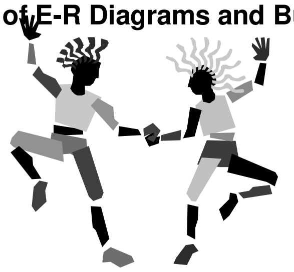
VI. Database Design

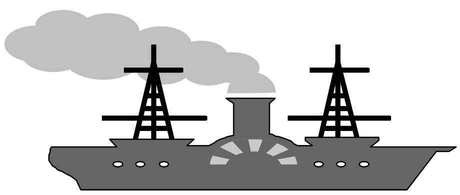
Information System Lifecycle
Database Design
Conceptual, Logical and Physical Design
The Entity-Relationship Model
Entities, Relationships and Attributes
Cardinalities, Identifiers and Generalization
Documentation of E-R Diagrams and Business Rules



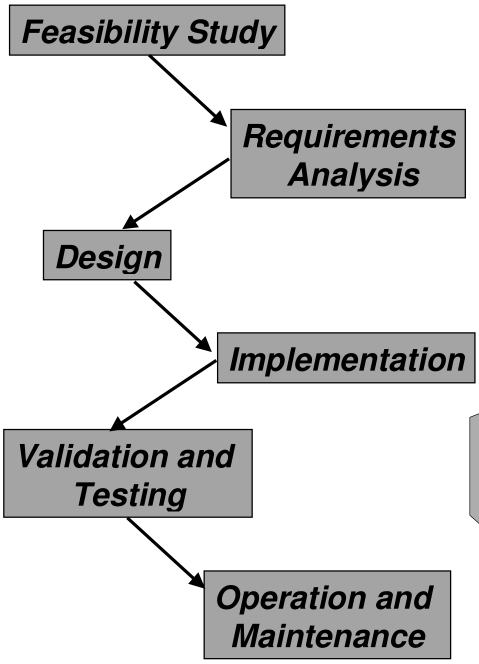
Database Design -- 1

Database Design

- ***Database design*** is just one of several activities in the development of an information system within an organization.
- it should therefore be presented within the wider context of the ***information system lifecycle***.



Database Design -- 2



Lifecycle of an Information System

The Lifecycle of an Information System

- ***Feasibility study*** -- defines the costs of possible solutions, and establishes criteria for selecting among them.
- ***Requirements analysis*** -- defines and studies the desired properties and functionality of the new system.
- ***Design*** -- covers ***database design*** and ***application design***.
- ***Implementation*** -- the system is created according to the specification defined in the design.
- ***Validation and testing*** -- checks the functioning and quality of the information system, to make sure it is consistent with the design.
- ***Operation and maintenance***. The system becomes operational and performs the tasks for which it was originally designed.

The Lifecycle of an Information System (cont'd)

- The process of developing an information system is rarely strictly sequential, given that during any one of the phases it is often necessary to reconsider decisions made earlier.
- Sometimes it is also useful to rapidly create a simplified version of the information system, which is used to test its functionality. This is known as ***prototyping***.
- The prototype is shown to users in order to verify that the high-level requirements of the information system were correctly collected and modelled.

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 database design and related activities.

