2a. The (Extended) Entity-Relationship Model

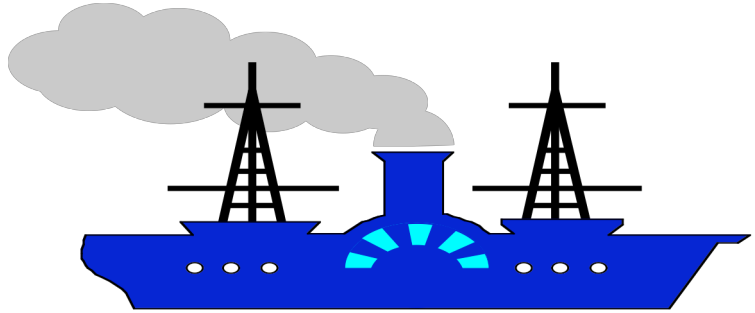
The Extended Entity-Relationship (EER) Model

Entities, Relationships and Attributes

Cardinalities, Identifiers and Generalization

Documentation of EER Diagrams and Business Rules

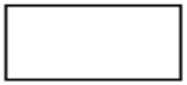
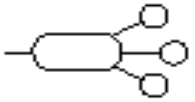
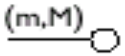
Modeling Strategies



The Extended Entity Relationship Mode

- The *Extended Entity-Relationship* (EER) *model* is a *conceptual* (or *semantic*) data model, capable of describing the data requirements for a new information system in a direct and easy to understand graphical notation.
- Data requirements for a database are described in terms of a *conceptual schema,* using the EER model.
- EER schemata are comparable to UML class diagrams.
- Actually, what we will be discussing is an extension of Peter Chen's proposal (hence "extended" ER).

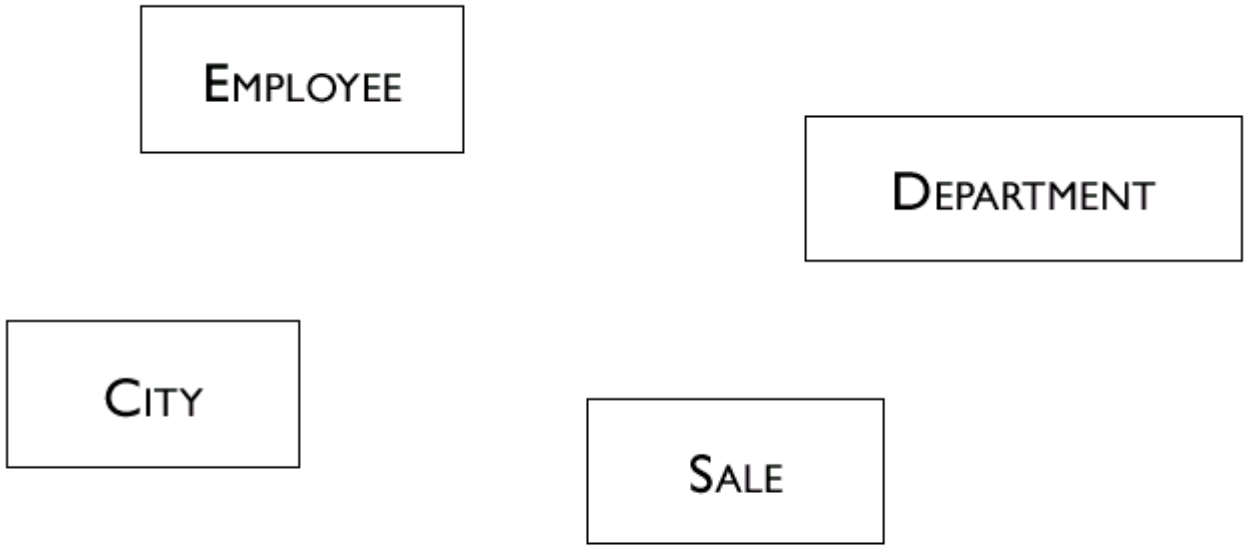
The Constructs of the EER Model

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,...) 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 represents 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 EER model is very different from the relational model in a number of ways; for example, in EER it is not possible to represent an object without knowing its properties, but in the relational model you need to know its key attributes.

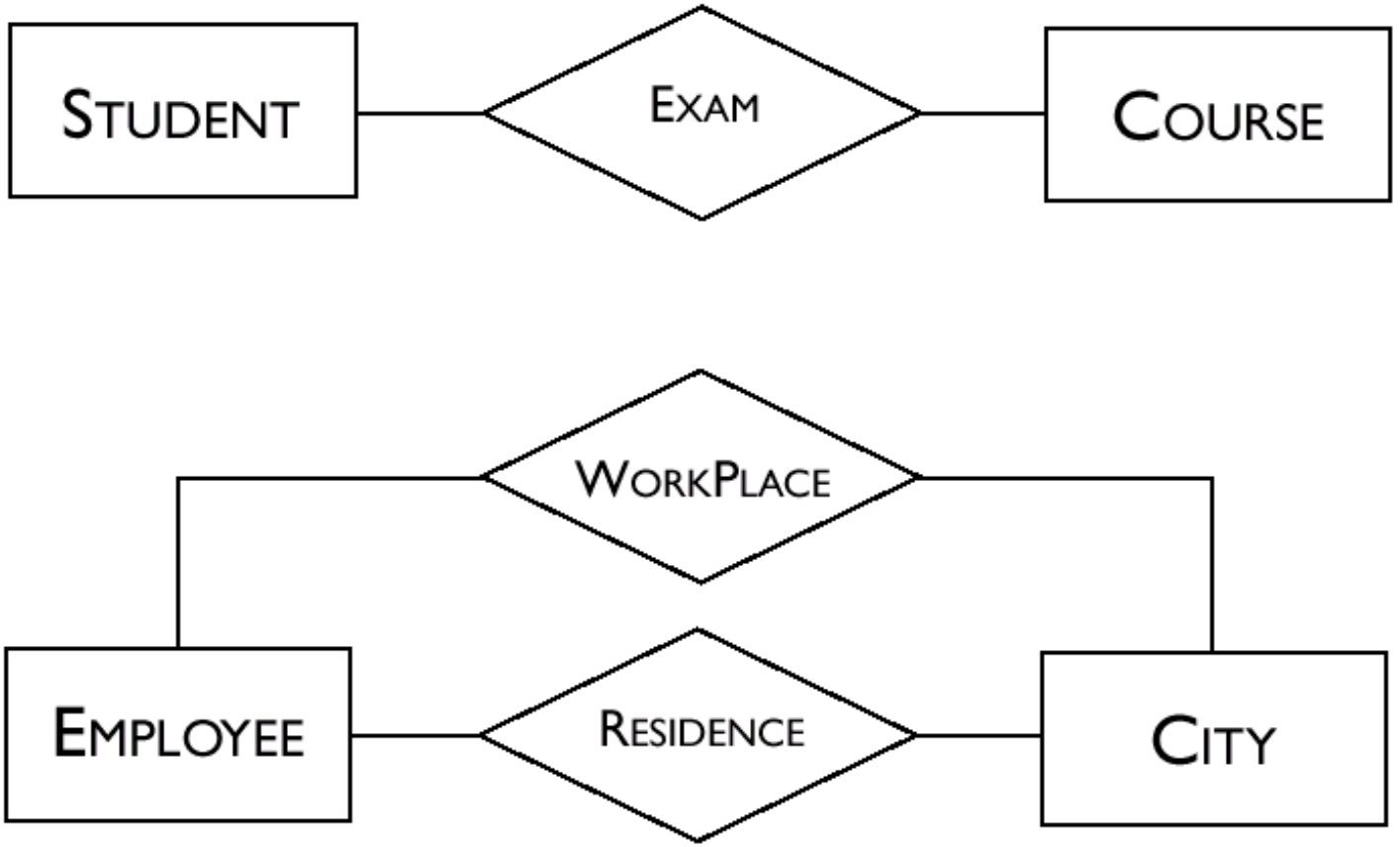
Examples of Entities



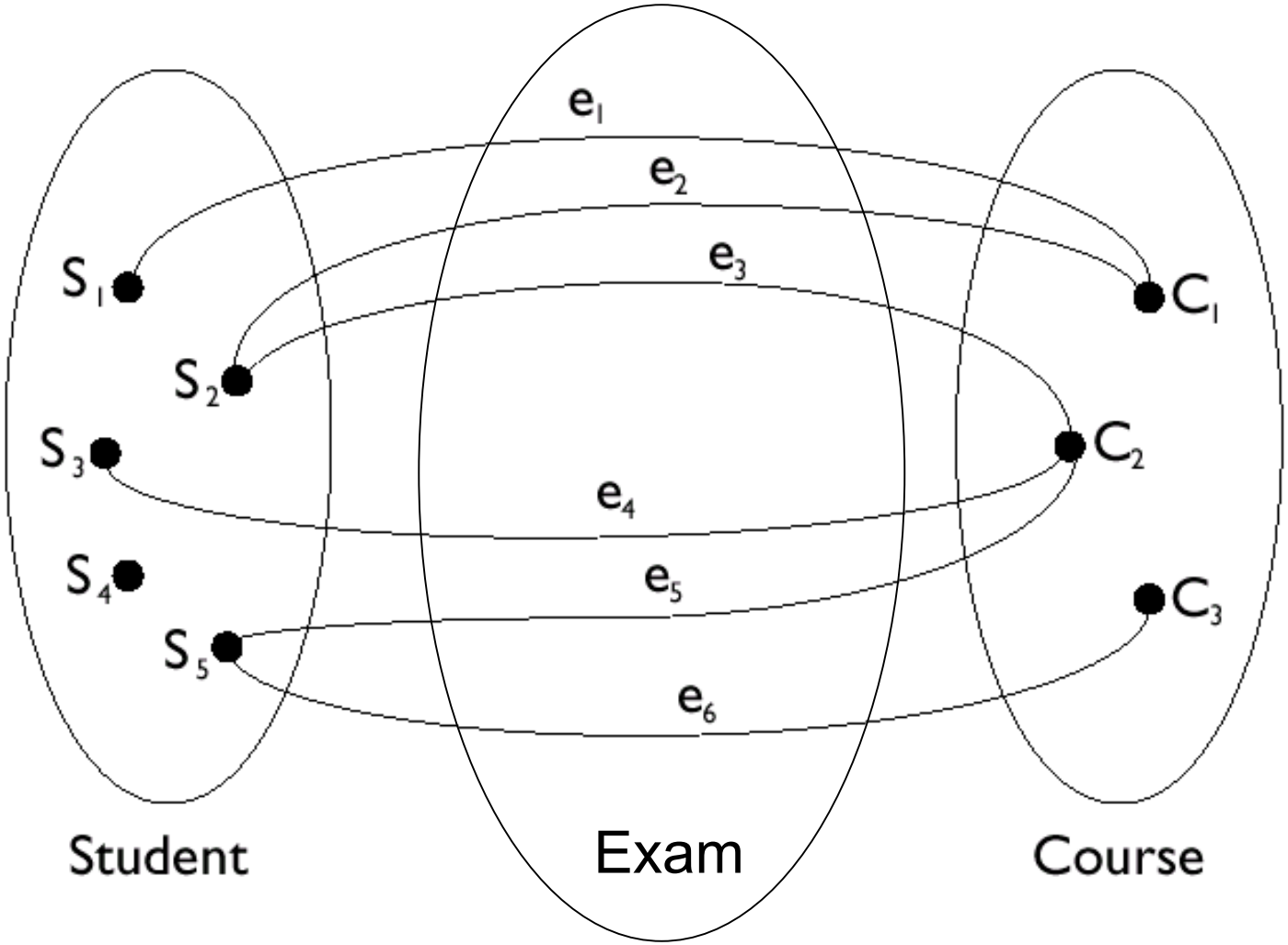
Relationships

- 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 `(Johanssen,Stockholm)`, or the pair `(Peterson,Oslo)`, are examples of instances of the relationship `Residence`.

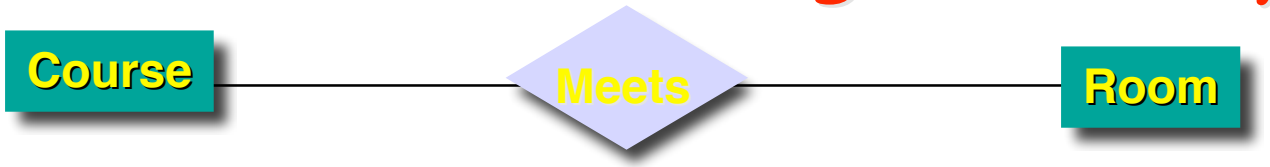
Examples of Relationships



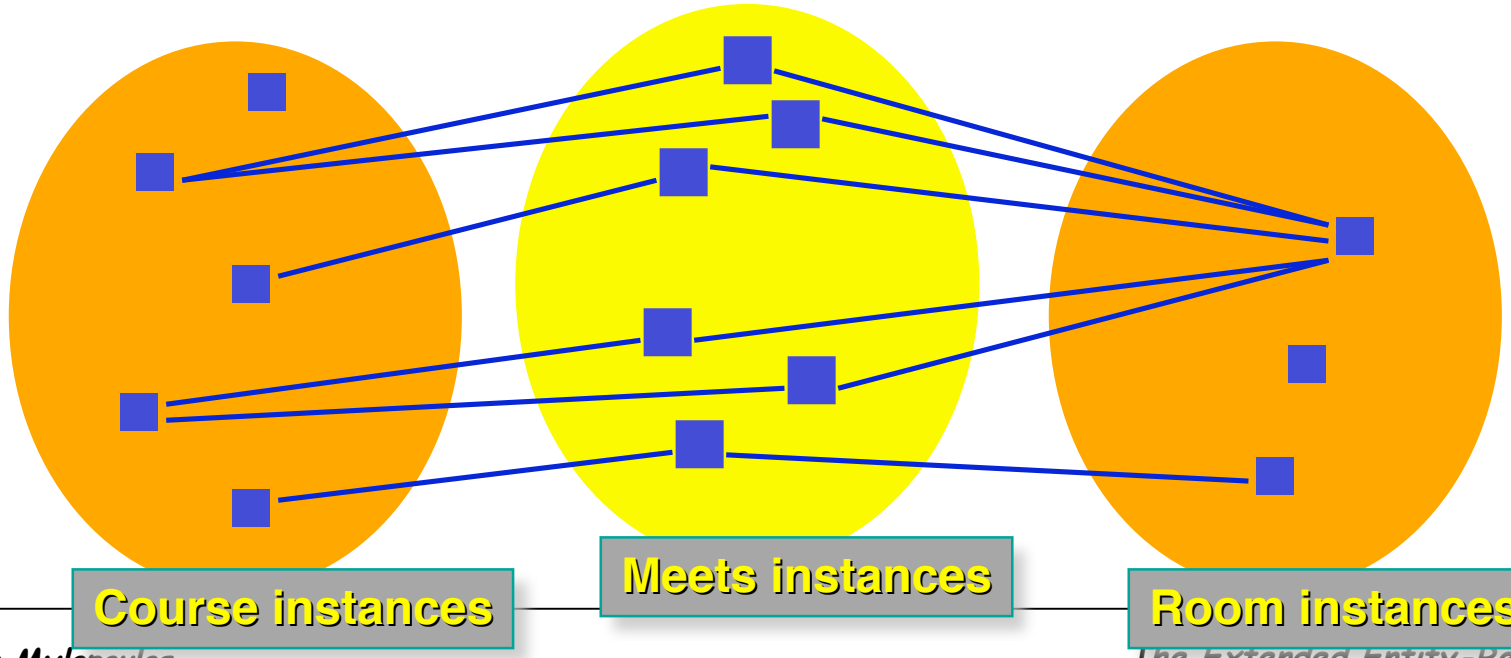
Example of Instances for Exam



What Does An EER Diagram Really Mean?

