Database Design

- Database design is just one of the many activities in the development of an information system within an organization;

- it should therefore be presented within the wider context of the information system life cycle.

Life cycle of an information system

$$\begin{tikzcd}[row sep=large]
\boxed{\text{Feasibility study}} \arrow[d] \\
\boxed{\text{Collection \& analysis of requirements}} \arrow[d] \\
\boxed{\text{Design}} \arrow[d] \\
\boxed{\text{Implementation}} \arrow[d] \\
\boxed{\text{Validation and testing}} \arrow[d] \\
\boxed{\text{Operation}}
\end{tikzcd}$$

Phases of information system life cycle

- **Feasibility study**. It serves to define the costs of the various possible solutions and to establish the priorities for the creation of the various components of the system.
- **Collection and analysis of requirements**. It consists of the definition and study of the properties and functionality of the information system.
- **Design**. It is generally divided into two tasks: *database design* and *operational design*.
- **Implementation**. It consists of the creation of the information system according to the characteristics defined in the design.
- **Validation and testing**. This is to check the correct functioning and quality of the information system.
- **Operation**. The information system becomes live and performs the tasks for which it was originally designed.

Information systems and databases

- The database constitutes only one of the components of an information system, which also includes application programs, user interfaces and other service programs.

- However, the central role that the data itself plays in an information system more than justifies an independent study of database design.

- For this reason, we deal with only those aspects of information system development that are closely related to databases, focusing on data design and on the related activities of collection and analysis of the requirements.

Methodologies for database design

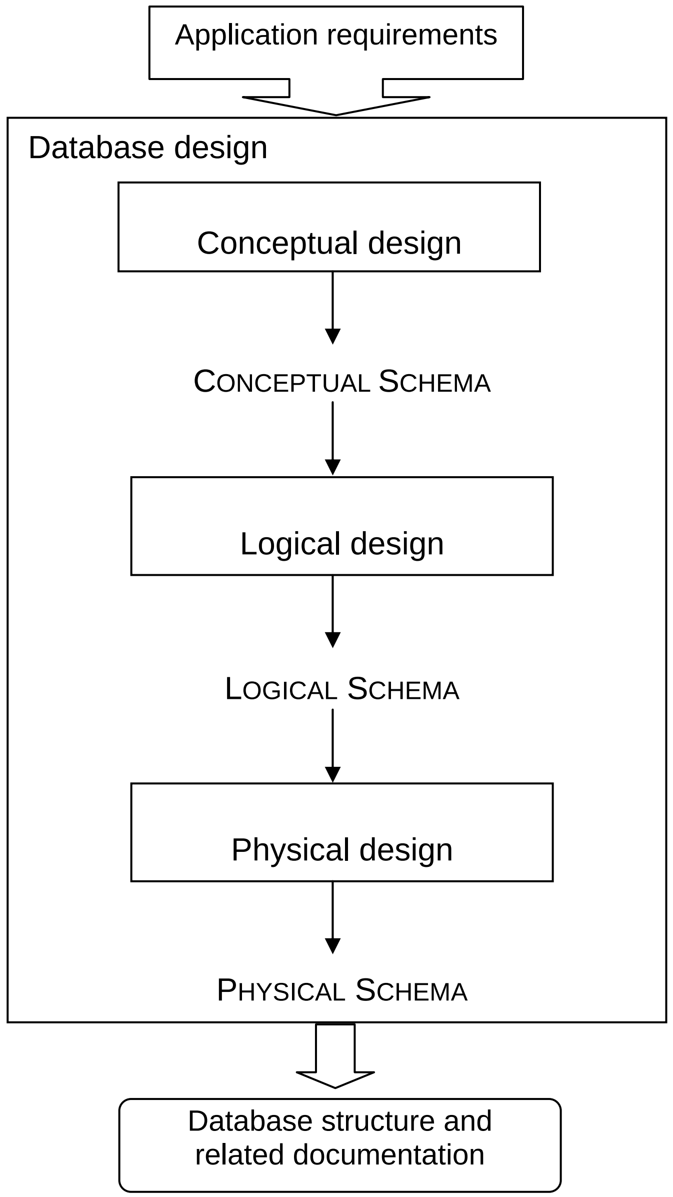
- We follow a structured approach to database design that can be regarded as a ‘design methodology’.

- As such, it is presented by means of:
  - a *decomposition* of the entire design activity in successive steps, independent one from the other;
  - a series of *strategies* to be followed in the various steps and some *criteria* from which to choose in the case of there being options;
  - some *reference models* to describe the inputs and outputs of the various phases.

A database design methodology

- Within the field of databases, a design methodology has been consolidated over the years.
- It is based on a simple but highly efficient engineering principle: separate the decisions relating to ‘what’ to represent in the database, from those relating to ‘how’ to do it.
- This methodology is divided into three phases to be followed consecutively.

The phases of database design



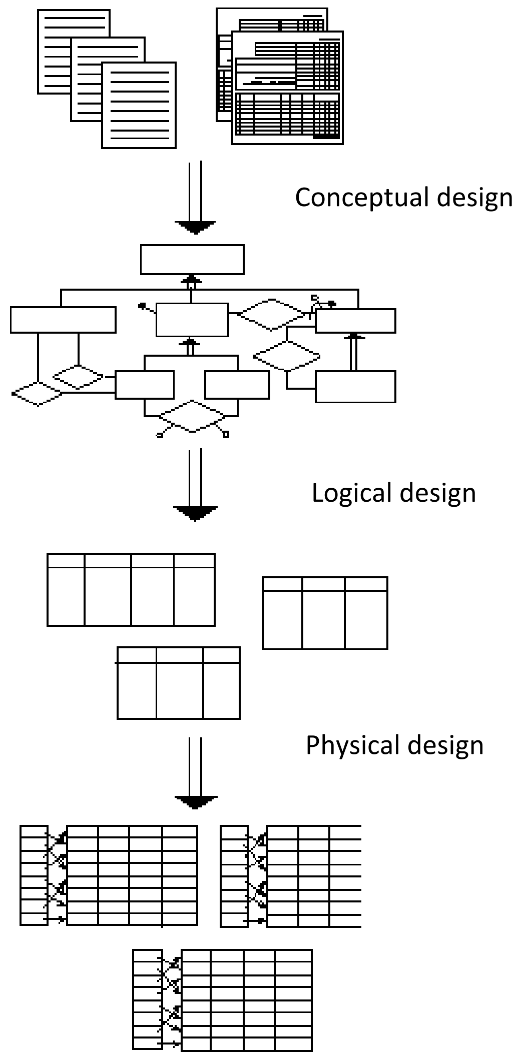
Phases of database design

- **Conceptual design**. The purpose of this is to represent the informal requirements of an application in terms of a *conceptual schema* that refers to a *conceptual data model*.

- **Logical design**. This consists of the translation of the conceptual schema defined in the preceding phase, into the *logical schema* of the database that refers to a *logical data model*.