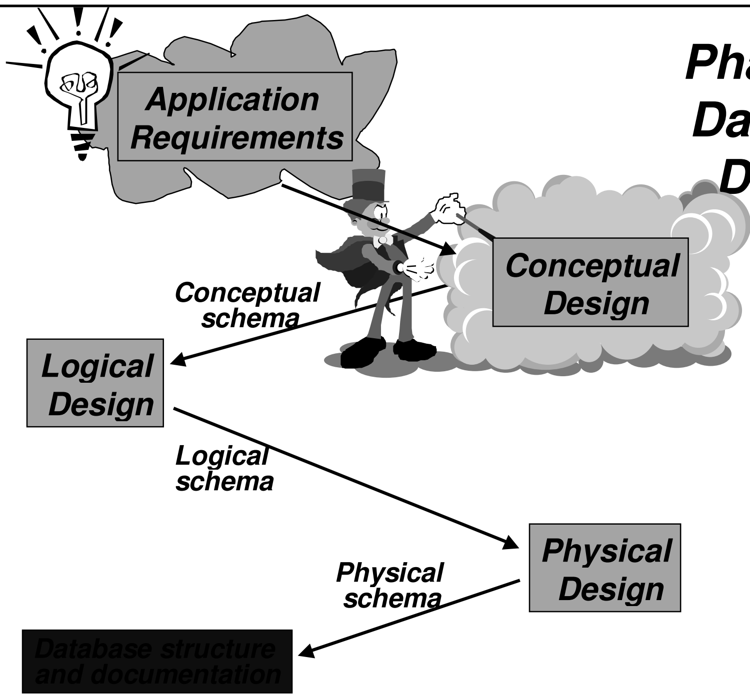
Methodologies for Database Design

- We follow a structured approach to database design that can be regarded as a 'design methodology'.
- As such, it is presented in terms of:
  - ✓ a ***decomposition*** of the entire design activity in successive, independent steps;
  - ✓ a series of ***strategies*** to be followed during these steps and some ***criteria*** for selecting among these strategies;
  - ✓ some ***reference models*** that describe the inputs and outputs of 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 carried out consecutively.

Phases of Database Design



Conceptual Design

- The purpose of this phase is to represent the informal requirements of an application in terms of a ***conceptual schema*** that refers to a ***conceptual data model***.
- Conceptual models allow the description of the organization of data at a high level of abstraction, without taking into account implementation details.
- In this phase, the designer must try to represent the ***information content*** of the database, without considering either the means by which this information will be implemented in the actual system, or the efficiency of the programs that makes use of this information.

Logical Design

- ***Logical design*** 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*** (usually the relational model.)
- In this phase the designer must also take into account some optimization criteria, based on the operations to be carried out on the data.
- Formal techniques for verification of the quality of the logical schema are often used. In the case of the relational data model, the most commonly used techniques is that of ***normalization***.

Physical Design

- During ***physical design***. the logical schema is completed with details of the physical implementation (file organization and indexes) on a given DBMS. The result is called the ***physical schema***, and refers to a ***physical data model***.
- Physical design is primarily concerned with performance, given information about operational requirements for the database.
- For relational DBMSs, physical design is often reduced to the identification of what indices to support for each relation.

Data and Operational Requirements

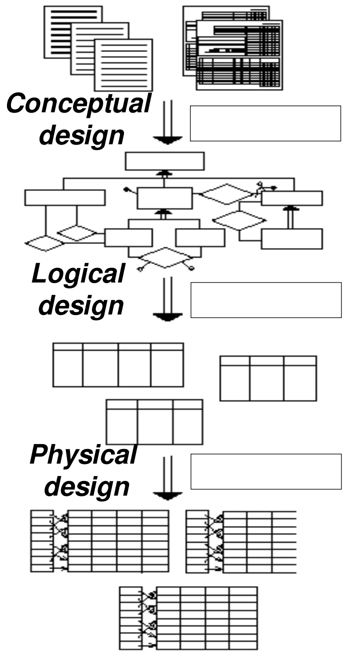
- ***Data requirements*** concern the contents of the database, and ***operational requirements*** concern the use of the database.
- In conceptual design, data requirements provide most of the information, whereas operational requirements are used only to verify that the conceptual schema is complete (that is, it contains the information necessary to carry out all the operations).
- In logical design the conceptual schema, given as input, summarizes the data requirements; whereas the the operational requirements, together with the predicted application load are used to obtain a logical schema.
- In the physical design, the logical schema and the operational requirements are used to optimize the performance of the information system. In this phase, it is necessary to take into account the characteristics of the particular DBMS used.

Outputs of Database Design

- The results of the design process of a database is not only the physical schema, but also the conceptual schema and the logical schema.
- The conceptual schema provides a high-level representation of the database, which can be very useful for documentation purposes.
- The logical schema provides a description of the contents of the database, that leaving aside the implementation aspects, is useful as a reference for writing queries and updates.

The figure shows the products of different phases in the design of a relational database, assuming the use of the best-known conceptual data model, the *Entity-Relationalship (E-R) model*.