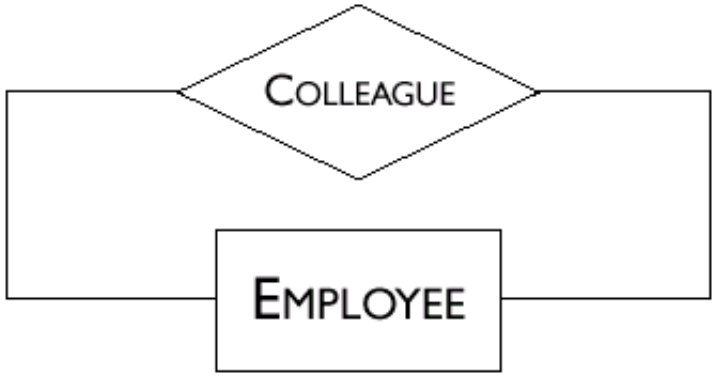


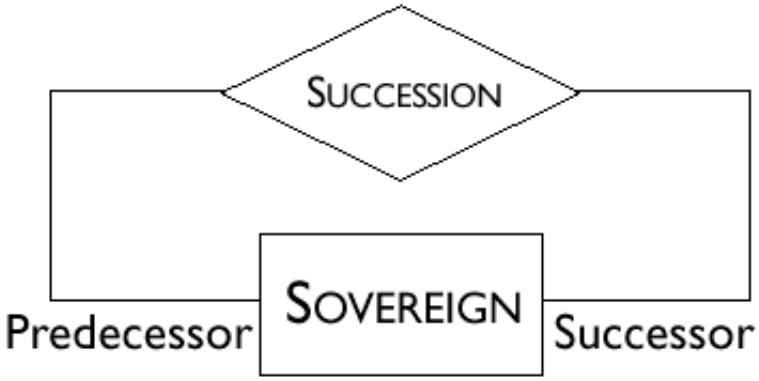
- **Course** and **Room** are entities. Their instances describe particular cours (e.g., CSC340S) and particular rooms (e.g., WB116).
- **Meets** is a relationship. Its instances describe particular meetings. Ea meeting has exactly one associated course and room



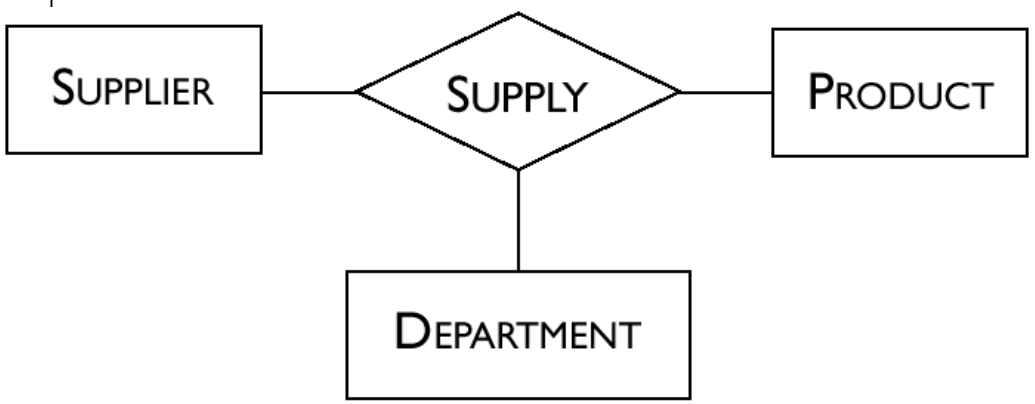
Recursive Relationships

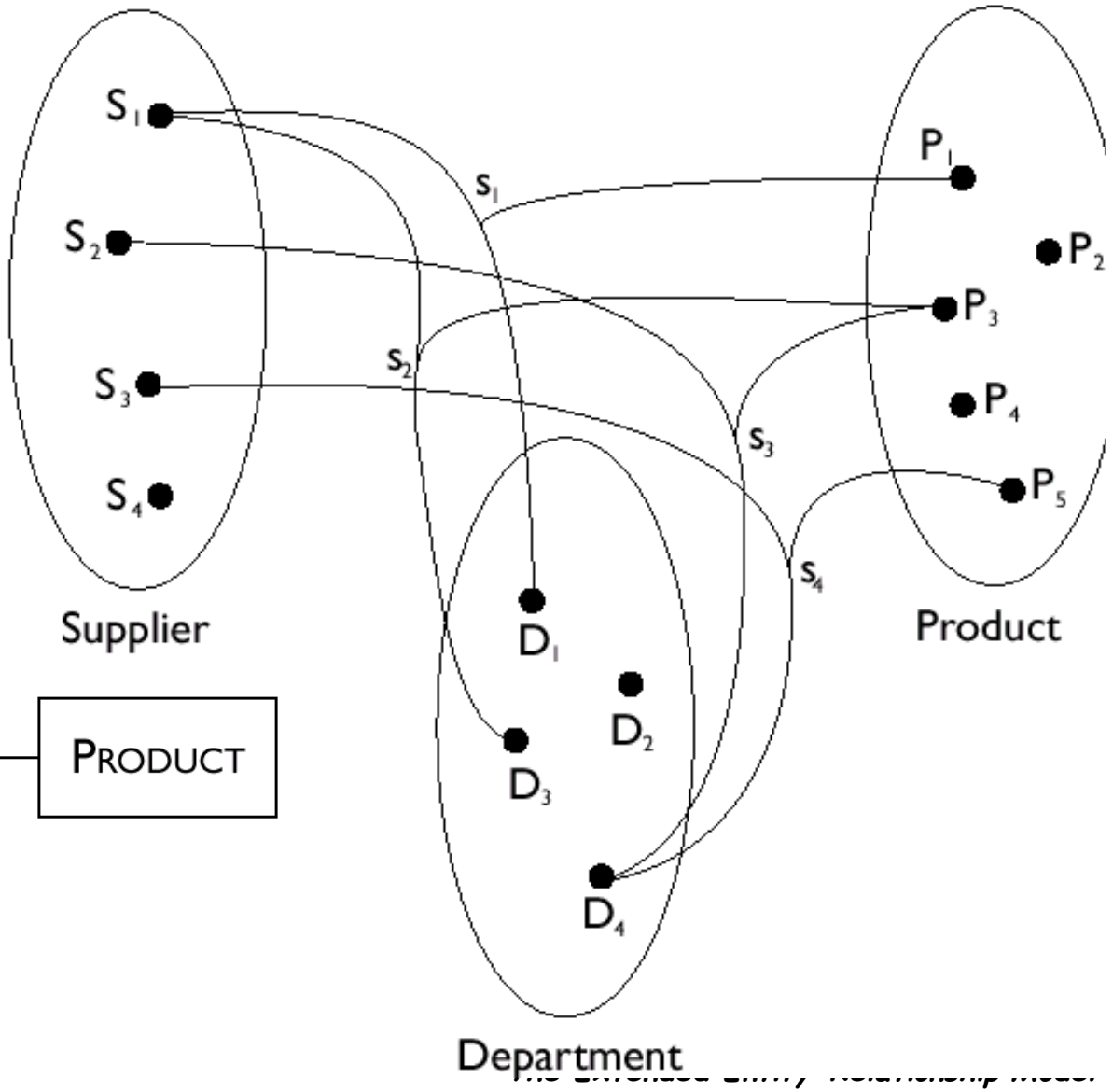
- *Recursive* relationships are also possible, that is relationships between an entity and itself.
- Note in the second example that the relationship is not symmetric. In this case it is necessary to indicate the two *roles* that the entity involved plays in the relationship.