- **Physical design**. In this phase, the logical schema is completed with the details of the physical implementation (file organization and indexes) on a given DBMS. The product is called the *physical schema* and refers to a *physical data model*.

The products of the various phases of a relational database with the E-R model

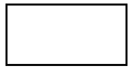
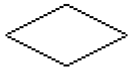
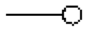
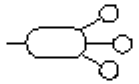
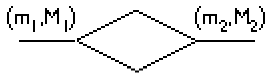
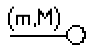
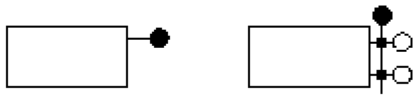
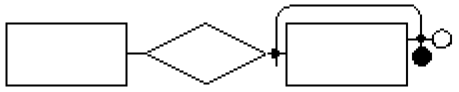
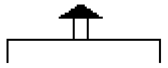
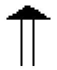


The Entity Relationship model

- The **Entity-Relationship** (E-R) model is a *conceptual* data model, and as such provides a series of *constructs* capable of describing the data requirements of an application in a way that is easy to understand and is independent of the criteria for the management and organization of data on a database system.

- For every construct, there is a corresponding graphical representation. This representation allows us to define an E-R schema diagrammatically.

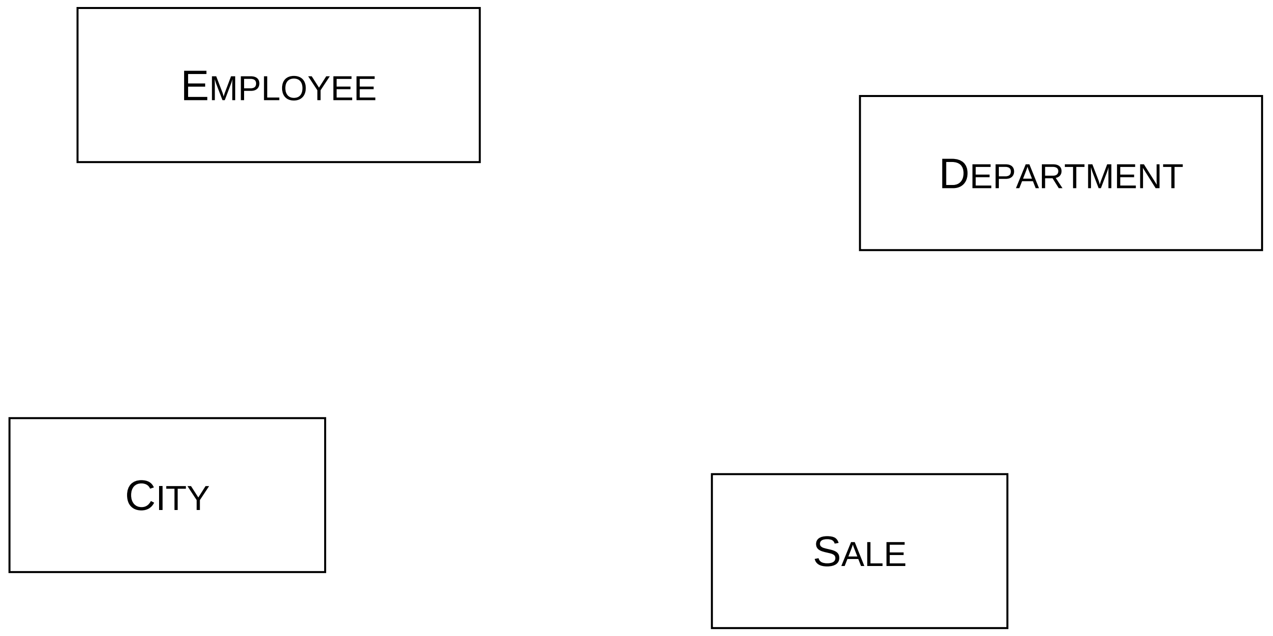
The constructs of the E-R model and their graphical representation

Construct	Graphical representation
Entity	
Relationship	
Simple attribute	
Composite attribute	
Cardinality of a	
Cardinality of an attribute	
Internal identifier	
External identifier	
Generalization	
Subset	

Entities

- These represent classes of objects (facts, things, people, for example) that have properties in common and an autonomous existence.
- CITY, DEPARTMENT, EMPLOYEE, PURCHASE and SALE are examples of entities in an application for a commercial organization.
- An occurrence of an entity is an object of the class that the entity represents.