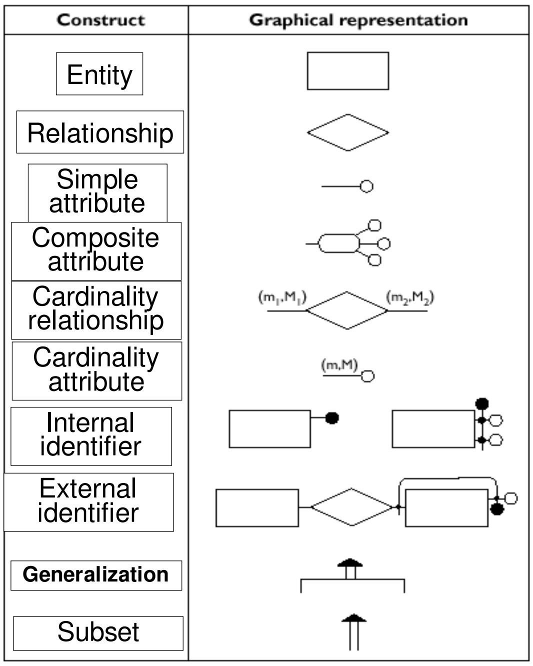
An E-R schema is represented as a diagram, This representation is then traslated into a logical schema, made up of a collection of tables. The physical schema describes data from a physical point of view (type and size of the fields), and auxilliary structures, such as indexes, are specified for efficient access to data.



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

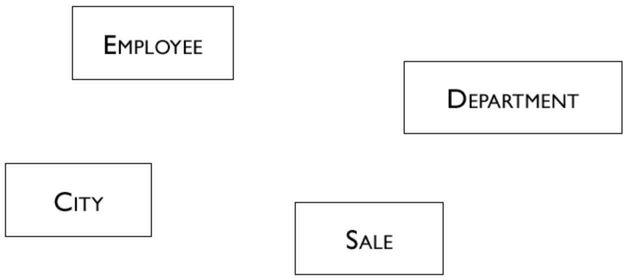
The Constructs of the E-R Model



Entities

- These represent classes of objects (facts, things, people,...) that have properties in common and an autonomous existence.
- `City`, `Department`, `Employee`, `Purchase` and `Sale` are examples of entities for a commercial organization.
- An instance of an entity is an object in the class represented by the entity.
- `Stockholm`, `Helsinki`, are examples of instances of the entity `City`, and the employees `Peterson` and `Johanson` are examples of instances of the `Employee` entity.
- The E-R model is very different from the relational model in which it is not possible to represent an object without knowing its properties (an employee is represented by a tuple containing the name, surname, age, and other attributes.)

