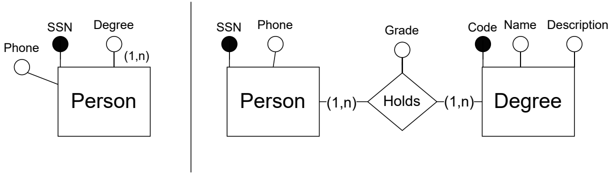
Note: a third solution might be associating a *Code* attribute to *Visit* acting as the key, and relying on a strong entity instead of a weak one.



Domain constraints:

- A given *Employee* must participate to at least one between *Works for* or *Worked for*, possibly both

- When should attributes be used, and when entity and relationship sets instead?
- It really depends on the domain, and on the amount of detail that we want to express



- While being highly expressive, an E-R diagram is seldom enough to encode all the requirements of a given application

- For instance, it does not convey the semantics behind entity set names
  - Does an entity set *Project* refer to something internal or external to the company?

- In addition, some domain constraints cannot simply be expressed using the E-R notation
  - The salary of an employee must be lower than that of his/her supervisor

- Thus, E-R diagrams are typically complemented by natural language notes that further describe their semantics and all the constraints that were not captured in them

Example of requirements in natural language

We wish to create a database for a company that runs training courses. For this, we must store data about the trainees and the instructors. For each course participant (about 5000), identified by a code, we want to store the social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and period employed), the courses attended (there are about 200 courses) and the final assessment of each course. We need also to represent the seminars that each participant is attending at present and, for each day, the places and times the classes are held. Each course has a code and a title and any course can be given any number of times. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we represent the start date, the end date, and the number of participants. If a trainee is a self-employed professional, we need to know his or her area of expertise, and, if appropriate, his or her title. For somebody who works for a company, we store the level and position held. For each instructor (about 300), we will show the surname, age, place of birth, the edition of the course taught, those taught in the past and the courses that the tutor is qualified to teach. All the instructors' telephone numbers are also stored. An instructor can be permanently employed by the training company or can be freelance.

Some rules for requirements analysis

- Choose the appropriate level of abstraction.
- Standardize sentence structure.
- Avoid complex phrases.
- Identify synonyms and homonyms, and standardize terms.
- Make cross-references explicit.
- Construct a glossary of terms.

Complete example

Database Systems
Chapter 6: Conceptual design

An example of a glossary of terms

Term	Description	Synonym	Links
Trainee	Participant in a course. Can be an employee or self-employed.	Participant	Course, Employer
Instructor	Course tutor. Can be freelance.	Tutor	Course
Course	Course offered. Can have various editions.	Seminar	Instructor, Trainee
Employer	Company by which a trainee is employed or has been employed.		Trainee

Rewriting and structuring of requirements (I)

Phrases of a general nature
We wish to create a database for a company that runs training courses. We wish to hold the data for the trainees and the instructors.
Phrases relating to the trainees
For each trainee (about 5000), identified by a code, we will hold the social security number, surname, age, sex, town of birth, current employer, previous employers (along with the start date and the end date of the period employed), the editions of the courses the trainee is attending at present and those he or she has attended in the past, with the final marks out of ten.
Phrases relating to the employers of the trainees
For each employer of a trainee we will hold the name, address and telephone number.

Rewriting and structuring of requirements (II)

Phrases relating to the courses
For each course (about 200), we will hold the name and code. Each time a particular course is given, we will call it an 'edition' of the course. For each edition, we will hold the start date, the end date, and the number of participants. For the editions currently in progress, we will hold the dates, the classrooms and the times in which the classes are held.
Phrases relating to specific types of trainee
For a trainee who is a self-employed professional, we will hold the area of expertise and, if appropriate, the professional title. For a trainee who is an employee, we will hold the level and position held.
Phrases relating to the instructors
For each instructor (about 300), we will hold surname, age, town of birth, all telephone numbers, the edition of courses taught, those taught in the past and the courses the instructor is qualified to teach. The instructors can be permanently employed by the training company or can be freelance.

Database Systems
Chapter 6: Conceptual design

Example of operational requirements

- **operation 1**: insert a new trainee including all his or her data (to be carried out approximately 40 times a day);
- **operation 2**: assign a trainee to an edition of a course (50 times a day);
- **operation 3**: insert a new instructor, including all his or her data and the courses he or she is qualified to teach (twice a day);
- **operation 4**: assign a qualified instructor to an edition of a course (15 times a day);
- **operation 5**: display all the information on the past editions of a course with title, class timetables and number of trainees (10 times a day);
- **operation 6**: display all the courses offered, with information on the instructors who are qualified to teach them (20 times a day);
- **operation 7**: for each instructor, find the trainees all the courses he or she is teaching or has taught (5 times a week);
- **operation 8**: carry out a statistical analysis of all the trainees with all the information about them, about the editions of courses they have attended and the marks obtained (10 times a month).