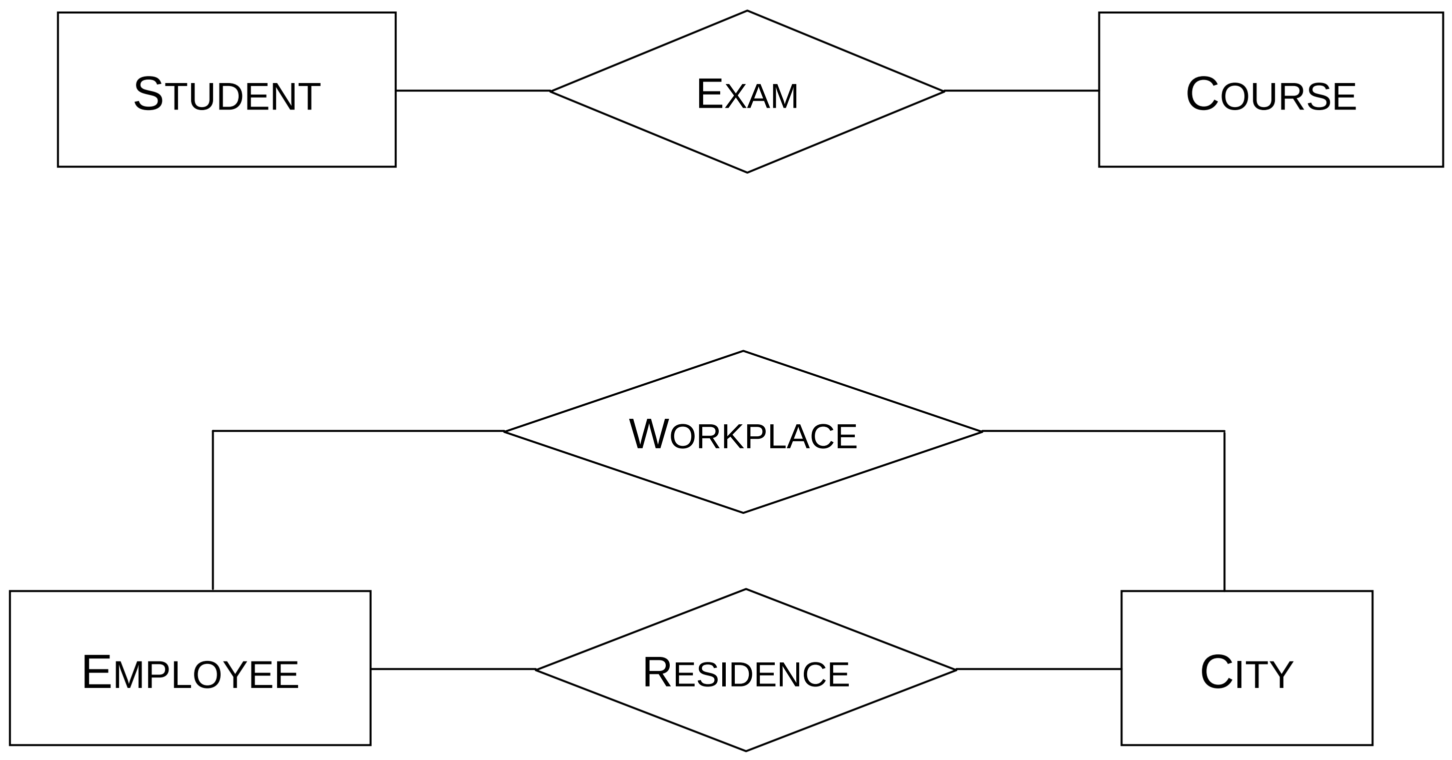
Examples of entities in the E-R model



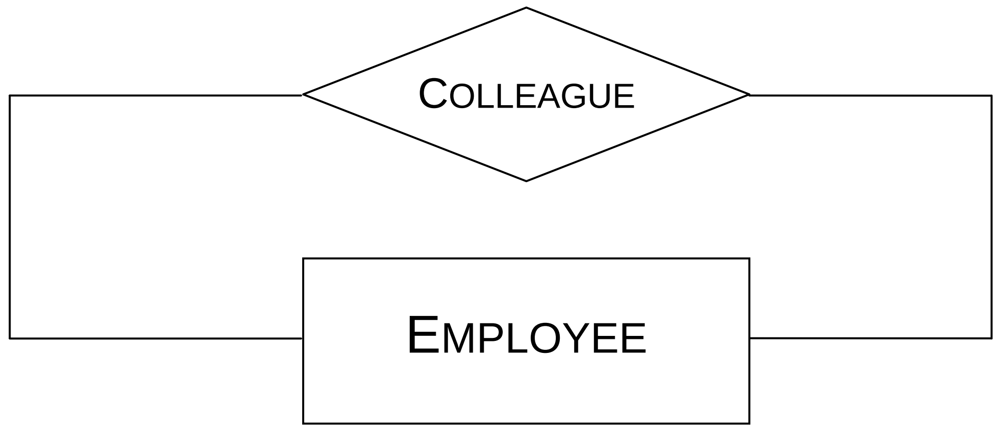
Relationships

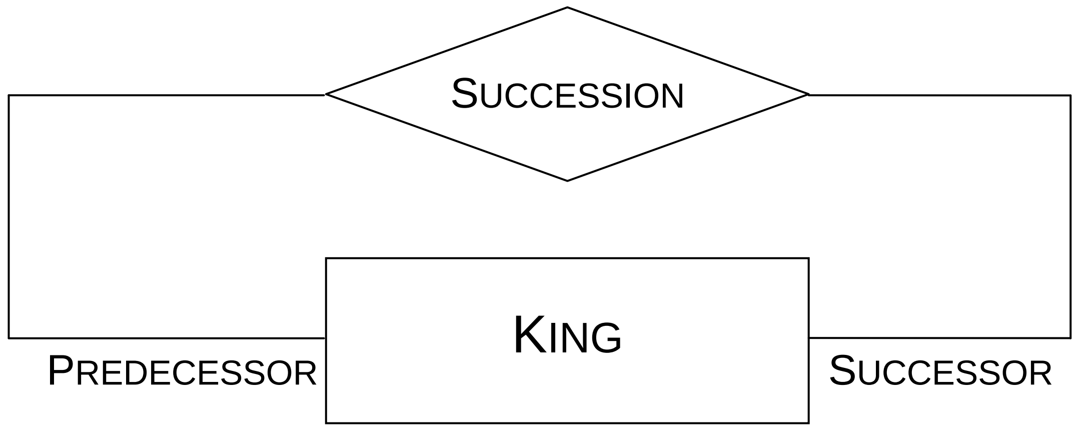
- They represent logical links between two or more entities.
- RESIDENCE is an example of a relationship that can exist between the entities CITY and EMPLOYEE; EXAM is an example of a relationship that can exist between the entities STUDENT and COURSE.
- An occurrence of a relationship is an n-tuple made up of occurrences of entities, one for each of the entities involved.