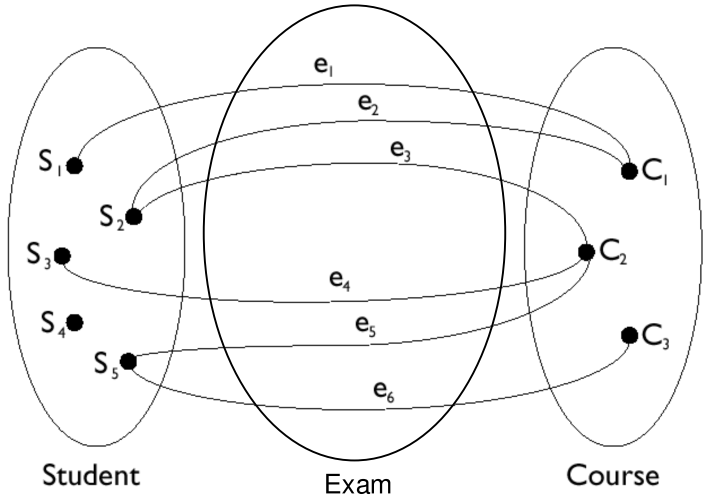
Examples of Entities



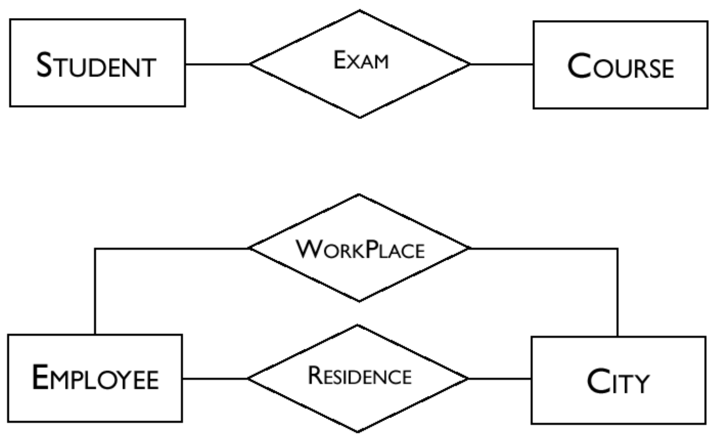
Relationships

- They represent semantic 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 instance of a relationship is an n-tuple made up of instances of entities, one for each of the entities involved.
- The pair of objects made up of the employee named `Johanssen` and the city `Stockholm`, or the pair of objects made from the employee `Peterson` and the city `Oslo`, are examples of instances in the relationship `Residence`.

Example of Instances for Exam

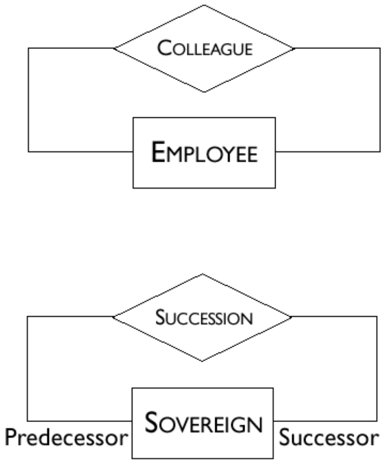


Examples of Relationships

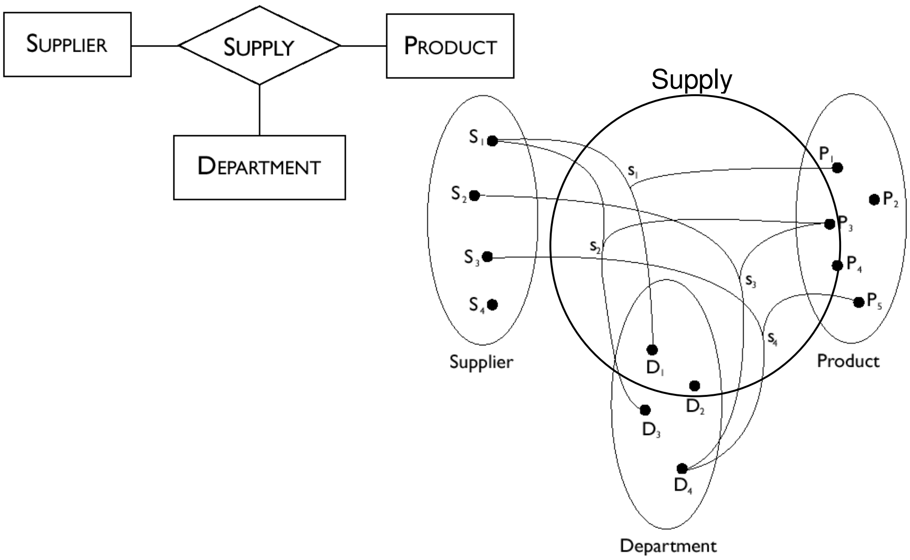


Recursive Relationships

- Recursive relationships (i.e., relationships between an entity and itself) are also possible.
- Note in the second example that the the relationship is not symmetrical. In this case it is necessary to indicate explicitly the two roles the entity plays in the relationship.