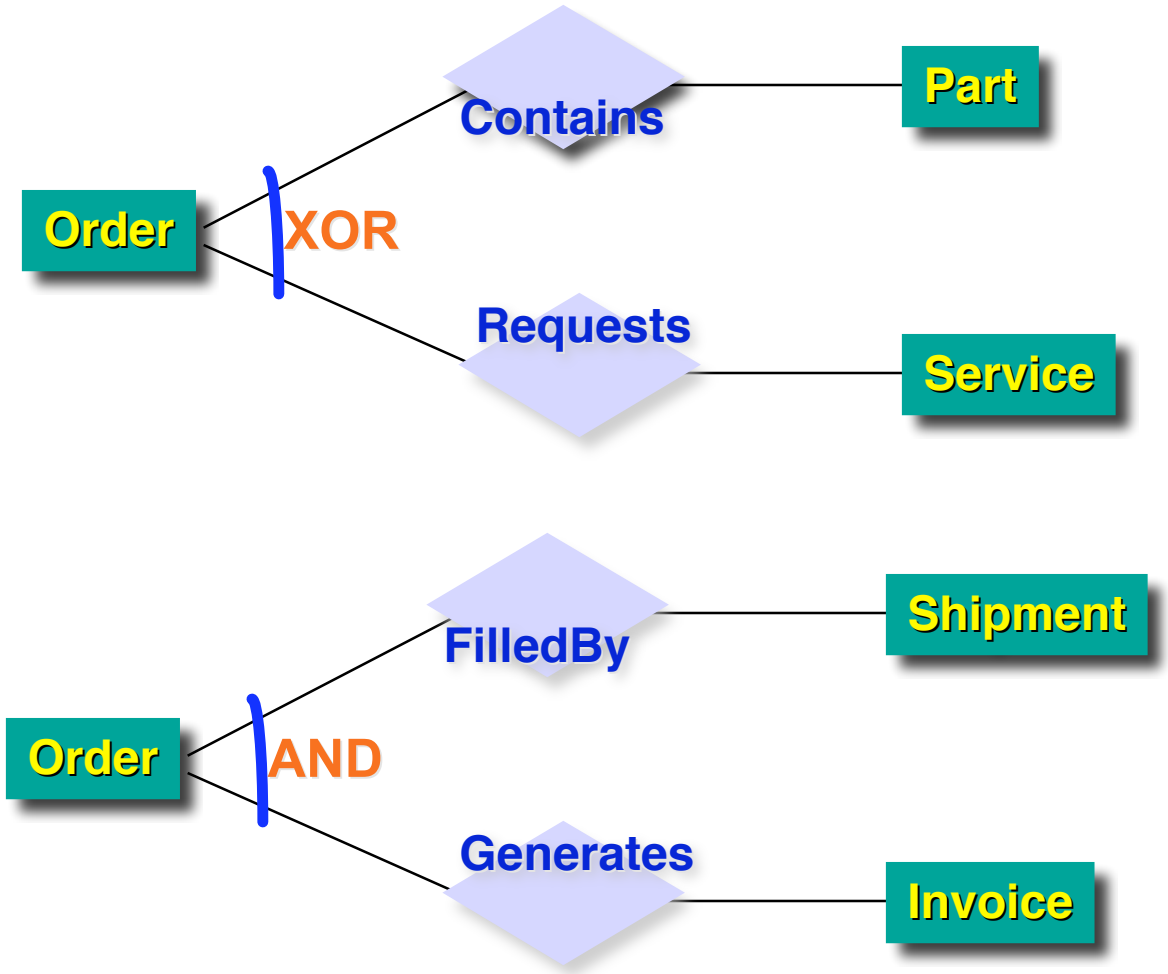


Ternary Relationships



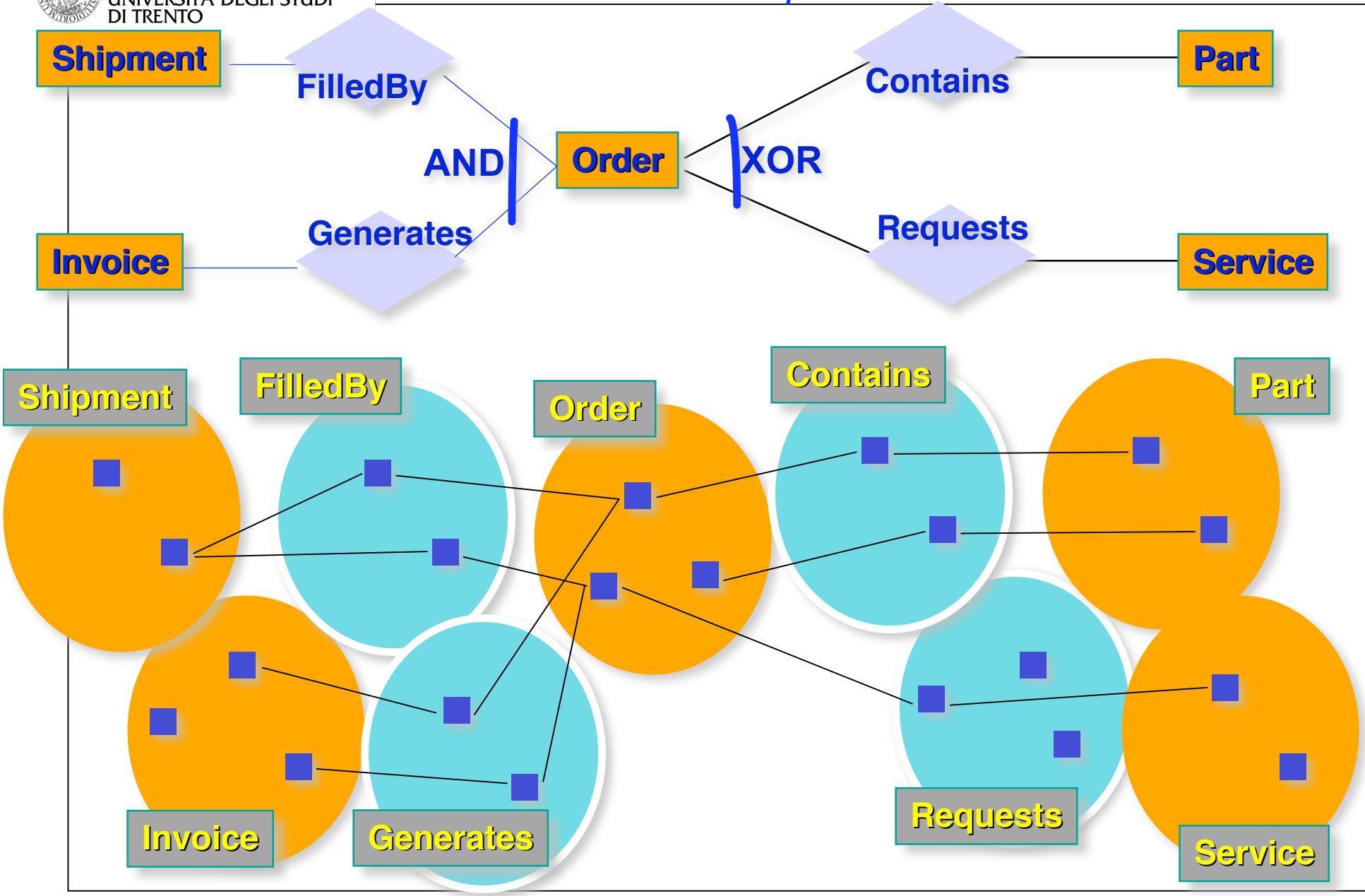


AND/XOR for Relationships



"Orders either order a part or request a service, but not both"

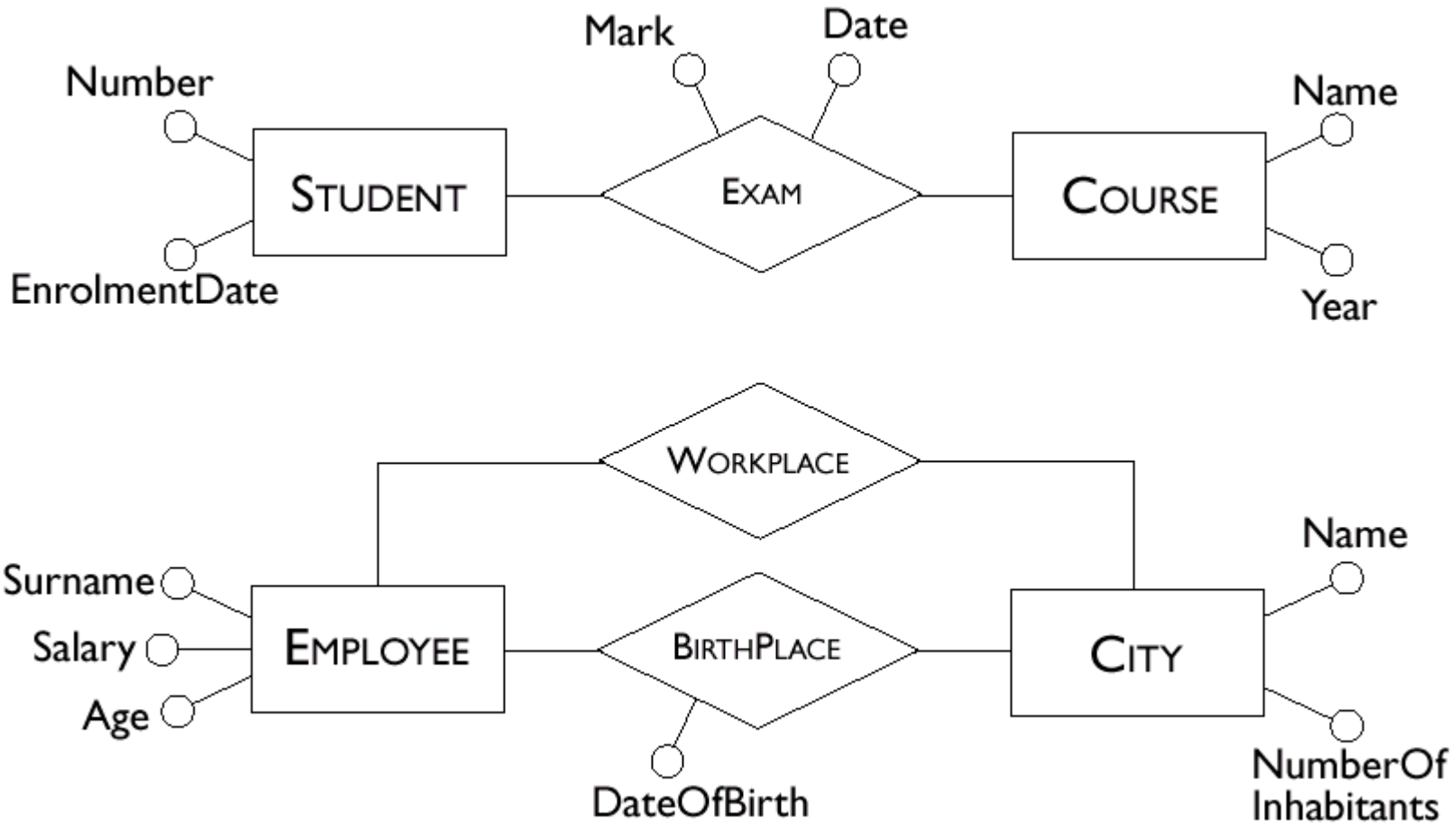
"For any given order, whenever there is at least one invoice there is also at least one shipment and vice versa"


Shipment
FilledBy
Order
Contains
Part
AND
XOR
Invoice
Generates
Requests
Service
Shipment
FilledBy
Order
Contains
Part
Invoice
Generates
Requests
Service

Attributes

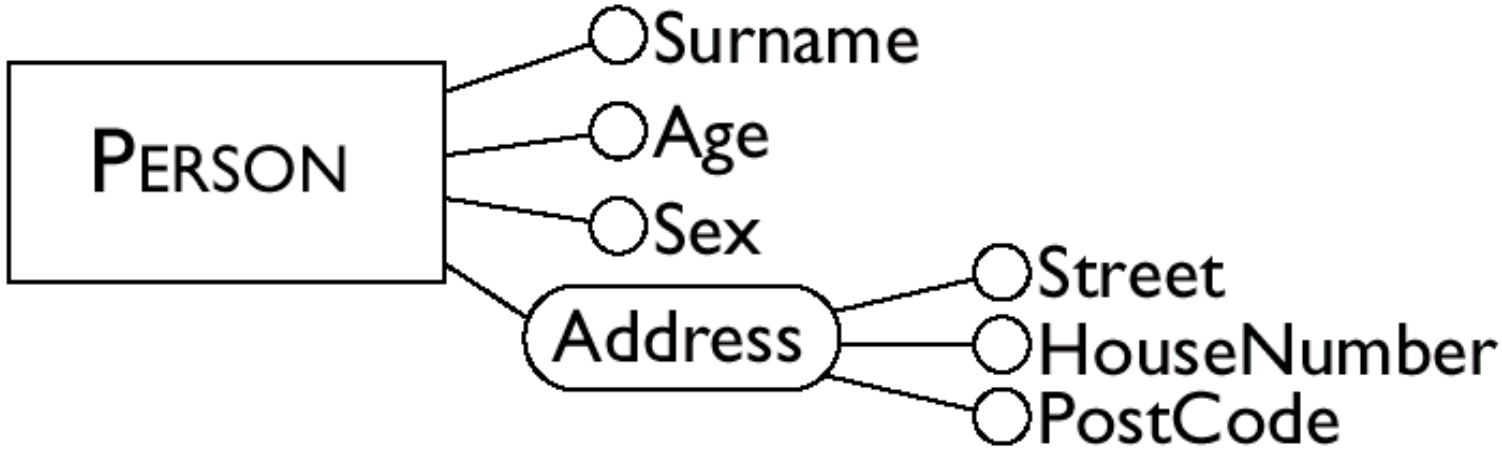
- These describe the elementary properties of entities or relationships.
- For example, `Surname`, `Salary` and `Age` are possible attributes of the `Employee` entity, while `Date` and `Mark` are possible attributes for the relationship `Exam` between `Student` and `Course`.
- An attribute associates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

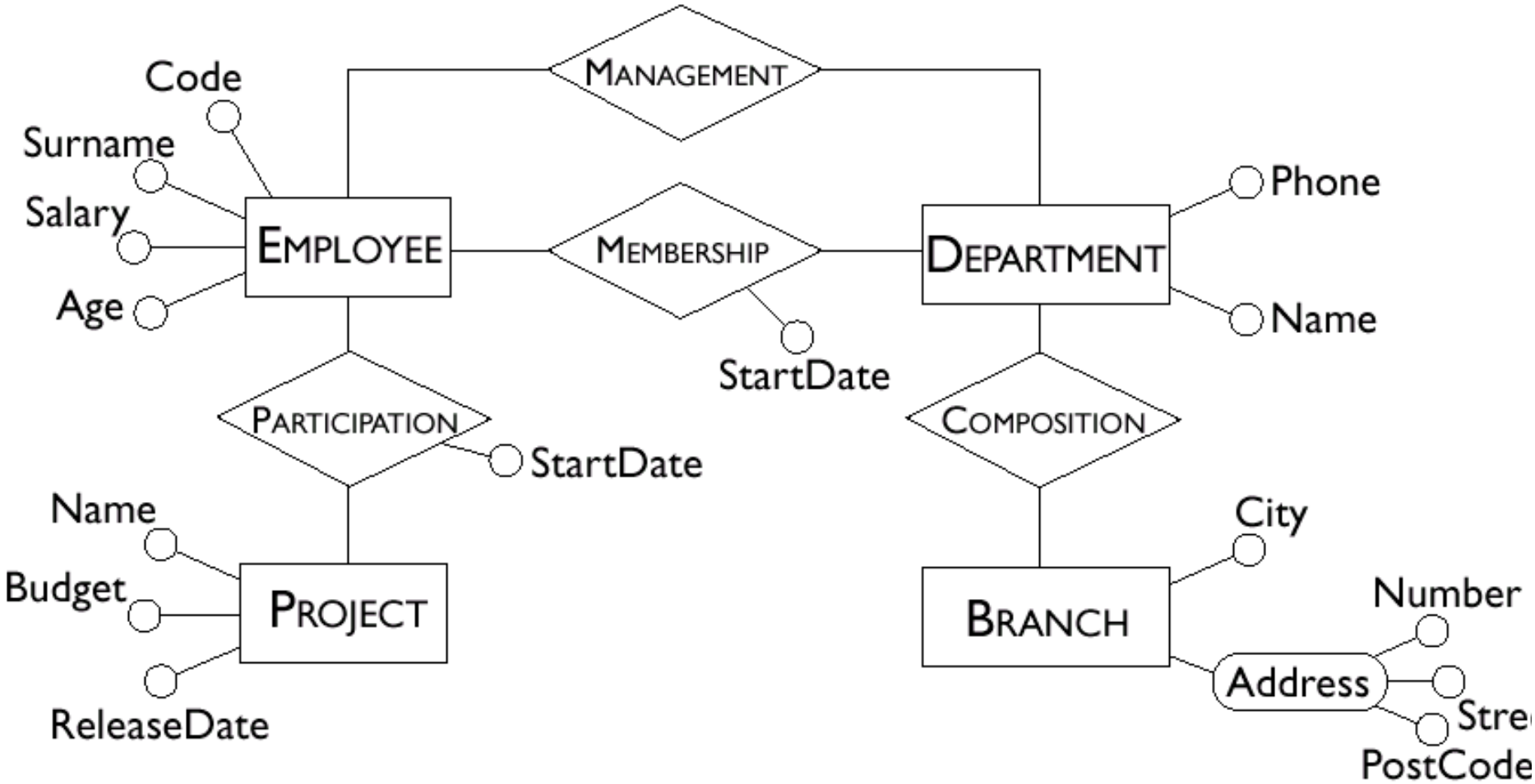


Composite Attributes

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



Schema with Attributes



Cardinalities

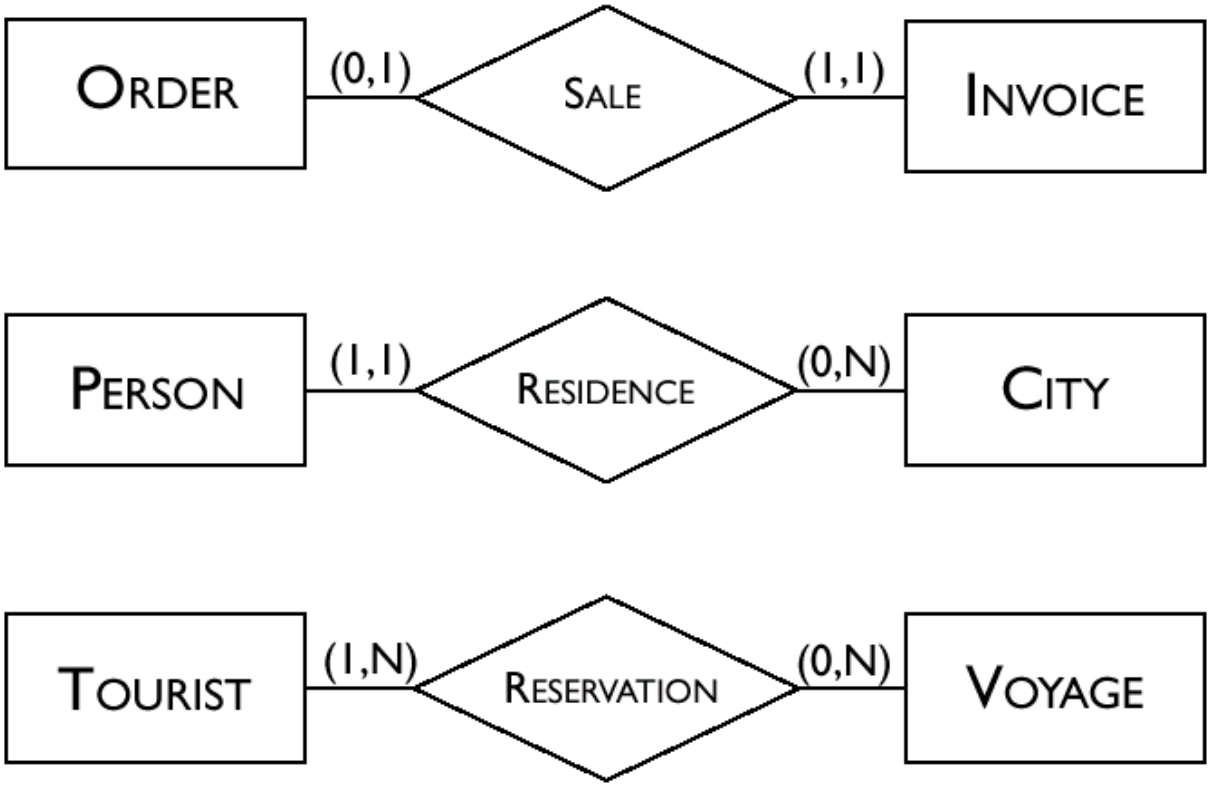
- These are specified for each entity participating in a relationship and describe the maximum and minimum number of relationship occurrences in which an entity occurrence can participate.
- Cardinalities state how many times can an entity instance participate in instances of a given relationship.