Examples of relationships in the E-R model

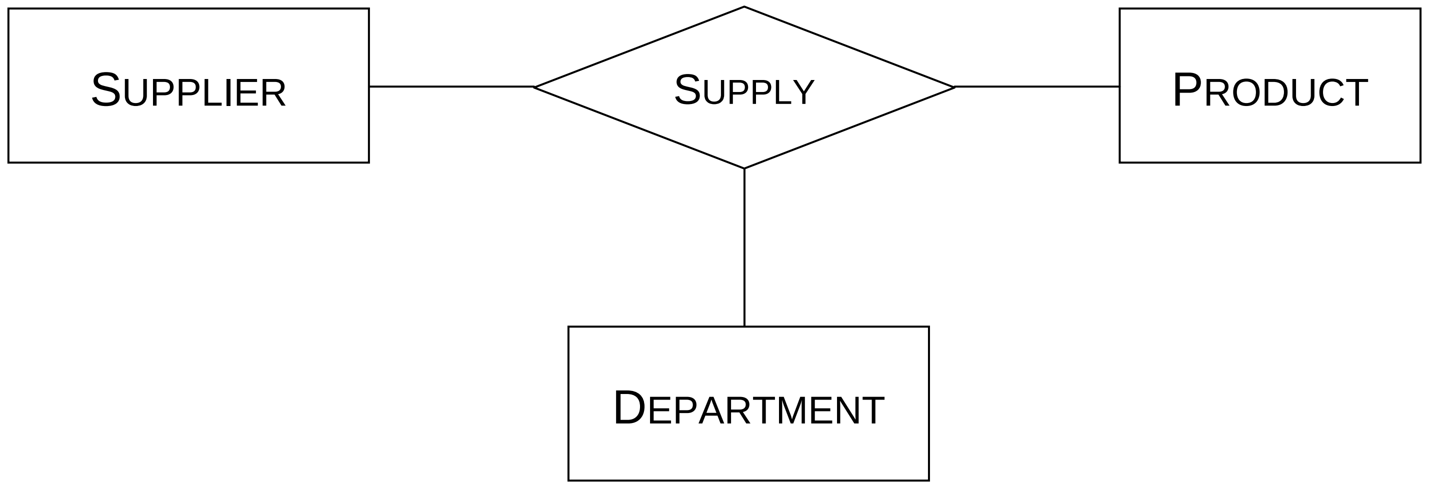


Examples of recursive relationships in the E-R model





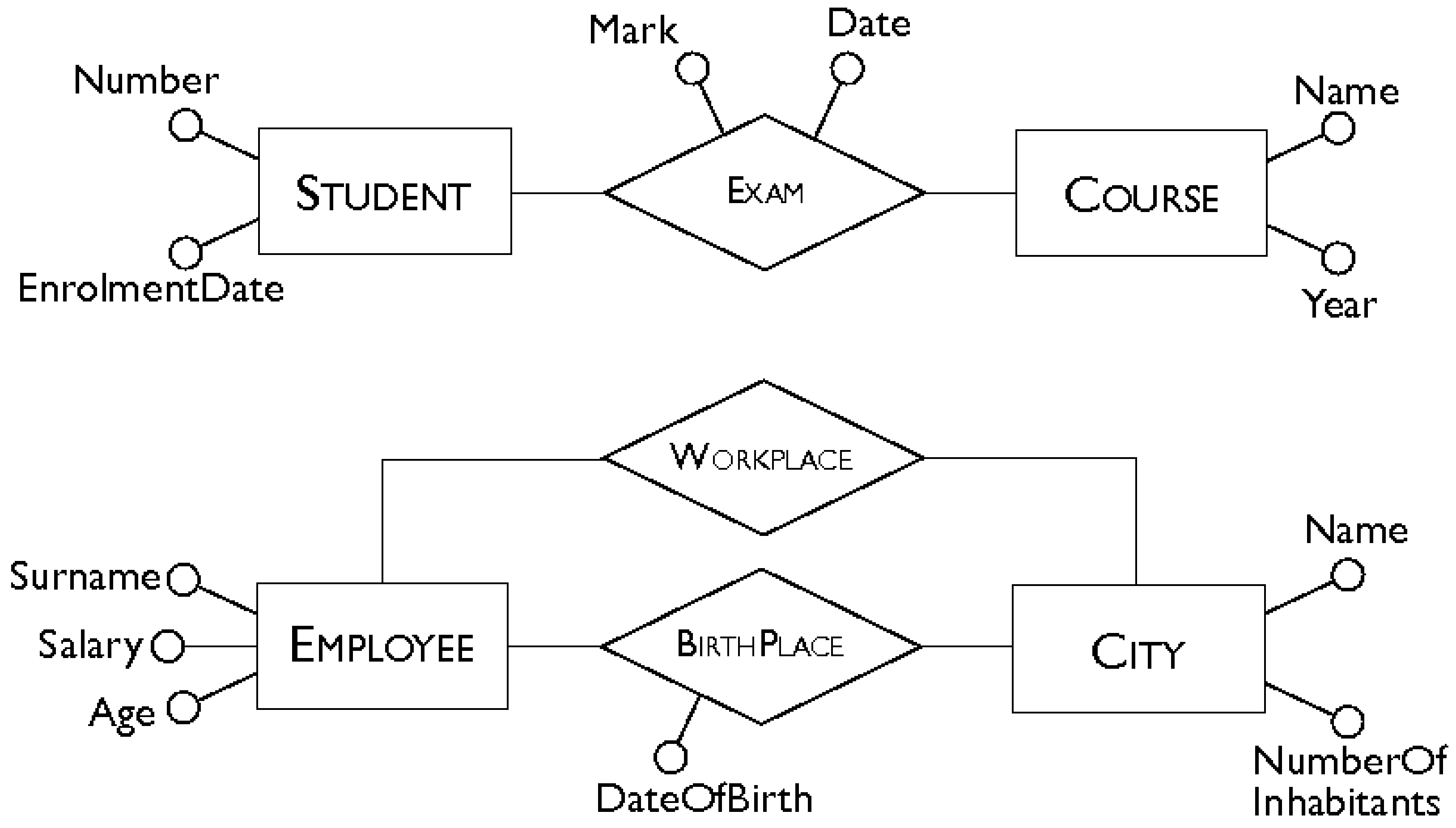
Example of a ternary relationship in the E-R model



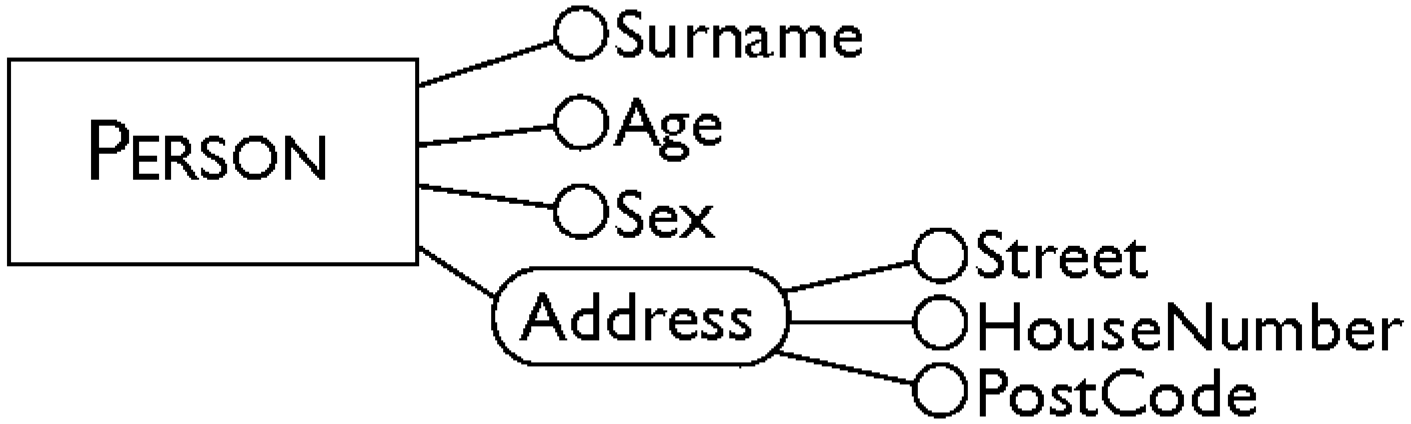
Attributes

- These describe the elementary properties of entities or relationships.
- Surname, Salary and Age are possible attributes of the EMPLOYEE entity, while Date and Mark are possible attributes for the relationship EXAM between STUDENT and COURSE.
- An attribute associates with each occurrence of an entity (or relationship) a value belonging to a set known as the domain of the attribute.
- The domain contains the admissible values for the attribute.