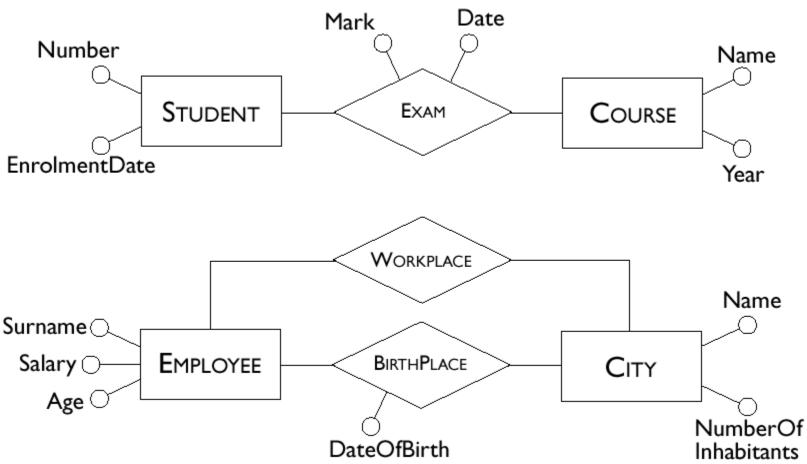
Ternary Relationships



Attributes

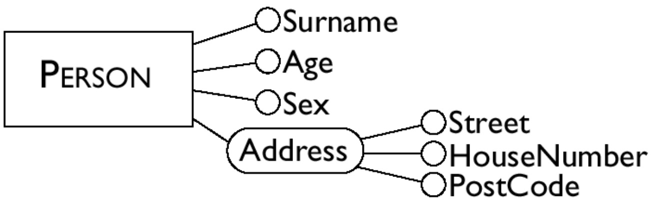
- These describe the elementary properties of entities or relationships.
- For example, `Surname`, `Salary` and `Age` are possible attributes of the `Employee` entity, while `Date` and `Mark` are possible attributes for the relationship `Exam` between `Student` and `Course`.
- An attribute associates with each instance 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.

Attribute Examples

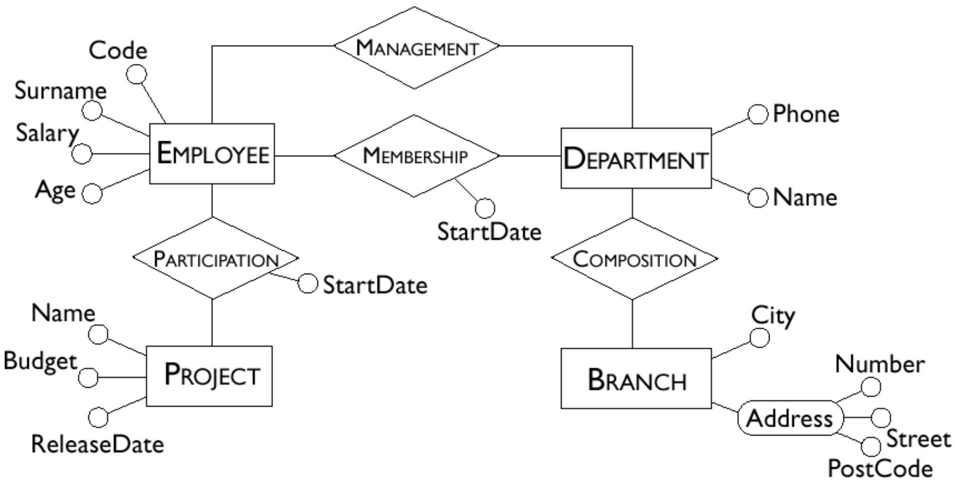


Composite Attributes

- It is sometimes convenient to group attributes of the same entity or relationship that have closely connected meanings or uses. Such groupings are called ***composite attributes***.



Schema with Attributes



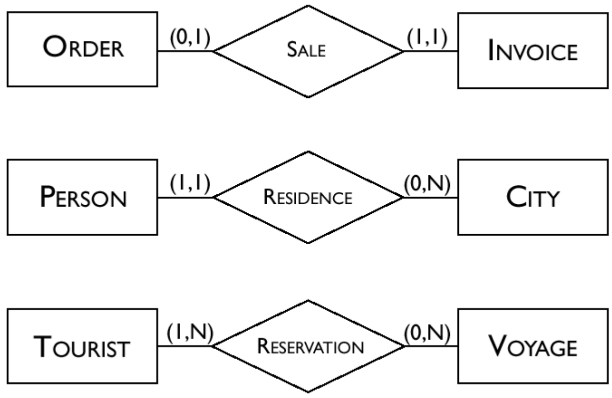
Cardinalities

- These are specified for each entity participating in a relationship and describe the maximum and minimum number of relationship occurrences in which an entity occurrence can participate.
- Cardinalities state how many times can an entity instance participate in instances of a given relationship.

Cardinalities (cont'd)

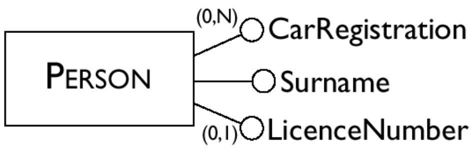
- In principle, a cardinality is any pair of non-negative integers (n,m) such that n≤m. or a pair of the form (n,N) where N means "any number".
- If minimum cardinality is 0, we say that entity participation in a relationship is optional. If minimum cardinality is 1, we say that entity participation in a relationship is mandatory.
- If maximum cardinality is 1, each instance of the entity is associated at most with a single instance of the relationship; if maximum cardinality is N, then each instance of the entity is associated with an arbitrary number of instances of the relationship.