General criteria for data representation

- If a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it by an entity.
- If a concept has a simple structure, and has no relevant properties associated with it, it is convenient to represent it by an attribute of another concept to which it refers.
- If the requirements contain a concept that provides a logical link between two (or more) entities, it is convenient to represent this concept by a relationship.
- If one or more concepts are particular cases of another concept, it is convenient to represent them by means of a generalization.

Complete example

Design strategies for conceptual design

- The development of a conceptual schema based on its specification must be considered to all intents and purposes an engineering process, and, as such, design strategies used in other disciplines can be applied to it.
  - Top-down
  - Bottom-up
  - Inside-out
  - Mixed

Top-down strategy

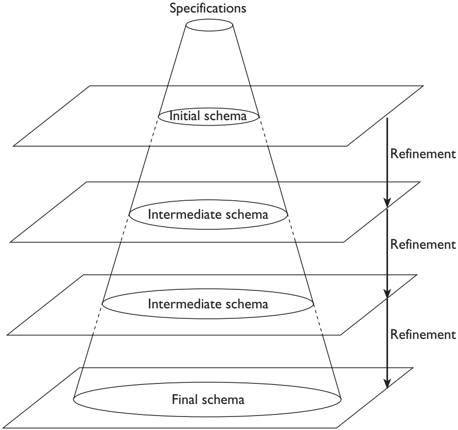
- The conceptual schema is produced by means of a series of successive refinements, starting from an initial schema that describes all the requirements by means of a few highly abstract concepts.

- The schema is then gradually expanded by using appropriate modifications that increase the detail of the various concepts.

- Moving from one level to another, the schema is usually modified using some basic transformations called *top-down transformation primitives*.

Database Systems
Chapter 6: Conceptual design

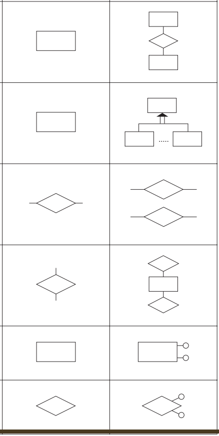
The top-down strategy



Database Systems
Chapter 6: Conceptual design

Top-down transformation primitives

Transformation	Initial concept	Result
T_1 From one entity to two entities and a relationship between them		
T_2 From one entity to a generalization		
T_3 From one relationship to multiple relationships		
T_4 From one relationship to an entity with relationships		
T_5 Adding attributes to an entity		
T_6 Adding attributes to a relationship		

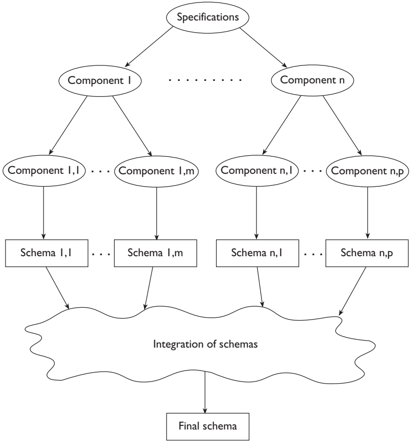


Bottom-up strategy

- The initial specifications are decomposed into smaller and smaller components, until each component describes an elementary fragment of the specifications.
- The various components are then represented by simple conceptual schemas that can also consist of single concepts.
- The various schemas thus obtained are then integrated until a final conceptual schema is reached.
- The final schema is usually obtained by means of some elementary transformations, called *bottom-up transformation primitives*.

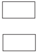
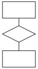
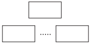
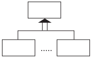
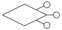
Database Systems
Chapter 6: Conceptual design