"An employee can participate in 1 to 5 assignments"
"A task can participate in 0 to 50 assignments"

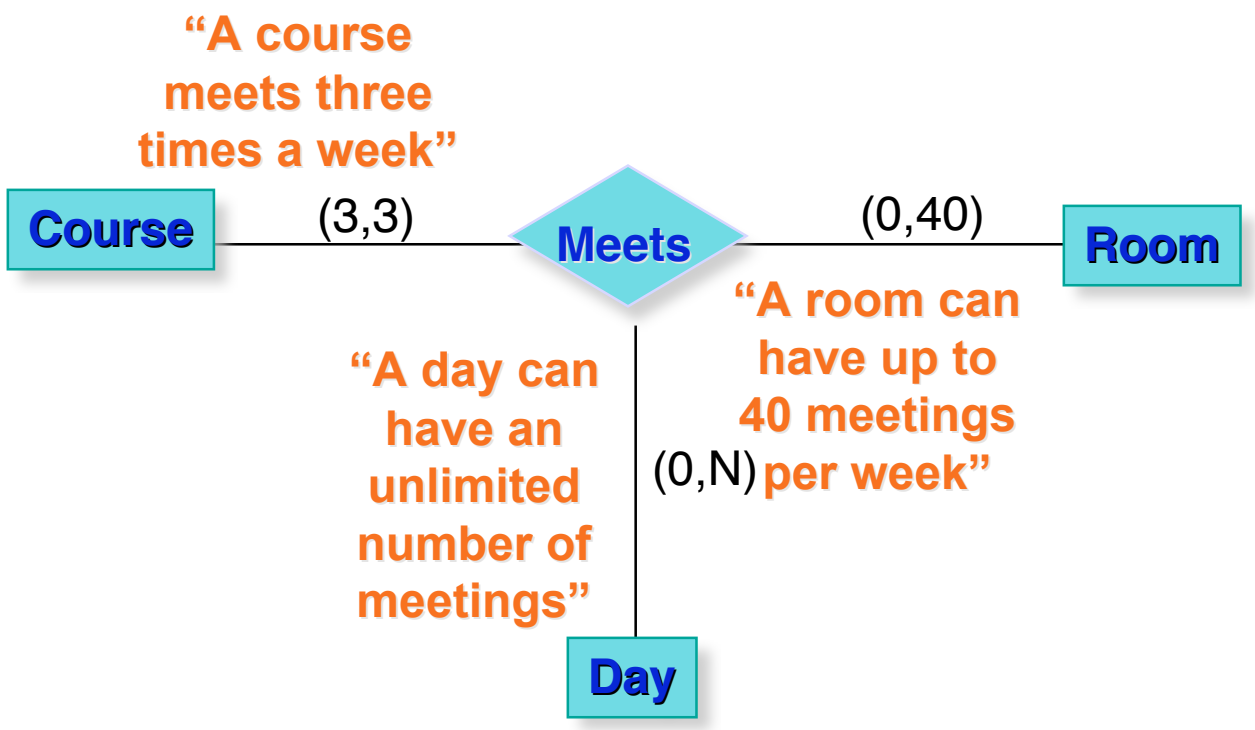
Cardinalities (cont'd)

- 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

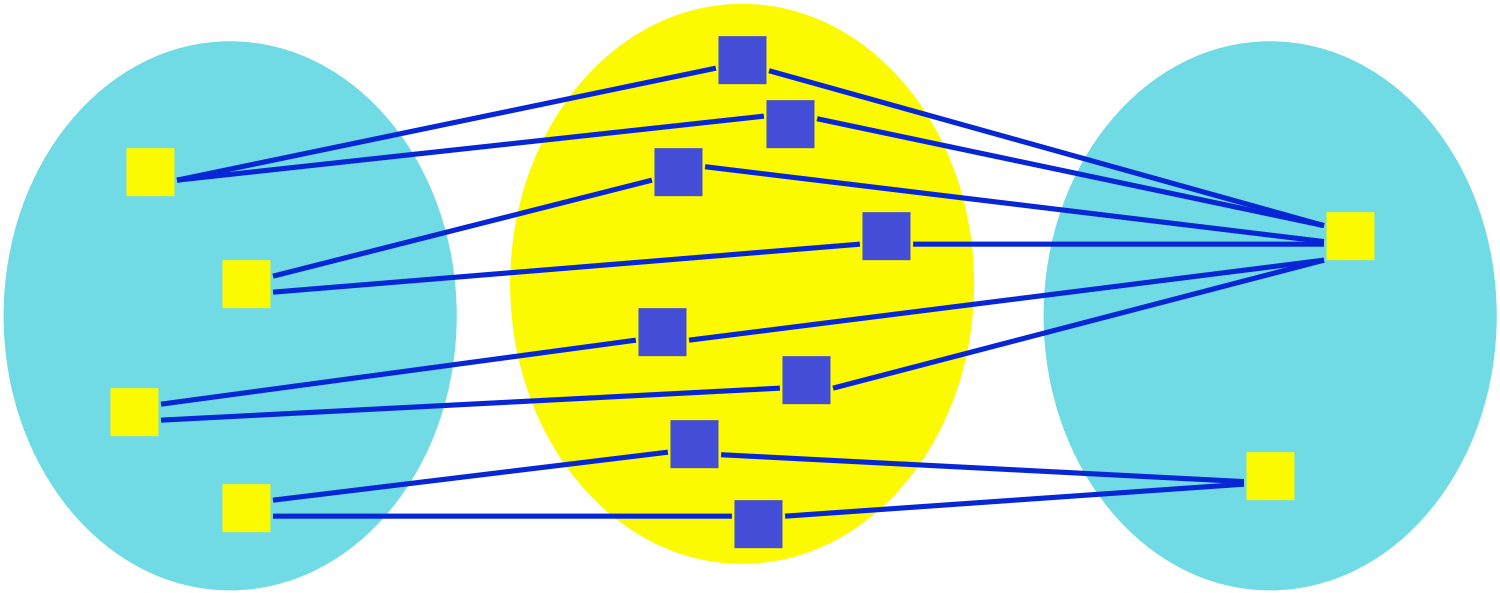


Cardinality Example

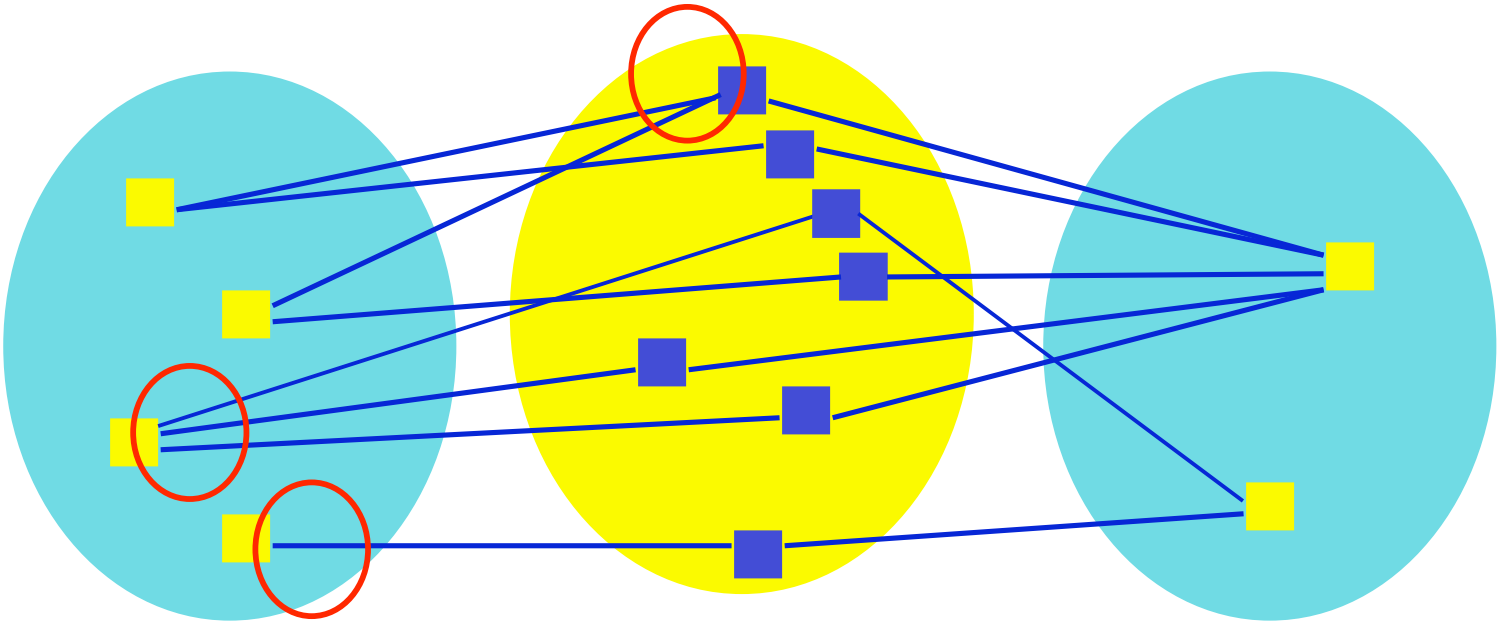


Instantiating ER Diagrams

→ An EER diagram specifies what states are possible in the world that is being modeled

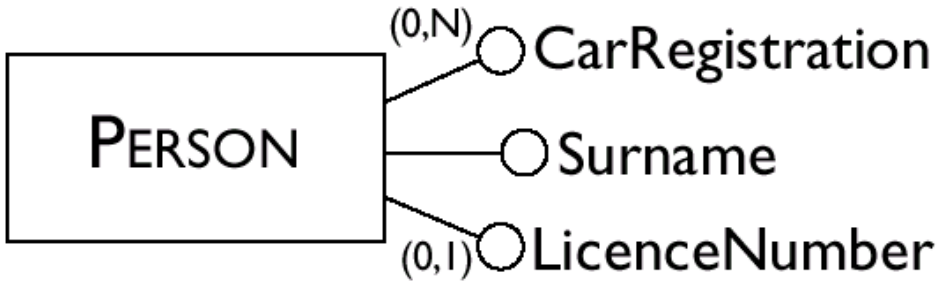


Illegal Instantiations



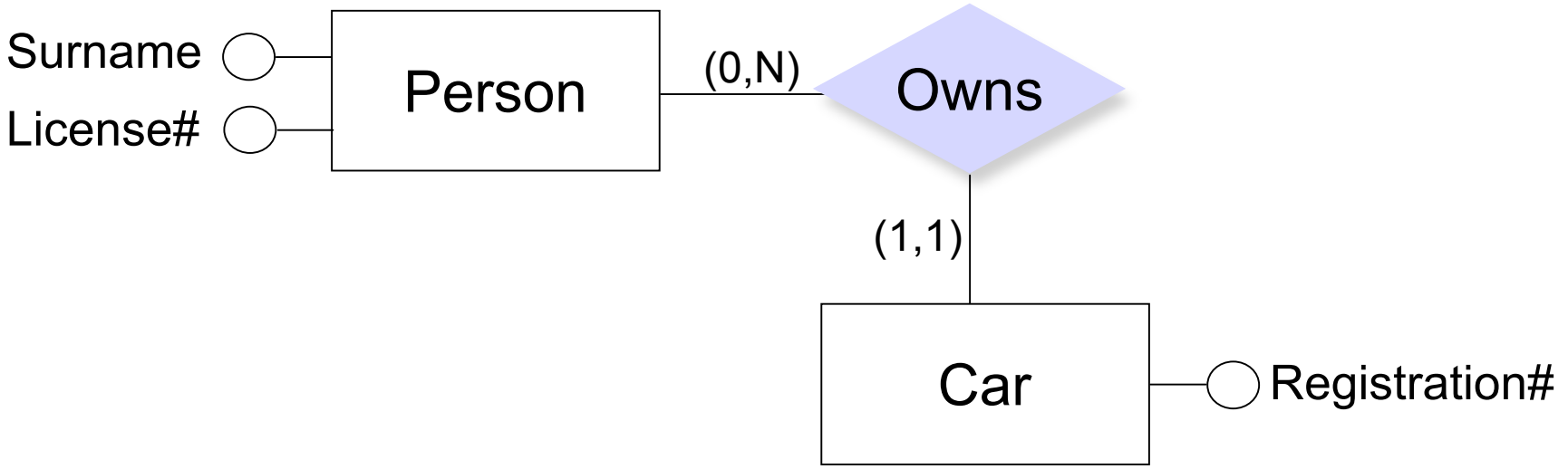
Cardinalities of Attributes

- 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)