Cardinality Examples



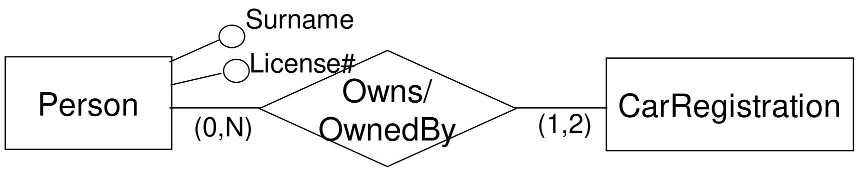
Cardinalities of Attributes

- They are specified for the attributes of entities (or relationships) and describe the minimum and maximum number of values of the attribute associated with instances of an entity or a relationship.
- In most cases, the cardinality of an attribute is equal to (1,1) and is omitted (**single-valued** attributes)
- The value of a certain attribute however, may also be null, or there may exist several values of a certain attribute for an entity instance (**multi-valued** attributes)



Cardinalities (cont'd)

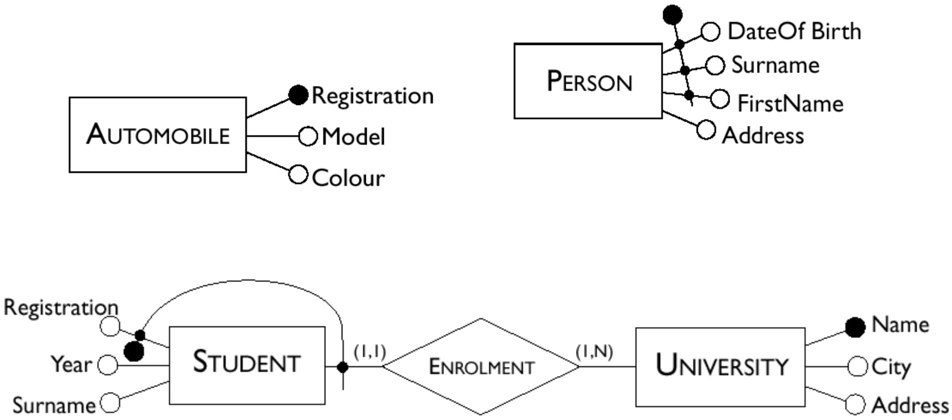
- Multi-valued attributes should be used with great caution, because they represent situations that can be modelled, sometimes, with additional entities linked by one-to-many (or many-to-many) relationships to the entity to which they refer.



Identifiers

- These 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 are involved in the identification.This is called an ***external*** identifier.

Examples of Identifiers

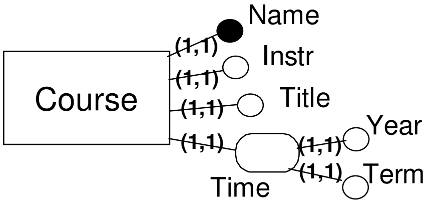


General Observations on Identifiers

- 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 may involve an entity that is in 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 Course Database

- We want a database about the courses offered at the University of Trento each year. For example, BDSI.1 was given last year 1st term, and this year as well.

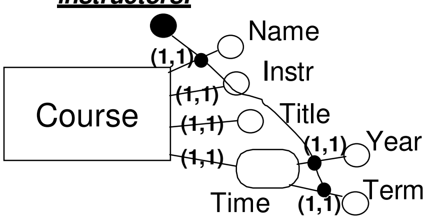


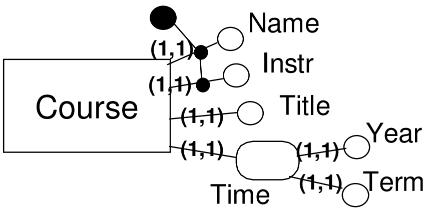
Does this make sense?

No, because a course can be given more than once!