The bottom-up strategy



Complete example

Bottom-up transformation primitives

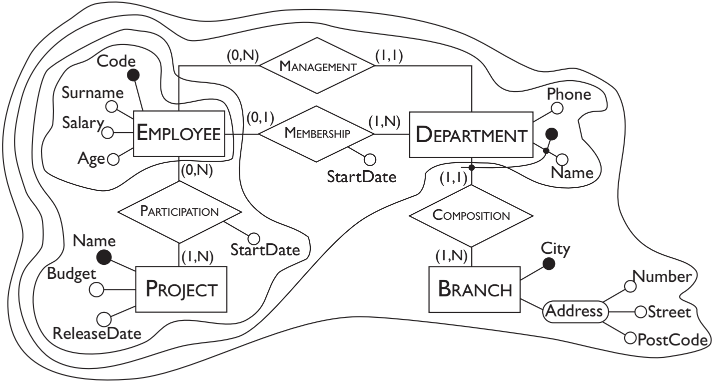
Transformation	Initial concept	Result
T_1 Generation of an entity		
T_2 Generation of a relationship		
T_3 Generation of a generalization		
T_4 Aggregation of attributes on an entity		
T_5 Aggregation of attributes on a relationship		

Inside-out strategy

- This strategy can be regarded as a particular type of bottom-up strategy.
- It begins with the identification of only a few important concepts and, based on these, the design proceeds, spreading outward ‘radially’.
- First the concepts nearest to the initial concepts are represented, and we then move towards those further away by means of ‘navigation’ through the specification.

Database Systems
Chapter 6: Conceptual design

An example of the inside-out strategy



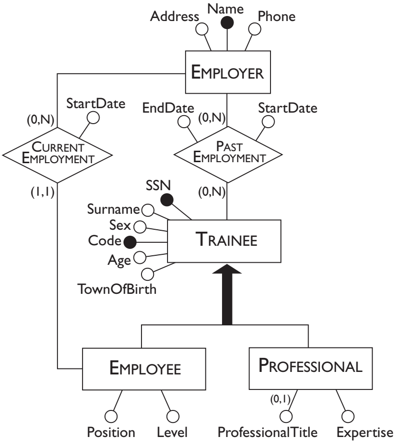
Mixed strategy

- In a mixed strategy the designer decomposes the requirements into a number of components, as in the bottom-up strategy, but not to the extent where all the concepts are separated.
- At the same time he or she defines a *skeleton schema* containing the main concepts of the application. This skeleton schema gives a unified view of the whole design and helps the integration of schemas developed separately.
- Then the designer examines separately these main concepts and can proceed with gradual refinements (following the top-down strategy) or extending a portion with concepts that are not yet represented (following the bottom-up strategy).

Database Systems
Chapter 6: Conceptual design

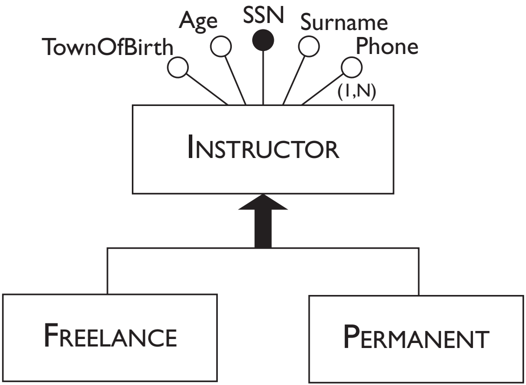
Skeleton schema for a training company

The refinement of a portion of the skeleton schema



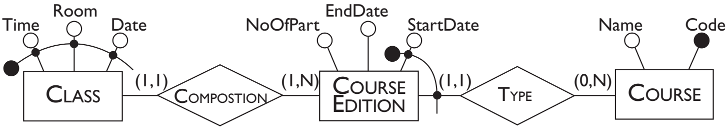
Database Systems
Chapter 6: Conceptual design

The refinement of another portion of the skeleton schema



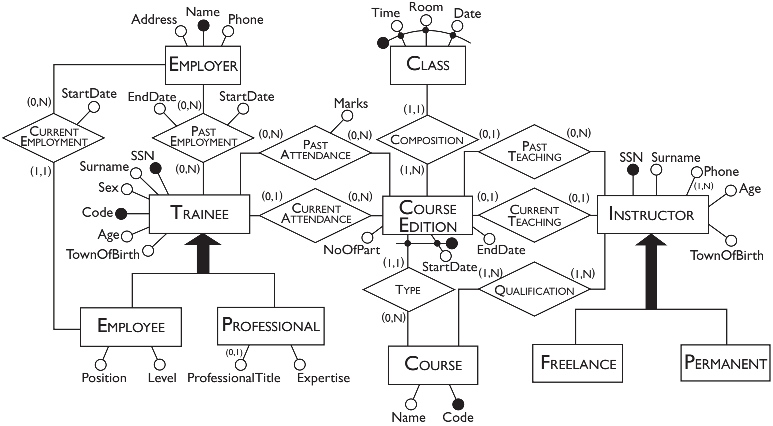
Database Systems
Chapter 6: Conceptual design