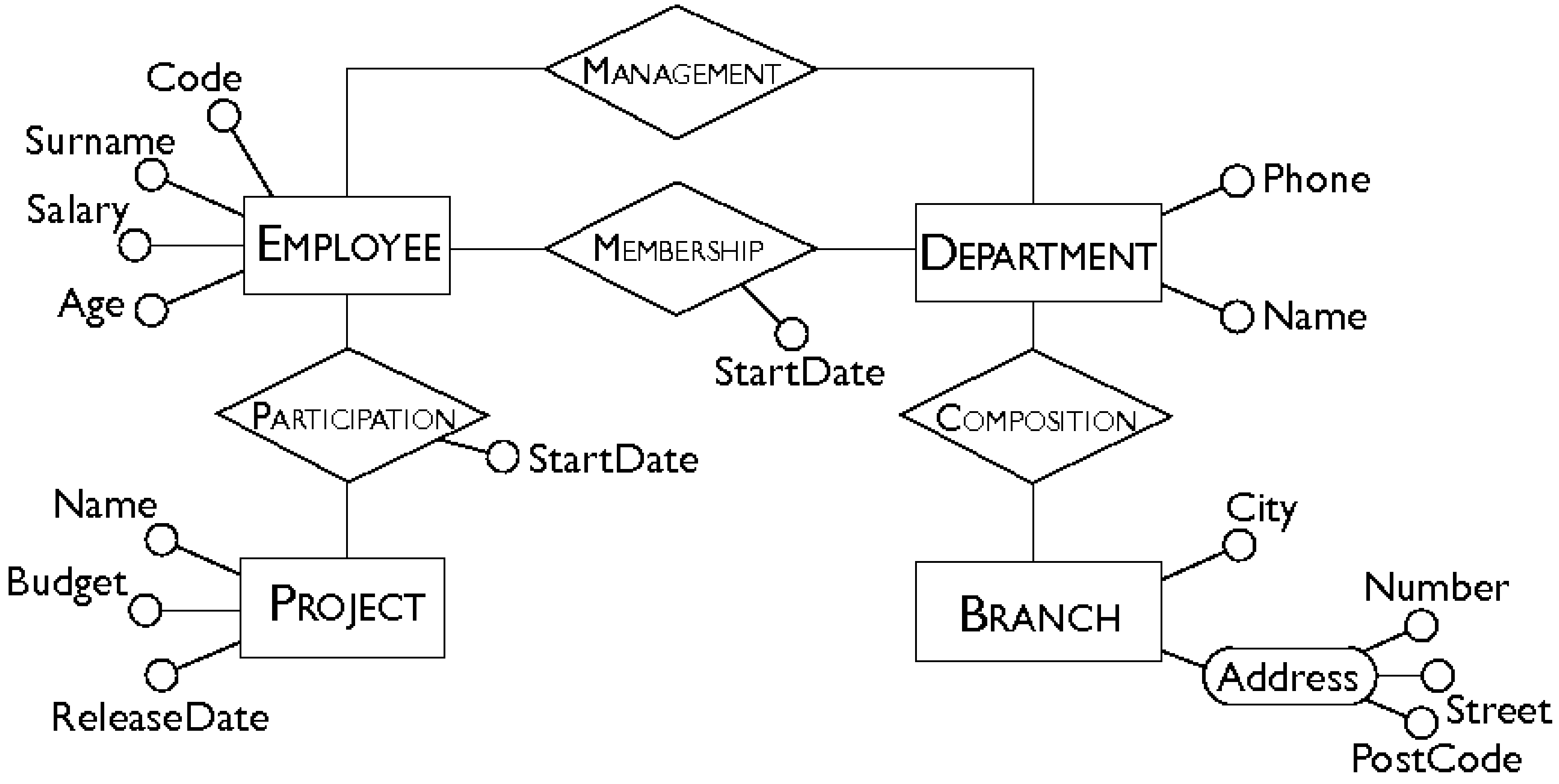
E-R schemas with relationships, entities and attributes



An example of an entity with a composite attribute



An E-R schema

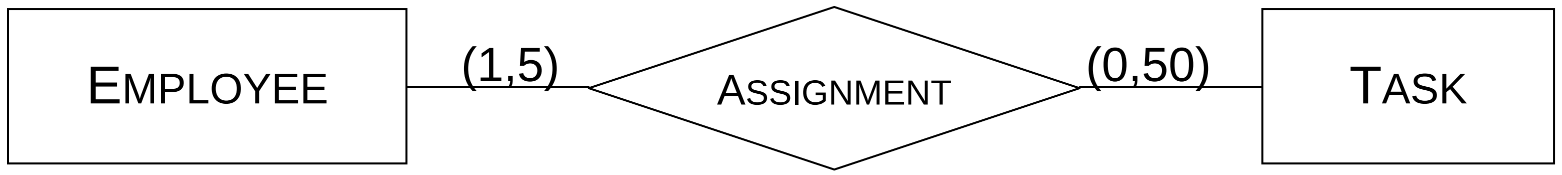

MANAGEMENT
Code
Surname
Salary
Age
EMPLOYEE
MEMBERSHIP
StartDate
DEPARTMENT
Phone
Name
PARTICIPATION
StartDate
COMPOSITION
Name
Budget
ReleaseDate
PROJECT
City
BRANCH
Number
Address
Street
PostCode

Cardinalities

- They are specified for each entity participating in a relationship and describe the **minimum** and **maximum** number of relationship occurrences in which an entity occurrence can participate.

- They state therefore how many times in a relationship between entities an occurrence of one of these entities can be linked to occurrences of the other entities involved.

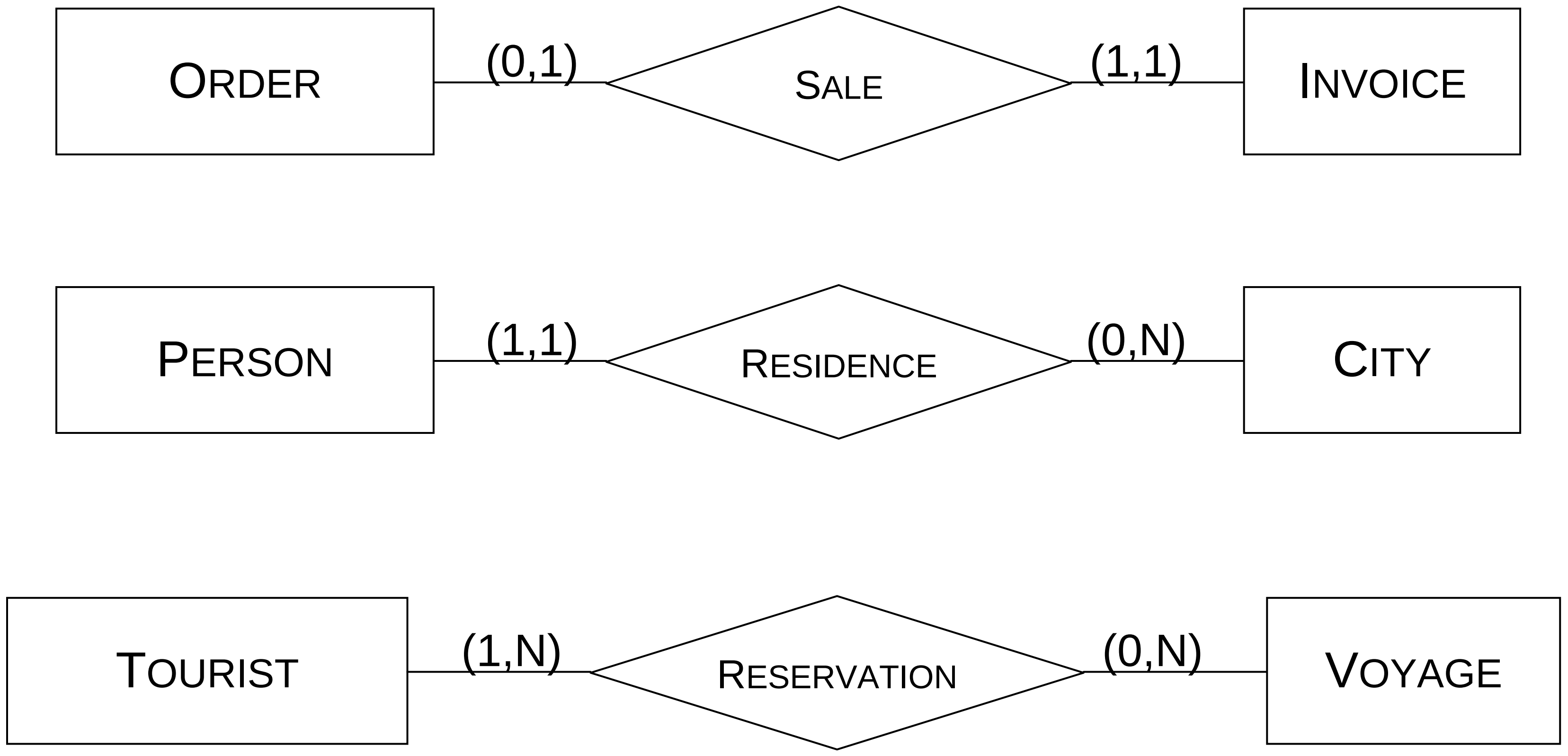
Cardinality of a relationship in the E-R model