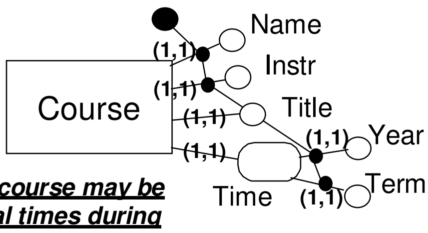
How About...

No, because a course can be given more than once during a term, with different instructors!



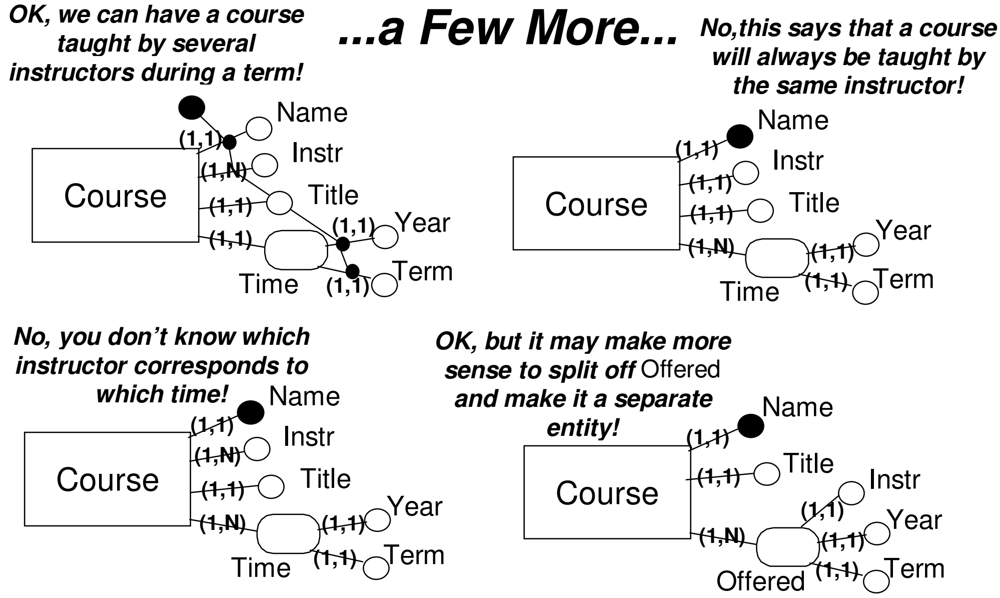


No, because an instructor may teach a course more than once!



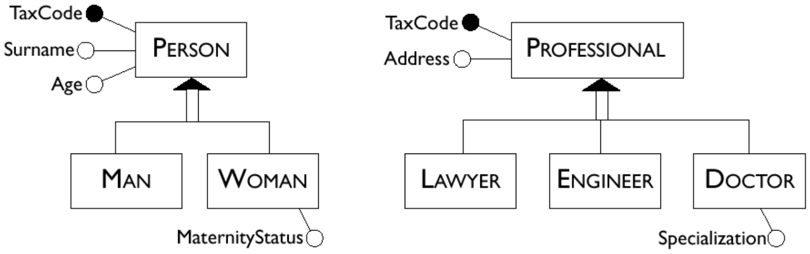
This is OK, a course may be taught several times during a term, with different instructors

...a Few More...



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.



Properties of Generalization

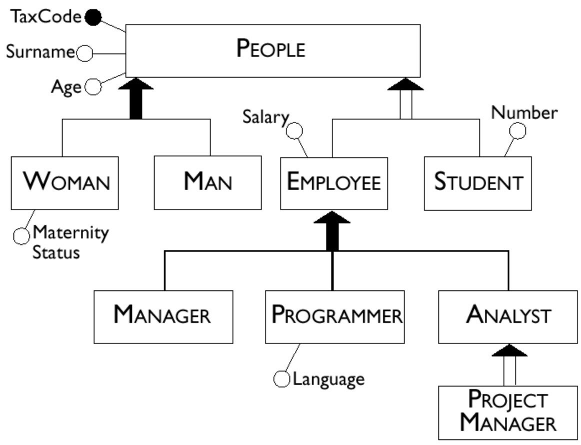
- Every instance of a child entity is also an instance of the parent entity.
- Every property of the parent entity (attributes, identifiers, relationships and other generalizations) is also a property of a child entity. This property of generalizations is known as ***inheritance***.