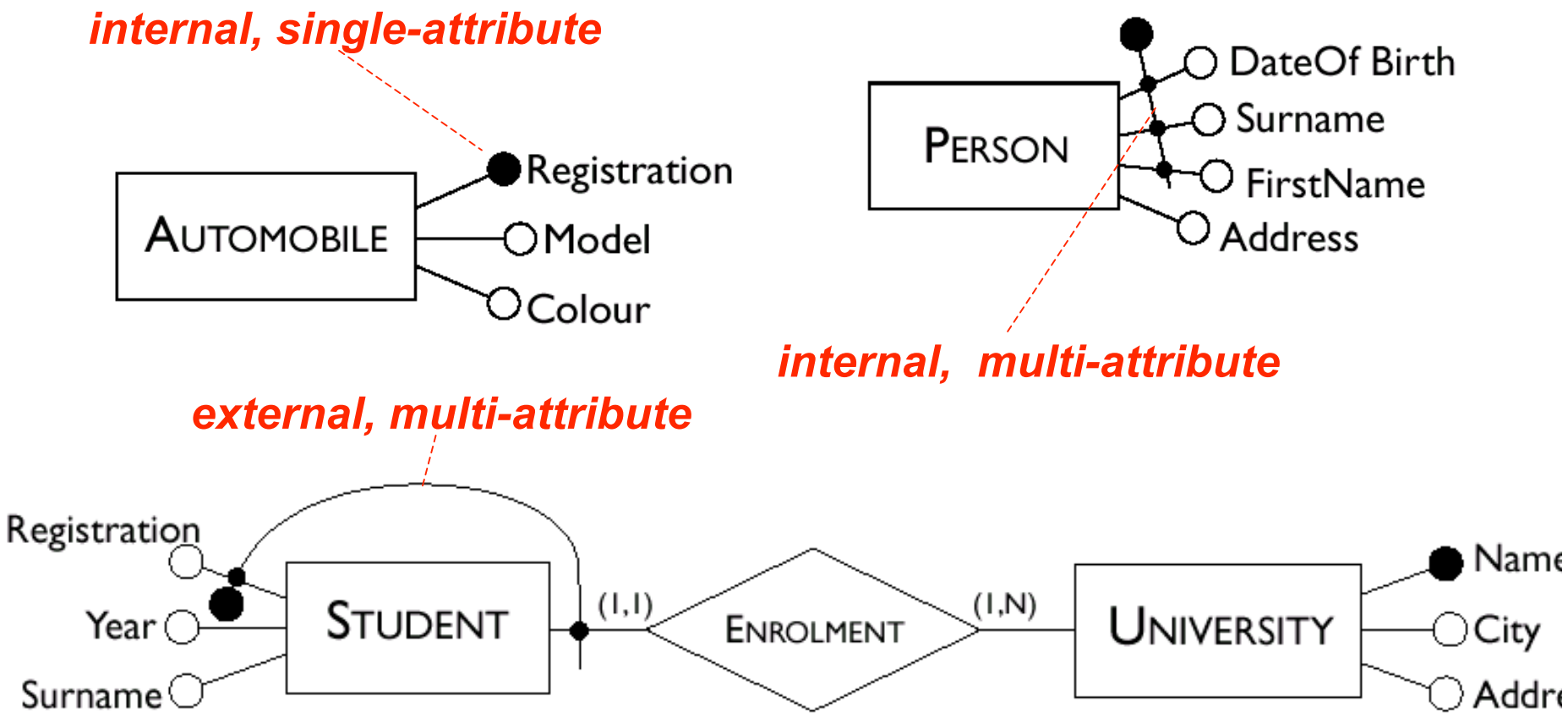
→ Multi-valued attributes should be used with great caution, because they represent situations that can be modelled in many cases with additional entities linked by one-to-many (or many-to-many) relationships to the entity to which they refer.



Identifiers

- *Identifiers* (or *keys*) consist of one or more attributes which identify uniquely instances of an entity.
- 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.
- Sometimes, however, the attributes of an entity are not sufficient to identify its instances unambiguously and other entities are involved in the identification. Identifiers of this type are called *external* identifiers.
- An identifier for a relationship consists of identifiers for all the entities it relates. For example, the identifier for the relationship (Person-) Owns(-Car) is a combination of the Person and Car identifiers.

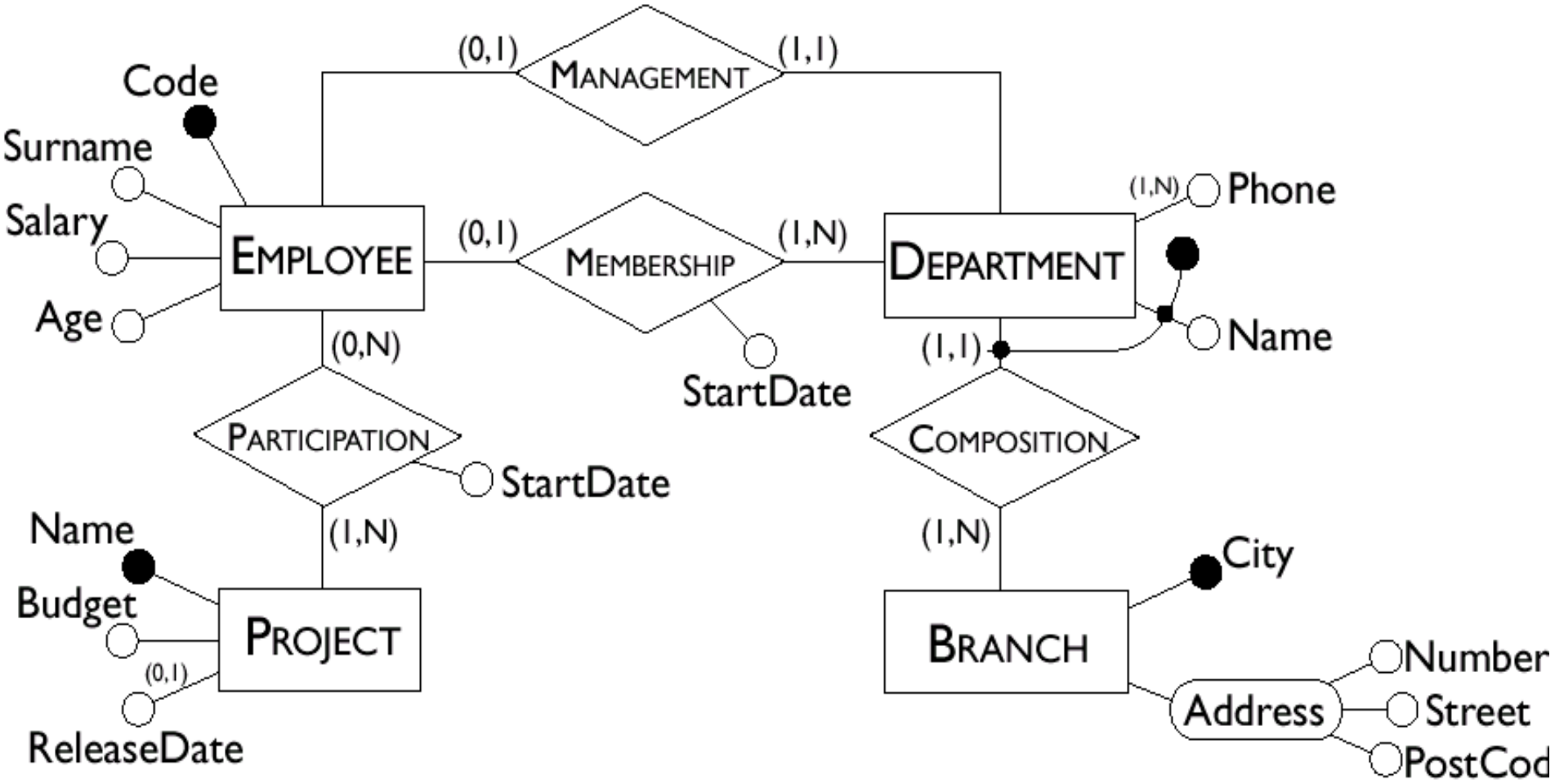
Examples of Identifiers



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 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, and 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).

Schema with Identifiers



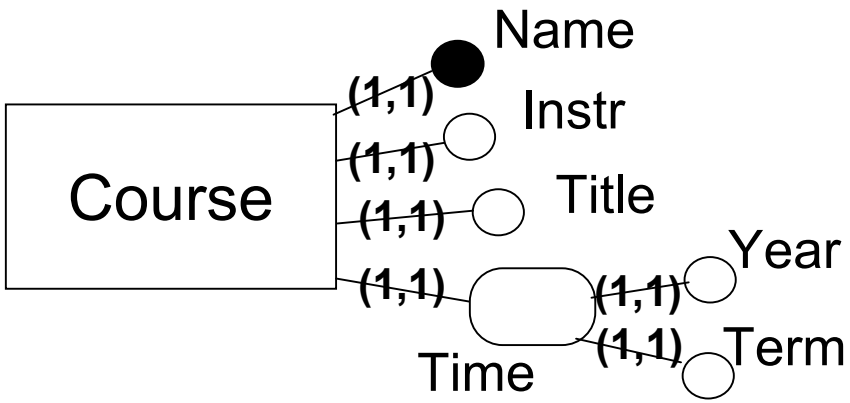
Modeling an Application with Identifiers

Identifiers constitute a powerful mechanism for modeling an application. Assume we want a database storing information about lecture meetings.

- Suppose first that we use the identifier `coursename,day,hour` for the `Meeting` entity. This says, that there can only be one meeting at any one time for a given course name, day, hour; in other words, we can't have two sections of the same course meeting at the same day+hour.
- Suppose now we use only `coursename` as identifier for `Meeting`. This says that there can only be one meeting per given course name (unreasonable!)
- If we use `courseinstructor,room` as identifier for `Meeting`, we are stating that there can only be one meeting for a given instructor+room combination, so an instructor must have all her meetings in different rooms!
- Finally, if the attribute `courseinstructor` by itself forms an identifier for `Meeting`, then the diagram we have built is stating that each instructor participates in at most one meeting, again this is unreasonable.

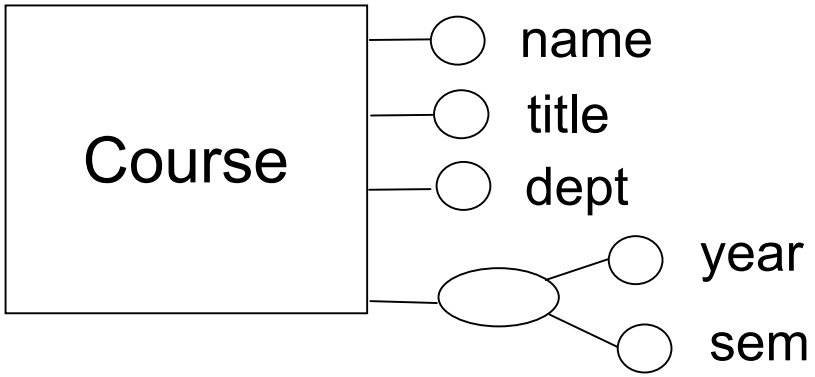
A Course Database

→ 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.



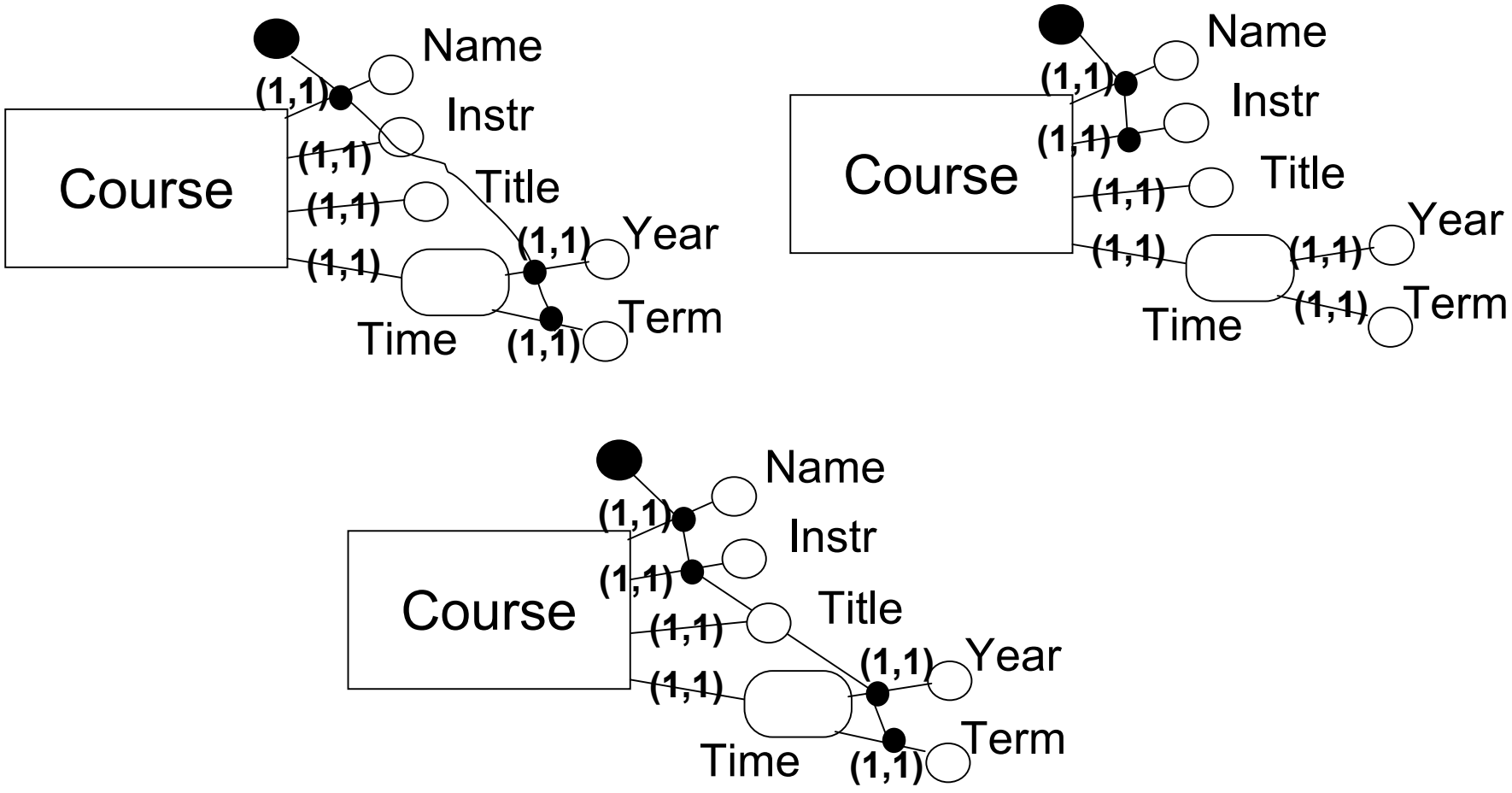
Does this make sense?