Values for cardinalities

- In most cases, it is sufficient to use only three values for cardinalities: zero, one and the symbol N:
  - for the minimum cardinality, zero or one; in the first case (zero) we say that the participation in the relationship is *optional*, in the second (one) we say that the participation is *mandatory*;
  - for the maximum cardinality, one or many (N); in the first case (one) each occurrence of the entity is associated at most with a single occurrence of the relationship, while in the second case (N) each occurrence of the entity is associated with an arbitrary number of occurrences of the relationship.

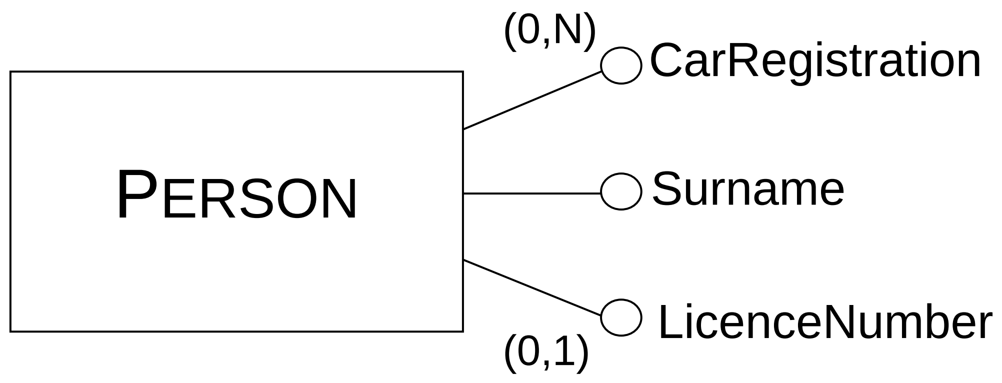
Examples of cardinality of relationships



Cardinalities of attributes

- They can be specified for the attributes of entities (or relationships) and describe the minimum and maximum number of values of the attribute associated with each occurrence of an entity or a relationship.
- In most cases, the cardinality of an attribute is equal to (1,1) and is omitted.
- The value of a certain attribute can be null or there can exist various values of a certain attribute associated with an entity occurrence.

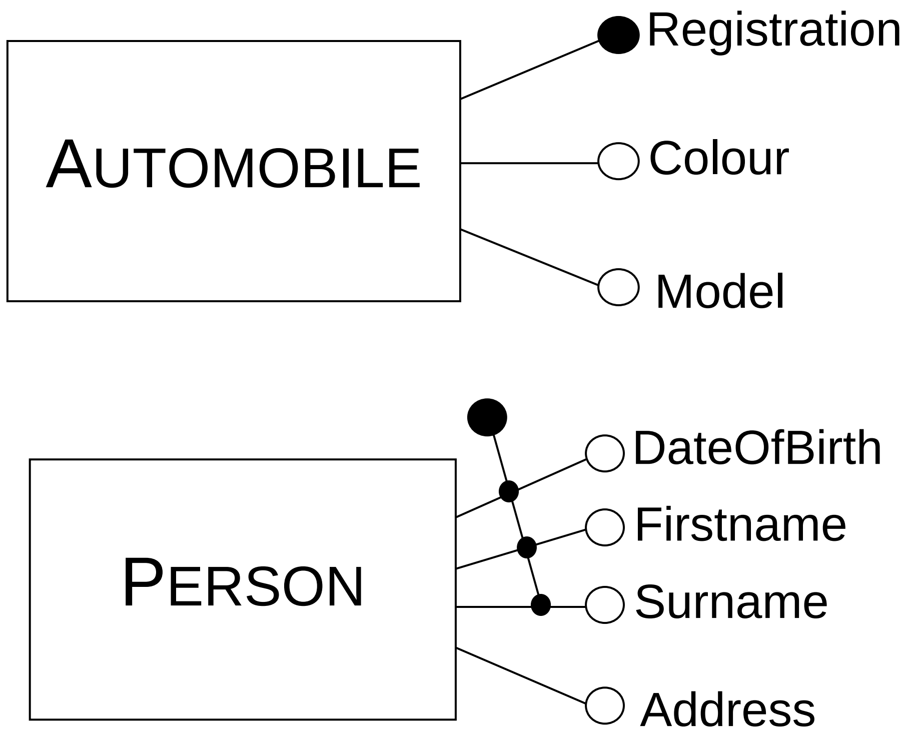
Example of attributes with cardinality



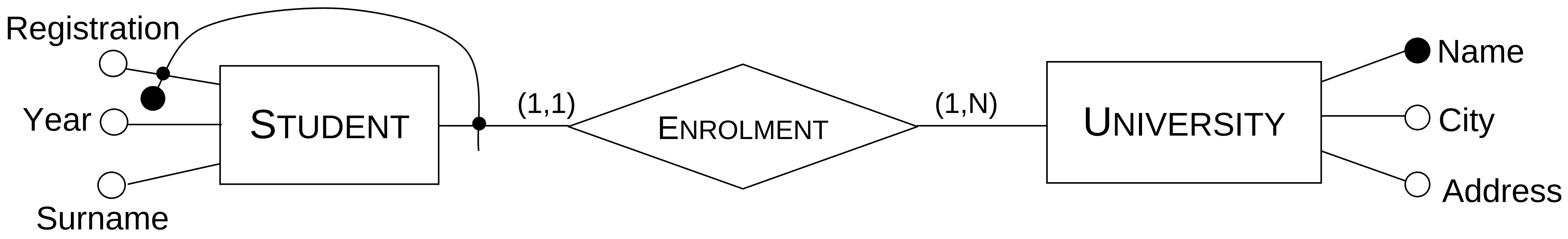
Identifiers

- They are specified for each entity of a schema and describe the concepts (attributes and/or entities) of the schema that allow the unambiguous identification of the entity occurrences.
- In many cases, an identifier is formed by one or more attributes of the entity itself: in this case we talk about an *internal* identifier (also known as a *key*).
- Sometimes, however, the attributes of an entity are not sufficient to identify its occurrences unambiguously and other entities need to be involved in the identification. This is called an *external* identifier.