Types of Generalizations

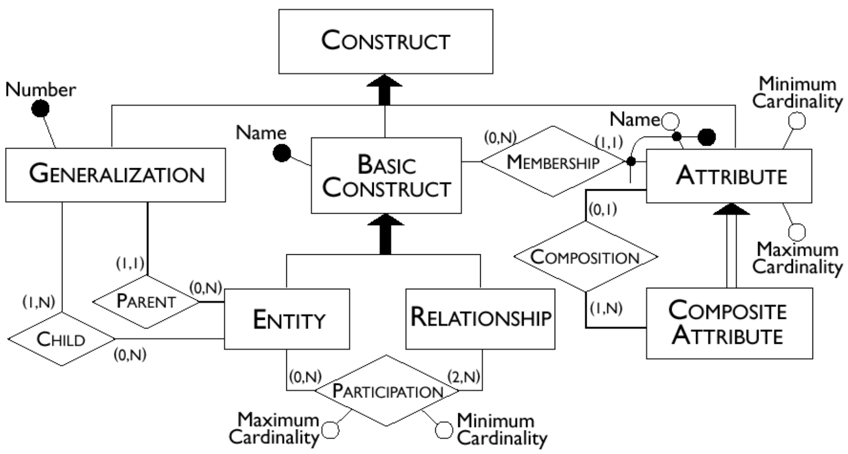
- A generalization is ***total*** if every instance of the parent entity is also an instance of one of its children, otherwise it is ***partial***.
- A generalization is ***exclusive*** if every instance of the parent entity is at most an instance of one of the children, otherwise it is ***overlapping***.
- The generalization `Person`, of `Man` and `Woman` is total (the sets of men and the women constitute 'all' the people) and exclusive (a person is either a man or a woman).
- The generalization `Vehicle` of `Automobile` and `Bicycle` is partial and exclusive, because there are other types of vehicle (for example, motor bike) that are neither cars nor bicycles.
- The generalization `Person` of `Student` and `Employee` is partial and overlapping, because there are students who are also employed.

Generalization Hierarchies

- Total generalization is usually represented by a solid arrow.
- We have a *hierarchy* of generalizations when we have several levels of generalizations.



The E-R Model, as an E-R Diagram



Documentation of E-R Schemas

- An Entity-Relationship schema is rarely sufficient by itself to represent all the aspects of an application in detail;
- It is therefore important to complement 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;
- A widely-used documentation concept for conceptual schemas is the ***business rule***.

Business Rules

- Business rules are used to describe the properties of an application, e.g., the fact that an employee cannot earn more than his or her manager.
- A business rule can be:
  - ✓ the ***description*** *of a concept* relevant to the application (also known as a ***business object***),
  - ✓ an ***integrity constraint*** on the data of the application,
  - ✓ a ***derivation rule***, whereby information can be derived from other information within a schema.

Documentation Techniques

- Descriptive business rules can be organized as a ***data dictionary***. This is made up of two tables: the first describes the entities of the schema, the others describes the relationships.
- Business rules that describe constraints can be expressed in the following form:

  <concept> must/must not <expression on concepts>
- Business rules that describe derivations can be expressed in the following form:

  <concept> is obtained by <operations on concepts>

Example of a Data Dictionary

Entity	Description	Attributes	Identifier
EMPLOYEE	Employee working in the company.	Code, Surname, Salary, Age	Code
PROJECT	Company project on which employees are working.	Name, Budget, ReleaseDate	Name
....	....	....	....

Relationship	Description	Entities involved	Attributes
MANAGEMENT	Associate a manager with a department.	Employee (0,1), Department (1,1)	
MEMBERSHIP	Associate an employee with a department.	Employee (0,1) Department (1,N)	StartDate
....	....	....	....

Examples of Business Rules

Constraints
(BR1) The manager of a department must belong to that department. **(BR2)** An employee must not have a salary greater than that of the manager of the department to which he or she belongs. **(BR3)** A department of the Rome branch must be managed by an employee with more than 10 years' employment with the company. **(BR4)** An employee who does not belong to a particular department must not participate in any project. ....
Derivations
(BR5) The budget for a project is obtained by multiplying the sum of the salaries of the employees who are working on it by 3. ....