Consider ...

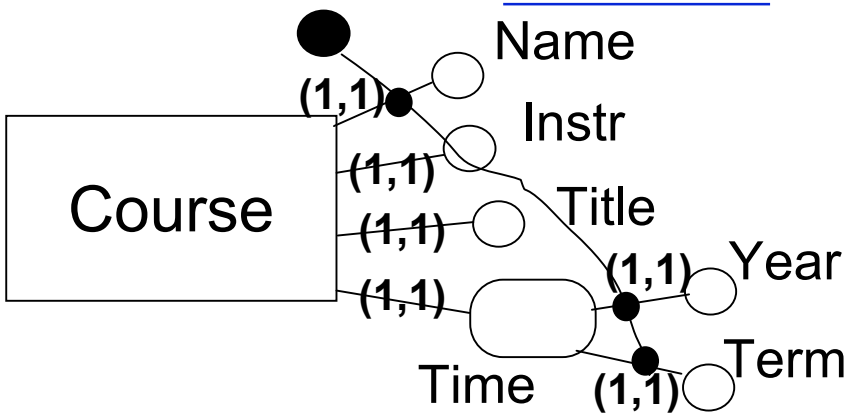


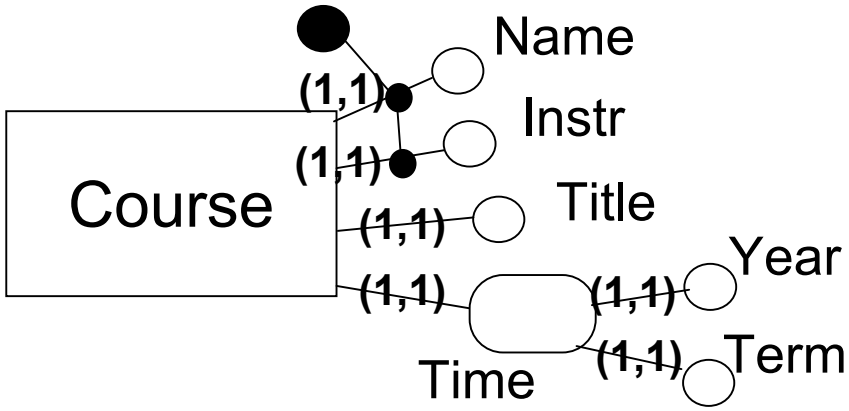
- Assume we want to keep track of course offerings over the years.
- What does each of the following key attributes say about the application?
  - ✓ name;
  - ✓ name, sem, year;
  - ✓ sem, year;
  - ✓ name, dept;
  - ✓ dept, year.

How About...

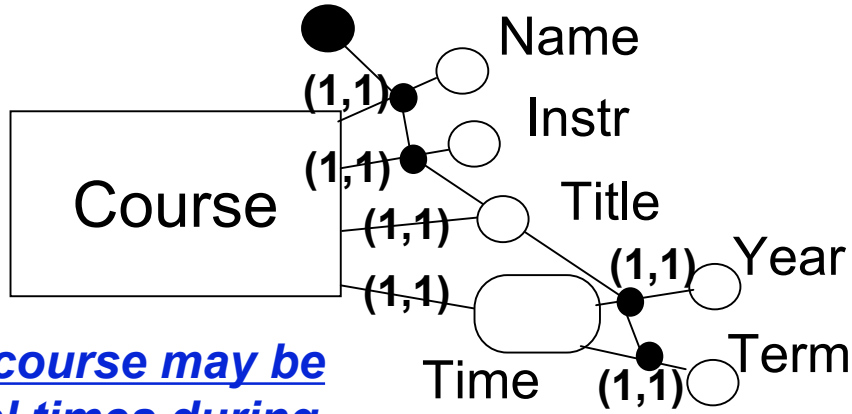


No, because a course can be given more than once during a term, with different instructors!



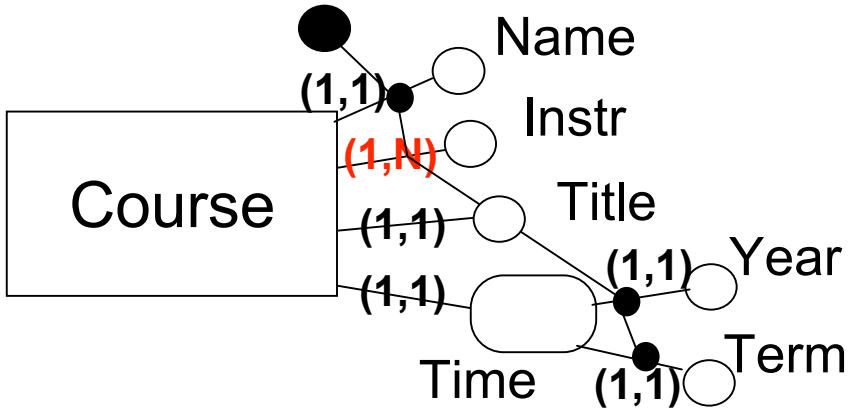


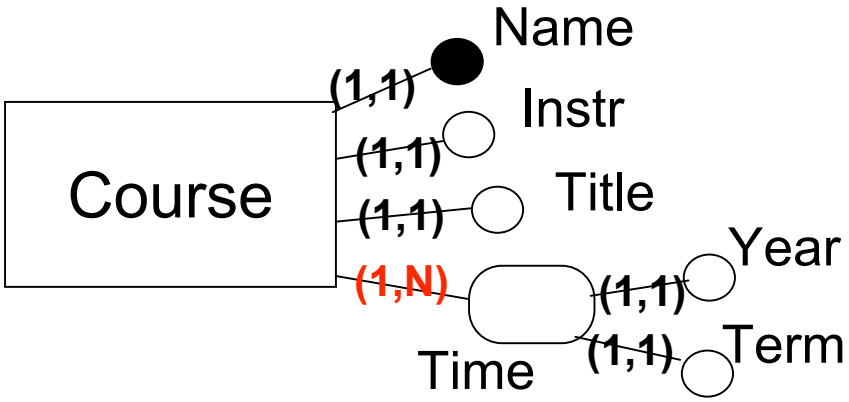
No, because an instructo may teach a course more than once!

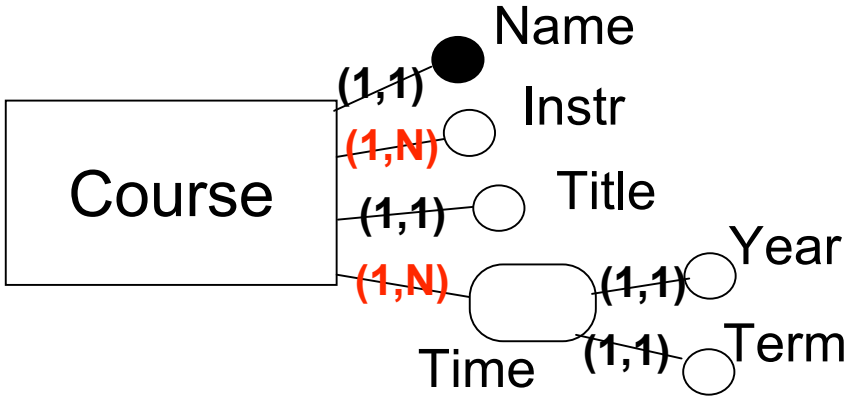


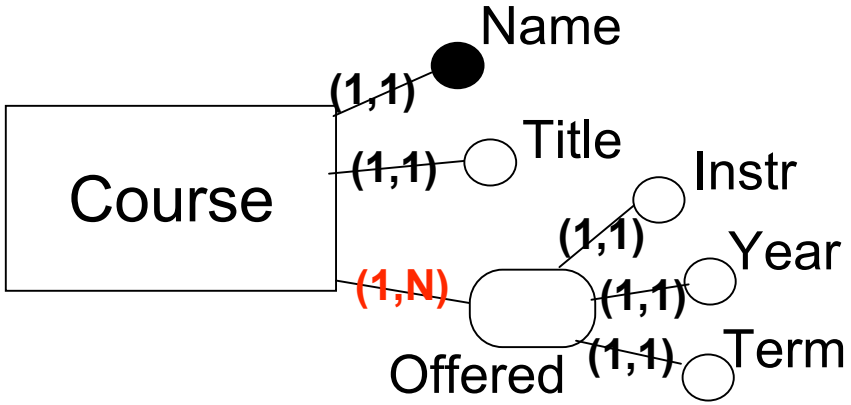
This is OK, a course may be taught several times during a term, with different instructors

...a Few More...


Course
Name
Instr
Title
Year
Term
Time
(1,1)
(1,N)
(1,1)
(1,1)
(1,1)
(1,1)

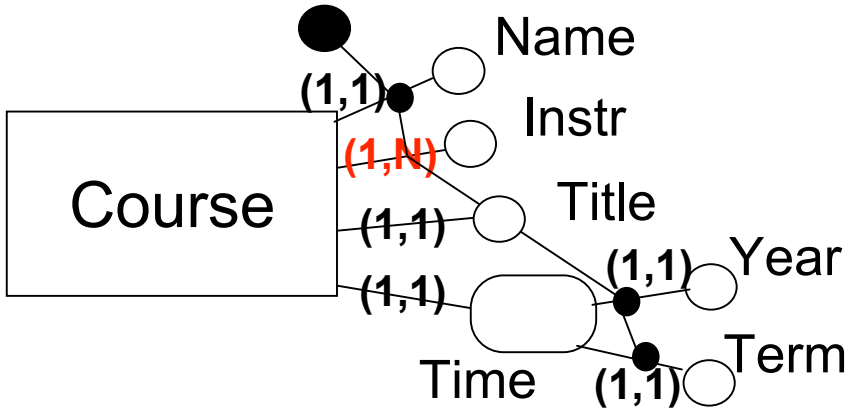

Course
Name
Instr
Title
Year
Term
Time
(1,1)
(1,1)
(1,1)
(1,N)
(1,1)
(1,1)


Course
Name
Instr
Title
Year
Term
Time
(1,1)
(1,N)
(1,1)
(1,N)
(1,1)
(1,1)

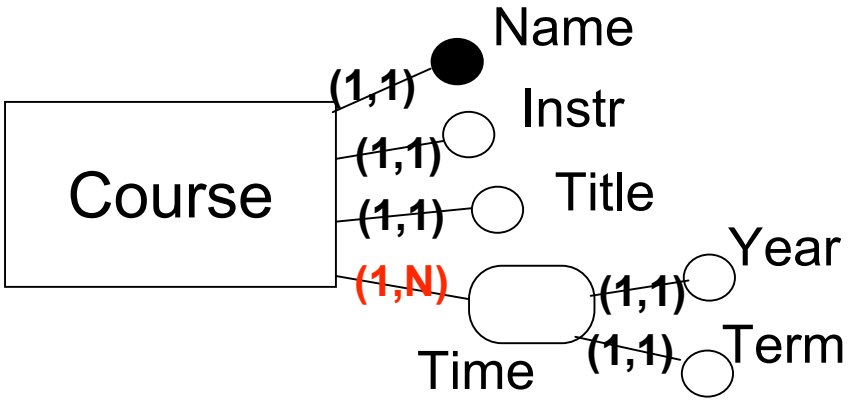

Course
Name
Title
Instr
Year
Term
Offered
(1,1)
(1,1)
(1,N)
(1,1)
(1,1)
(1,1)

. . . a Few More . . .

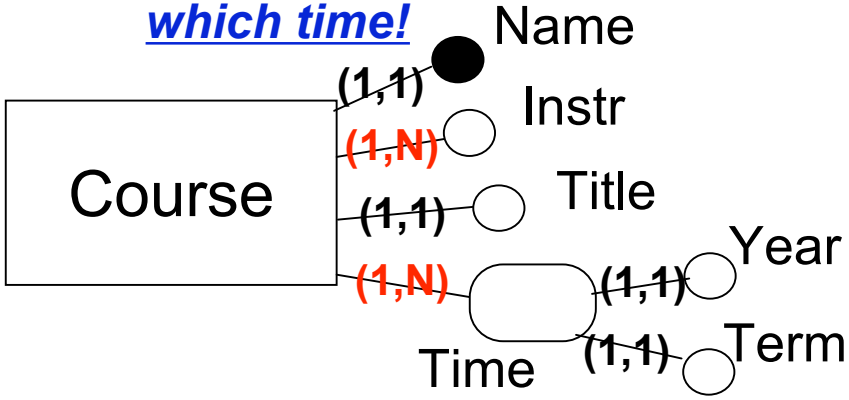
OK, we can have a course taught by several instructors during a term!



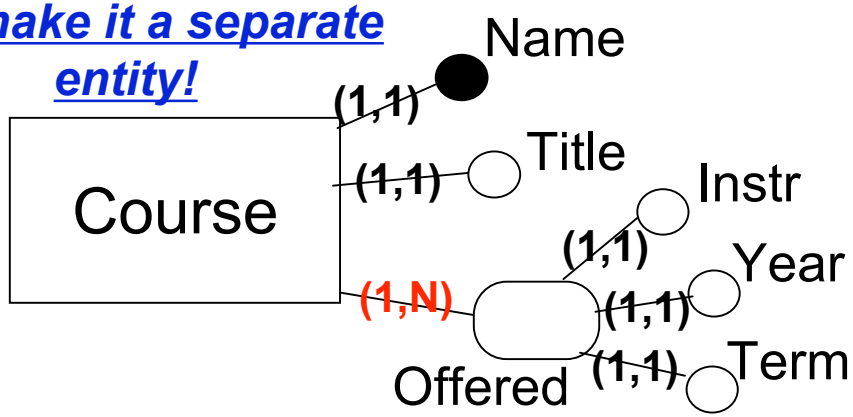
No,this says that a co will always be taught the same instructo



No, you don't know which instructor corresponds to which time!