Examples of internal identifiers



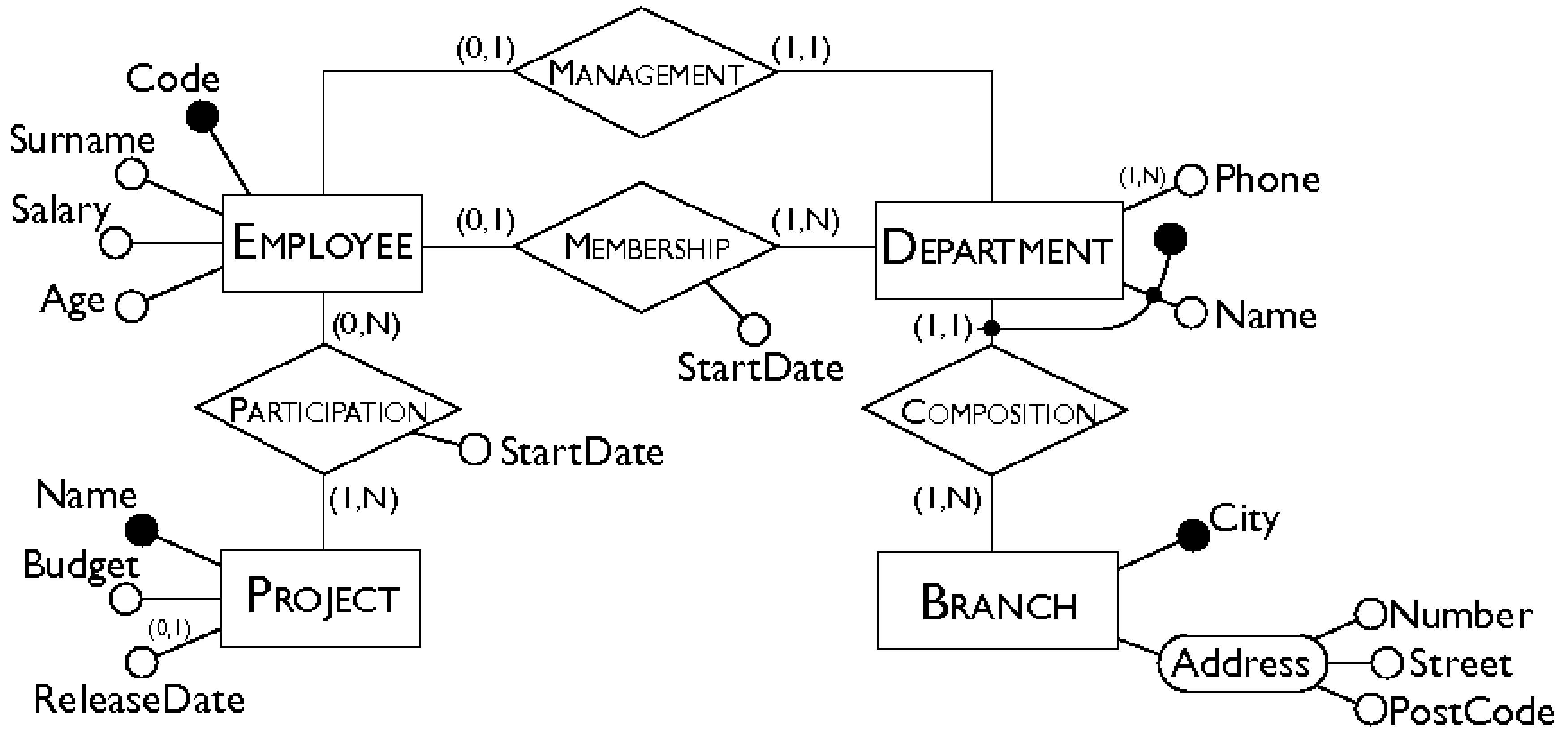
Example of an external entity identifier



General observations on identifiers

- An identifier can involve one or more attributes, provided that each of them has **(1,1) cardinality**;
- an external identifier can involve one or more entities, provided that each of them is member of a relationship to which the entity to identify participates with cardinality equal to (1,1);
- an external identifier can involve an entity that is in its turn identified externally, as long as cycles are not generated;
- each entity must have one (internal or external) identifier, but can have more than one. Actually, if there is more than one identifier, then the attributes and entities involved in an identification can be optional (minimum cardinality equal to 0).