The refinement of another portion of the skeleton schema



Database Systems
Chapter 6: Conceptual design

The final E-R schema for the training company



Complete example

Quality of a conceptual schema

- Correctness
- Completeness
- Readability
- Minimality

Database Systems
Chapter 6: Conceptual design

A comprehensive method for conceptual design (I)

1. Analysis of requirements
   (a) Construct a glossary of terms.
   (b) Analyze the requirements and eliminate any ambiguities.
   (c) Arrange the requirements in groups.
2. Basic step
   (a) Identify the most relevant concepts and represent them in a skeleton schema.
3. Decomposition step (to be used if appropriate or necessary).
   (a) Decompose the requirements with reference to the concepts present in the skeleton schema.

A comprehensive method for conceptual design (II)

4. Iterative step (to be repeated for all the schemas until every specification is represented)
   - (a) Refine the concepts in the schema, based on the requirements.
   - (b) Add new concepts to the schema to describe any parts of the requirements not yet represented.
5. Integration step (to be carried out if step 3 has been used)
   - (a) Integrate the various subschemas into a general schema with reference to the skeleton schema.
6. Quality analysis
   - (a) Verify the correctness of the schema.
   - (b) Verify the completeness of the schema.
   - (c) Verify the minimality of the schema.
   - (d) Verify the readability of the schema.

Database Systems
Chapter 6: Conceptual design

CASE tools for database design

- There are CASE tools expressly designed for the creation and development of databases.
- These systems provide support for the main phases of the development of a database (conceptual, logical and physical design).
- They also support specific design tasks like the automatic layout of diagrams, correctness and completeness tests, quality analysis, automatic production of DDL code for the creation of a database.

Exercise 1, TV series

Let us synthesize the ER conceptual schema for a database of tv series data recording information about episodes, actors and directors, based on the following requirements:

- Let us assume that both actors and directors are uniquely identified by their name and surname, and are characterized by their birth date and their nationality. The same person may be both an actor as well as a director
- Each tv series is characterized by its title, genre, release year, and director. We assume that there is only one director per series

- It may be the case that distinct tv series have the same title (this is the case, for instance, with remakes). Nevertheless, given a year, there cannot be two series with the same title
- A tv series is composed by a number of episodes, each with an incremental number that uniquely identifies it within the series, and a title
- Different episodes may involve different actors, each with a specific role. For the sake of simplicity, let us assume that, in a given episode, each actor may play one role only

Let us synthesize the ER conceptual schema of a database for the management of a university on the basis of the following set of requirements.

- Each course offered by the university is uniquely identified by its numerical code and characterized by a name, a set of prerequisites (a set of other courses that must be already passed before attending the course), a number of hours, and the professors that are allowed to teach it
- Professors are partitioned into full and associate professors. Each professor is uniquely identified by his/her fiscal code and characterized by a first and a last name, one or more mobile numbers, possibly an email address, and the department to which he/she belongs. For full professors, we store the day they were promoted

Exercise 2, University DB (cont.)

- Each university's department is uniquely identified by its name and characterized by a code and a budget. Every department has a manager, selected among the full professors. In addition, we want to keep track of the number of professors belonging to each department
- A given course can be offered on different years. We call each course offering an "edition" of a course (e.g., the *databases* course of year 2022). Each course edition is taught by one or more professors, among those allowed to teach it
- For teaching purposes, a given professor can use one or more books for each course edition in which he/she is involved. A book is identified by its ISBN, and has a title, editor, edition, and one or more authors

Exercise 3

Design an ER schema for a database storing information regarding people that are resident or buried in a given city:

- For each person we are interested in storing the name, surname, tax code (uniquely identifying the person), gender, and date of birth. A person can be resident (living) or buried (deceased) in the city. For each living person we want to store the address (street, house number), one or more telephone numbers, and marital status
- Let us assume that the city has several cemeteries, each identified by an alphanumeric code and characterized by a name and an address. Within each cemetery, each burial site is identified by a numeric code (thus, burial sites located in different cemeteries may have the same number)

Exercise 3

- For each deceased person buried in a cemetery of the city, we store the burial site (note that some burial sites are collective, e.g., family sites), date of death, age at the time of death, and death cause

- Build an ER schema that describes the above mentioned requirements, clearly explaining any assumptions you made. In particular, for each entity, identify its candidate keys, and explain how possible derived attributes should be calculated. Finally, carefully specify the constraints associated with each relation and the derivation rules for any derived attributes