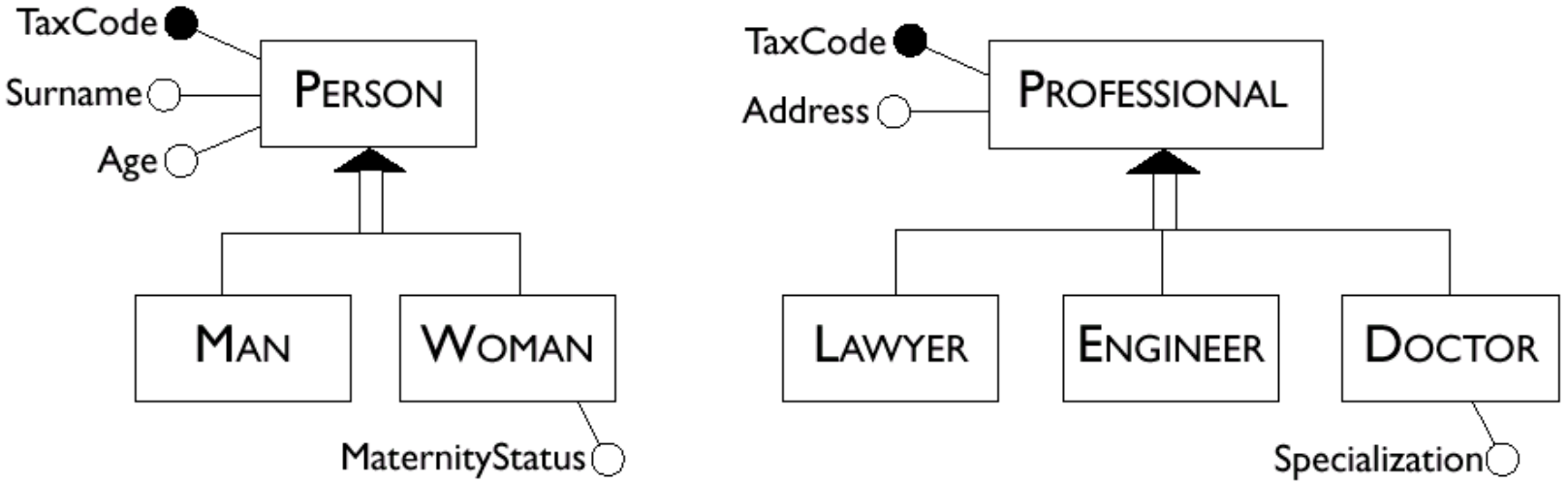


OK, but it may make more sense to split off Offered ***and make it a separate entity!***



Generalizations

- These represent logical links between an entity E, known as *parent* entity, and one or more entities $E_1,\ldots,E_n$ called *child* entities, of which E is more general, in the sense that they are a particular case.
- In this situation we say that E is a *generalization* of $E_1,\ldots,E_n$ and that the entities $E_1,\ldots,E_n$ are *specializations* of E.



Properties of Generalization

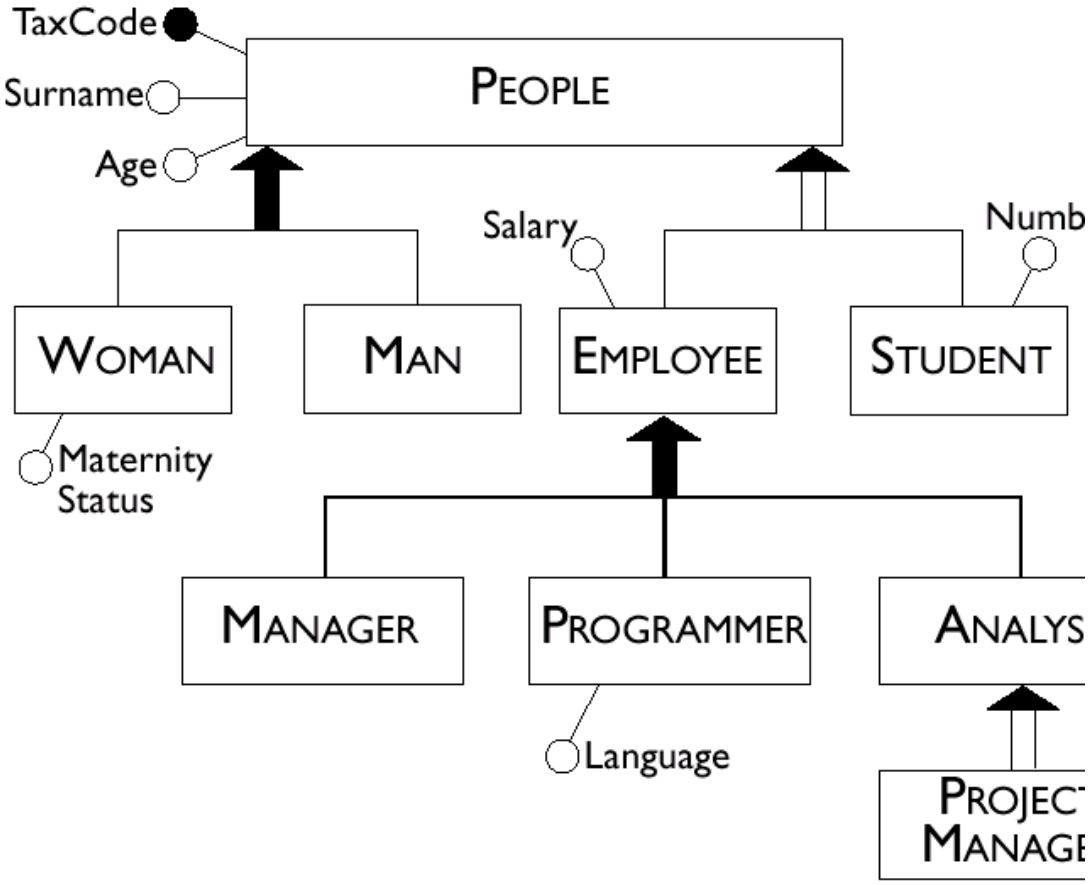
- Every instance of a child entity is also an instance of the parent entity.
- Every property of the parent entity (attribute, identifier, relationship or other generalization) is also a property of a child entity. This property of generalizations is known as *inheritance*.

Types of Generalizations

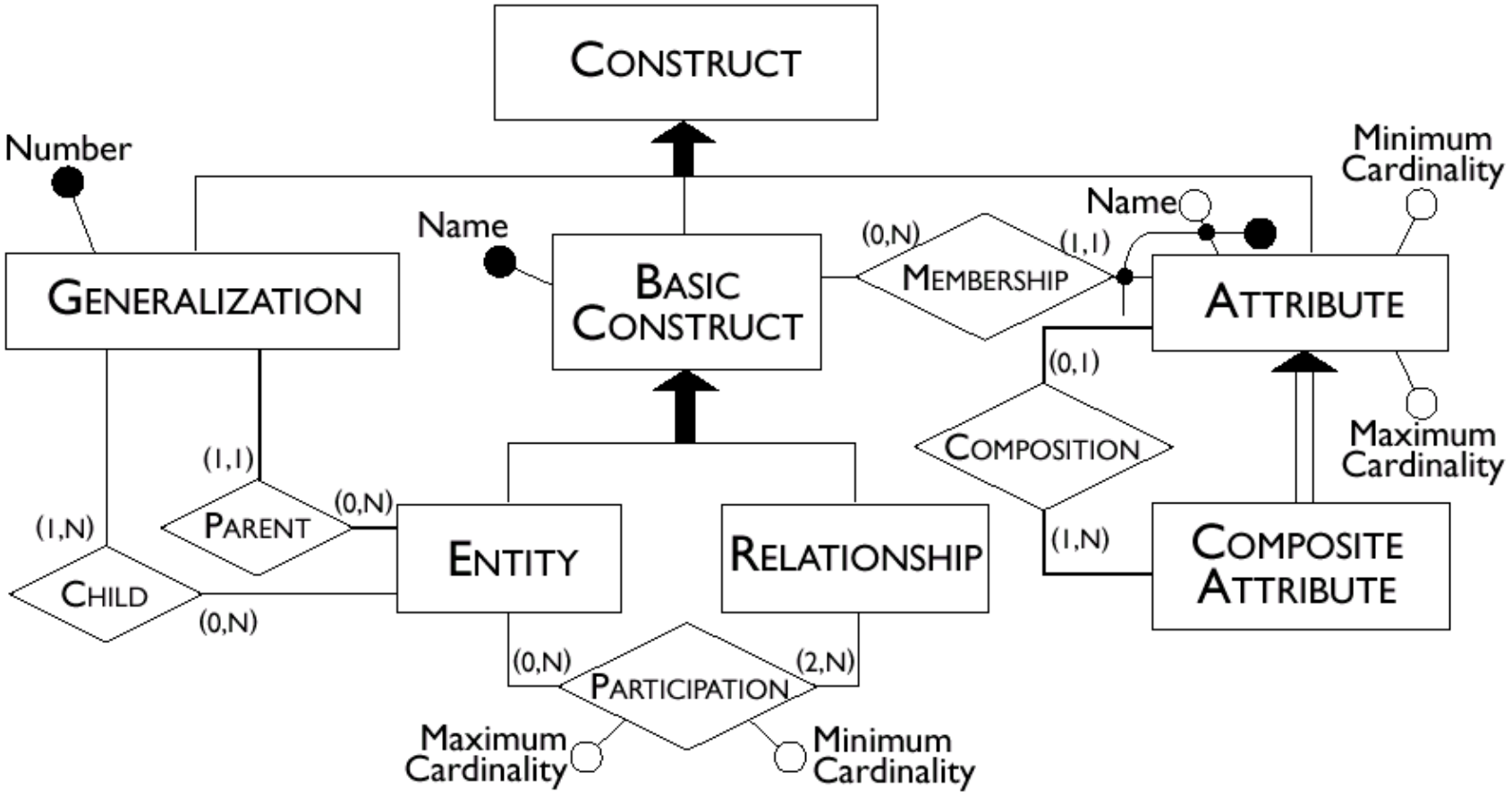
- A generalization is *total* if every instance of the parent entity is also an instance of one of its children, otherwise it is *partial*.
- A generalization is *exclusive* if every instance of the parent entity is at most an instance of one of the children, otherwise it is *overlapping*.
- The generalization `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 bicycle.
- The generalization `Person` of `Student` and `Employee` is partial and overlapping, because there are students who are also employed.

Generalization Hierarchies

- Total generalization (i.e., every instance of the superclass is an instance of some subclass) is represented by a solid arrow.
- One arrow with multiple subclasses (e.g., arrow from Woman/Man to People) means that subclasses are mutually exclusive.
- In most applications, modeling the domain involves a generalization *hierarchy* that includes several levels



The EER Model, as an EER Diagram



Example

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 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for 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 self-employed, we need to know her area of expertise, and, if appropriate, 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 freelance.

Example, Annotated

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 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for 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 self-employed, we need to know her area of expertise, and, if appropriate, 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 freelance.

More Annotations

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 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for 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 self-employed, we need to know her area of expertise, and, if appropriate, 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 freelance.

Glossary Example

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

More Annotations

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 5,000), identified by a code, we want to store her social security number, surname, age, sex, place of birth, employer's name, address and telephone number, previous employers (and periods employed), the courses attended (there are about 200 courses) and the final assessment for 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 self-employed, we need to know her area of expertise, and, if appropriate, 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 freelance.

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 maintain data for the trainees and the instructors.
Phrases relating to trainees
For each trainee (about 5000), identified by a code, the database 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 trainees
For each employer of a trainee the database will store name, address and telephone number.

Structuring of Requirements (II)

Phrases relating to 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 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.

Operational Requirements

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

Conceptual Design with the EER Model

Design choices:

- → Should a concept be modeled as an entity or an attribute?
- → Should a concept be modeled as an entity or a relationship?
- → Identifying relationships: Binary or ternary? Aggregation?

Constraints on the ER Model:

- → A lot of data semantics can (and should) be captured.
- → But some constraints cannot be captured by ER diagrams.

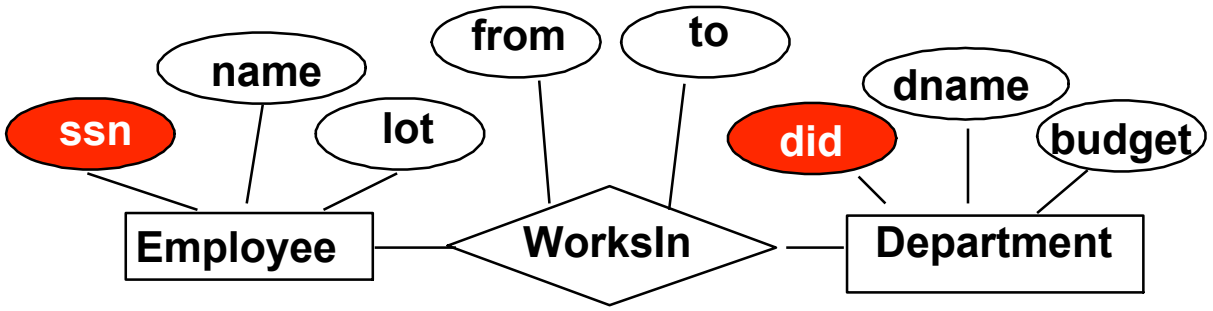
Some Rules of Thumb

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

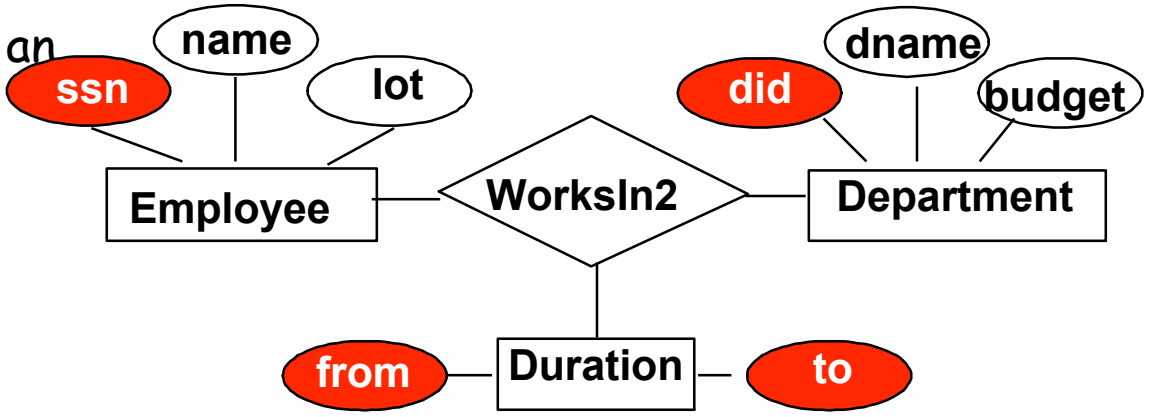
Examples

- Consider address of a trainee. Is it an entity, relationship, or attribute?
- Consider address for a telephone company database, which has to keep track of how many and what type of phones are available in any one household, who lives there (there may be several phone bills going to the same address) etc. for this case, address is probably best treated as an entity.
- Or, consider an employee database, where for each employee you maintain personal information, such as her address. Here address is best represented as an attribute.
- Or, consider a police database where we want to keep track of a person's whereabouts, including her address (i.e., address from Date1 to Date2, address from Date2 to Date3, etc.) Here, address is treated best as a relationship.