A schema completed by identifiers and cardinalities

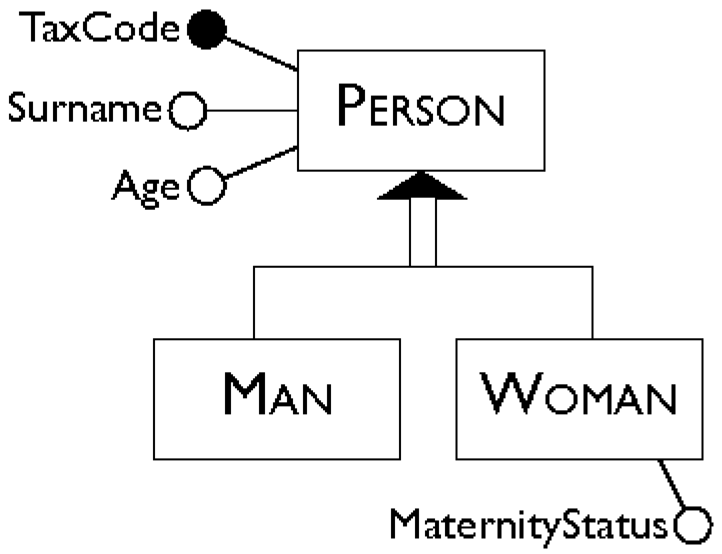


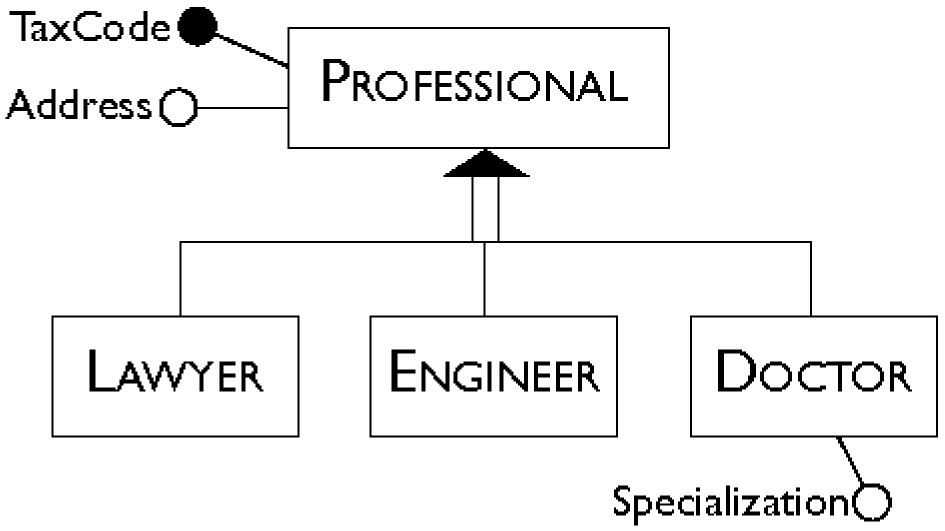
Generalizations

- These represent logical links between an entity E, known as *parent* entity, and one or more entities $E_1,...,E_n$ called *child* entities, of which E is more general, in the sense that it comprises them as a particular case.

- In this situation we say that E is a *generalization* of $E_1,...,E_n$ and that the entities $E_1,...,E_n$ are *specializations* of the E entity.

Examples of generalizations among entities





Properties of generalizations

- Every occurrence of a child entity is also an occurrence of the parent entity.

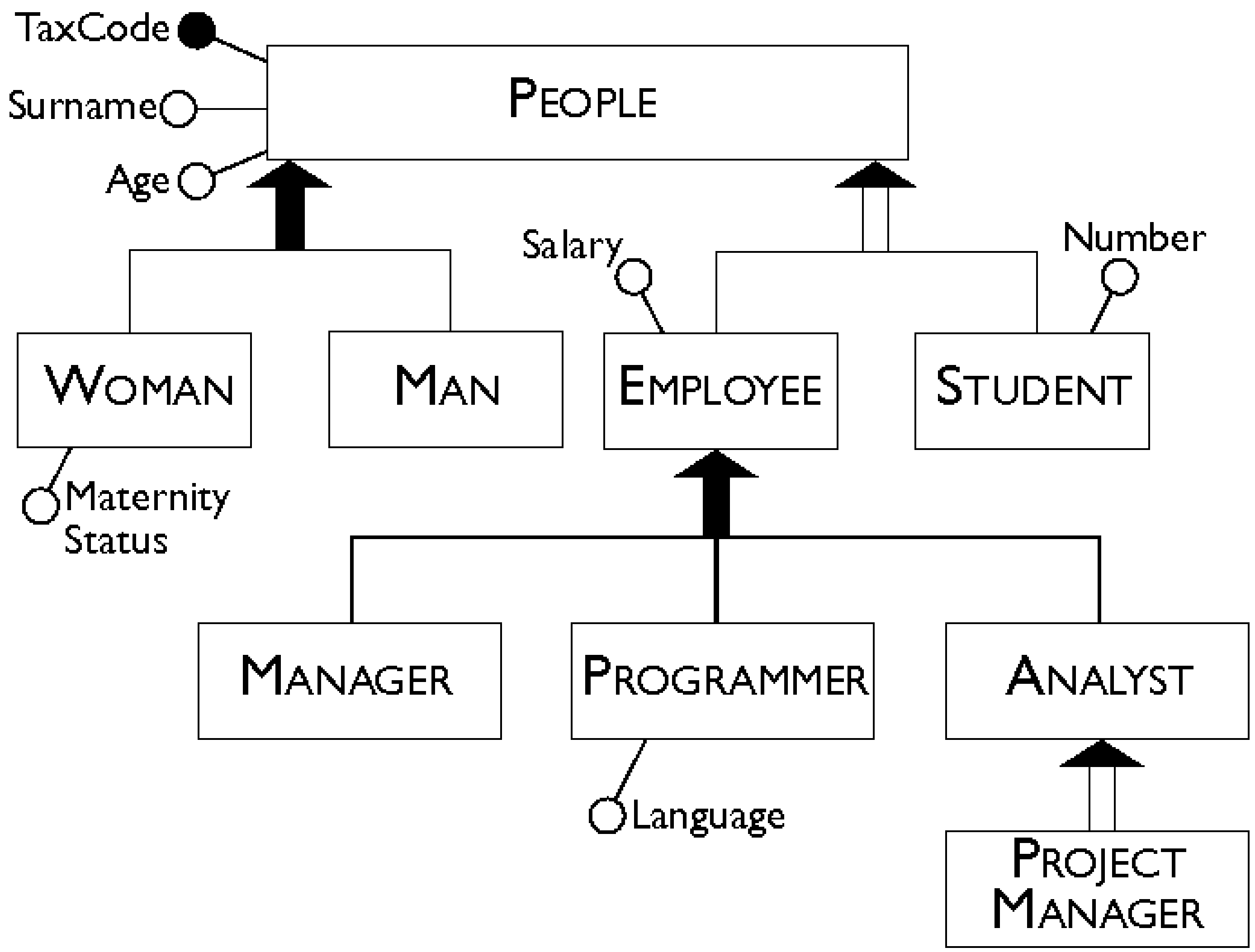
- Every property of the parent entity (attributes, identifiers, relationships and other generalizations) is also a property of a child entity. This property of generalizations is known as *inheritance*.

Classification of generalizations

- A generalization is *total* if every occurrence of the parent entity is also an occurrence of one of the child entities, otherwise it is *partial*;

- A generalization is *exclusive* if every occurrence of the parent entity is at most an occurrence of one of the child entities, otherwise it is *overlapping*.

Hierarchy of generalizations between entities



Documentation of E-R schemas

- An Entity-Relationship schema is rarely sufficient by itself to represent all the aspects of an application in detail; (eg. specific constraints, etc.)
- it is therefore indispensable to provide every E-R schema with support documentation, which can facilitate the interpretation of the schema itself and describe properties of the data that cannot be expressed directly by the constructs of the model

Conceptual design

- The conceptual design of a database consists of the construction of an Entity-Relationship schema, providing an optimal description of the user requirements;

- before we begin to discuss strategies to build this schema, we will discuss the activity that precedes the design process itself: the collection and analysis of the requirements.

Requirements collection and analysis

- The *requirements collection* consists of the complete identification of the problems that the application must solve, and the features that should characterize such an application.
- The *requirements analysis* consists of the clarification and organization of the requirements specification.
- These activities are closely related to one another.

Main sources of requirements

The requirements generally come from different sources, as follows.

- The users of the application.
- All the existing documentation that has some connection with the problem: forms, internal rules, business procedures, laws and regulations, etc.
- Possible earlier applications that are to be replaced or that must interact in some way with the new application.

Natural language specifications

- The requirements specifications are often written in natural language, at least in the first draft.
- Natural language is, by nature subject to ambiguity and misinterpretation.
- We need to carry out an in-depth analysis of the specification document in order to remove any inaccuracies and ambiguous terms.

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.

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.