Entity vs. Attribute

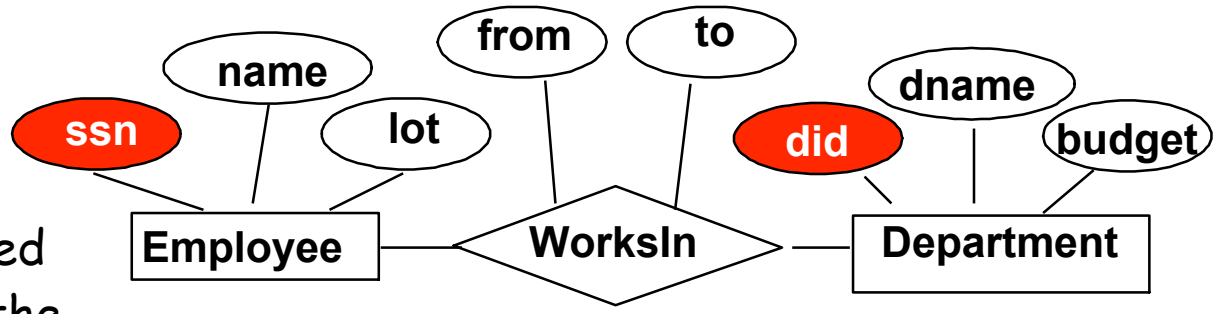


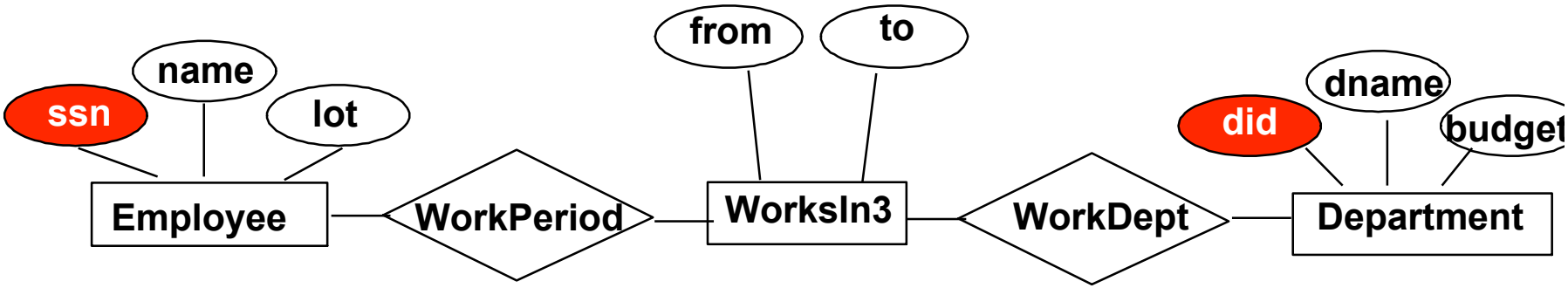
WorksIn does not allow an employee to work in a department for two or more periods. So, it is best to turn the from/to attributes into an entity.



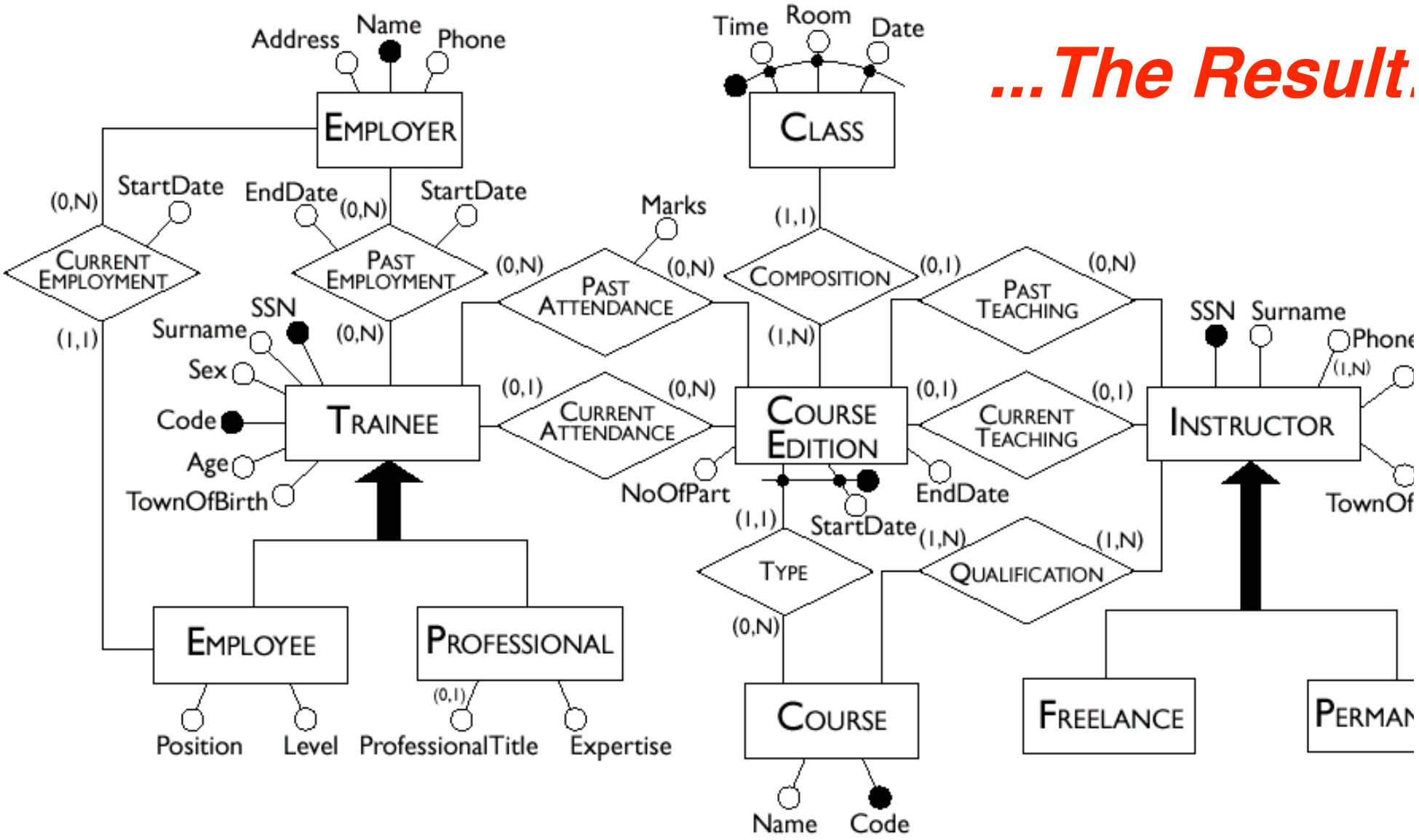
Entity vs. Relationship

WorksIn can also be turned into an entity to avoid the problem mentioned in the previous slide





...The Result


Address
Name
Phone
EMPLOYER
Time
Room
Date
CLASS
(0,N)
StartDate
CURRENT EMPLOYMENT
EndDate
(0,N)
StartDate
PAST EMPLOYMENT
Marks
(0,N)
PAST ATTENDANCE
(1,1)
COMPOSITION
(0,1)
(0,N)
PAST TEACHING
(1,1)
SSN
Surname
(0,N)
Sex
Code
Age
TownOfBirth
TRAINEE
(0,1)
(0,N)
CURRENT ATTENDANCE
(1,N)
COURSE EDITION
NoOfPart
(1,1)
StartDate
EndDate
(0,1)
(0,1)
CURRENT TEACHING
SSN
Surname
Phone
(1,N)
INSTRUCTOR
TownOf
(1,1)
TYPE
(1,N)
(1,N)
QUALIFICATION
(0,N)
EMPLOYEE
PROFESSIONAL
(0,1)
Position
Level
ProfessionalTitle
Expertise
COURSE
Name
Code
FREELANCE
PERMAN

Documentation of an EER Schema

- 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 *business object*),
  - ✓ an *integrity constraint* on the data of the application,
  - ✓ a *derivation rule*, whereby information can be derived from othe 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. ....

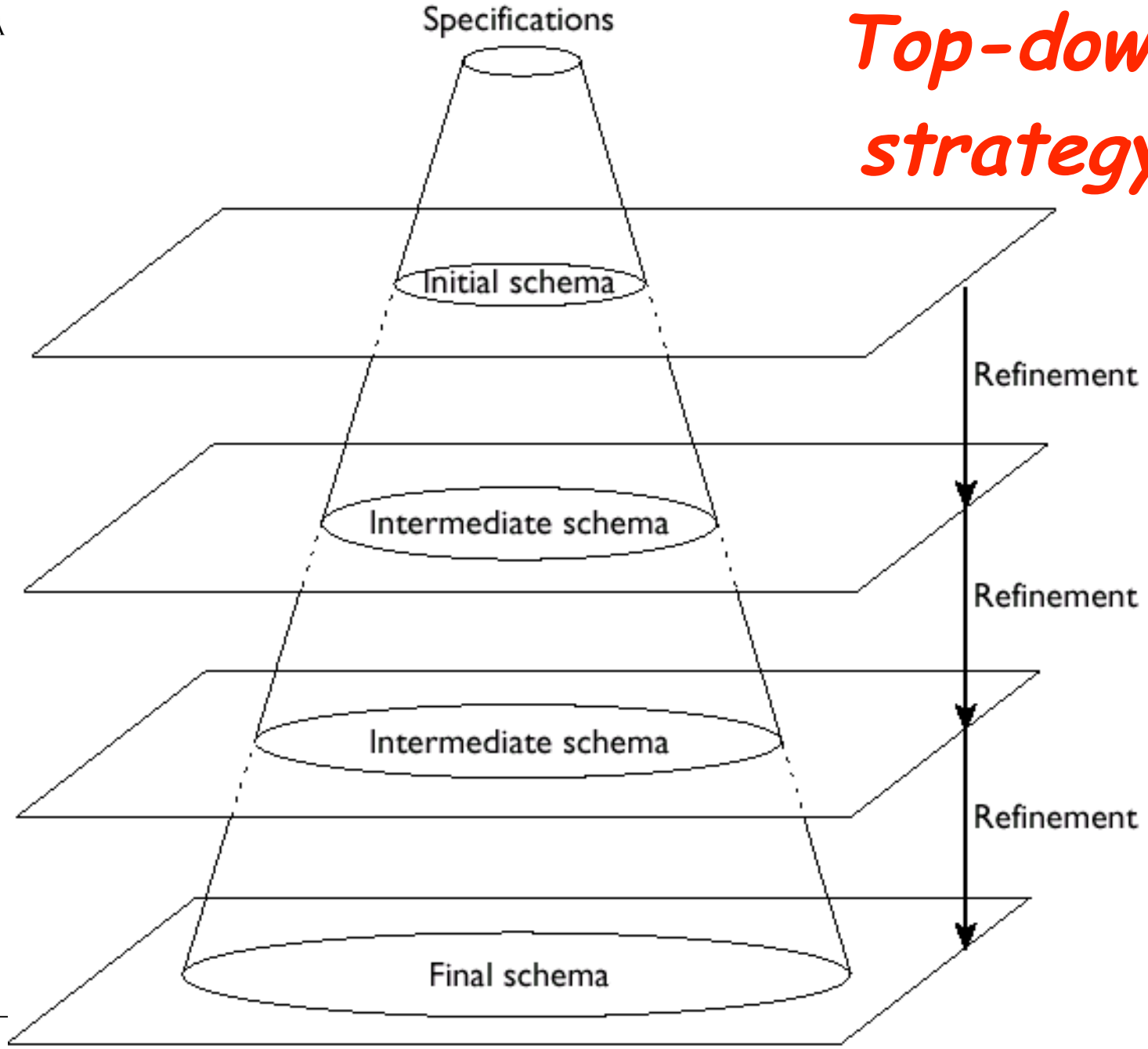
Conceptual Modeling Strategies

- The design of a conceptual schema for a given set of requirements is an engineering process and, as such, can use design strategies from other disciplines:
  - ✓ Top-down
  - ✓ Bottom-up
  - ✓ Middle-out
  - ✓ Mixed

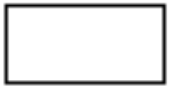
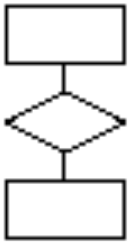
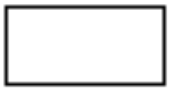
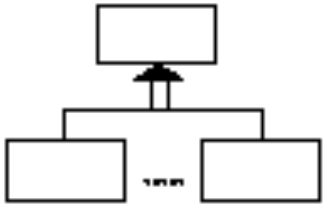
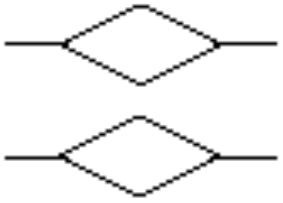
Top-Down Strategy

- 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 transformations that add detail.
- Moving from one level to the next involves modifications of the schema using a basic set of top-down transformations.

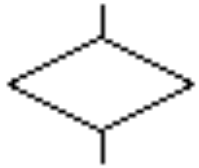
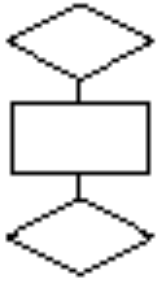
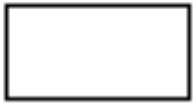
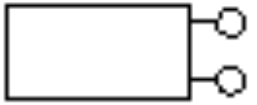
Top-down strategy



Top-Down Transformations I

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		

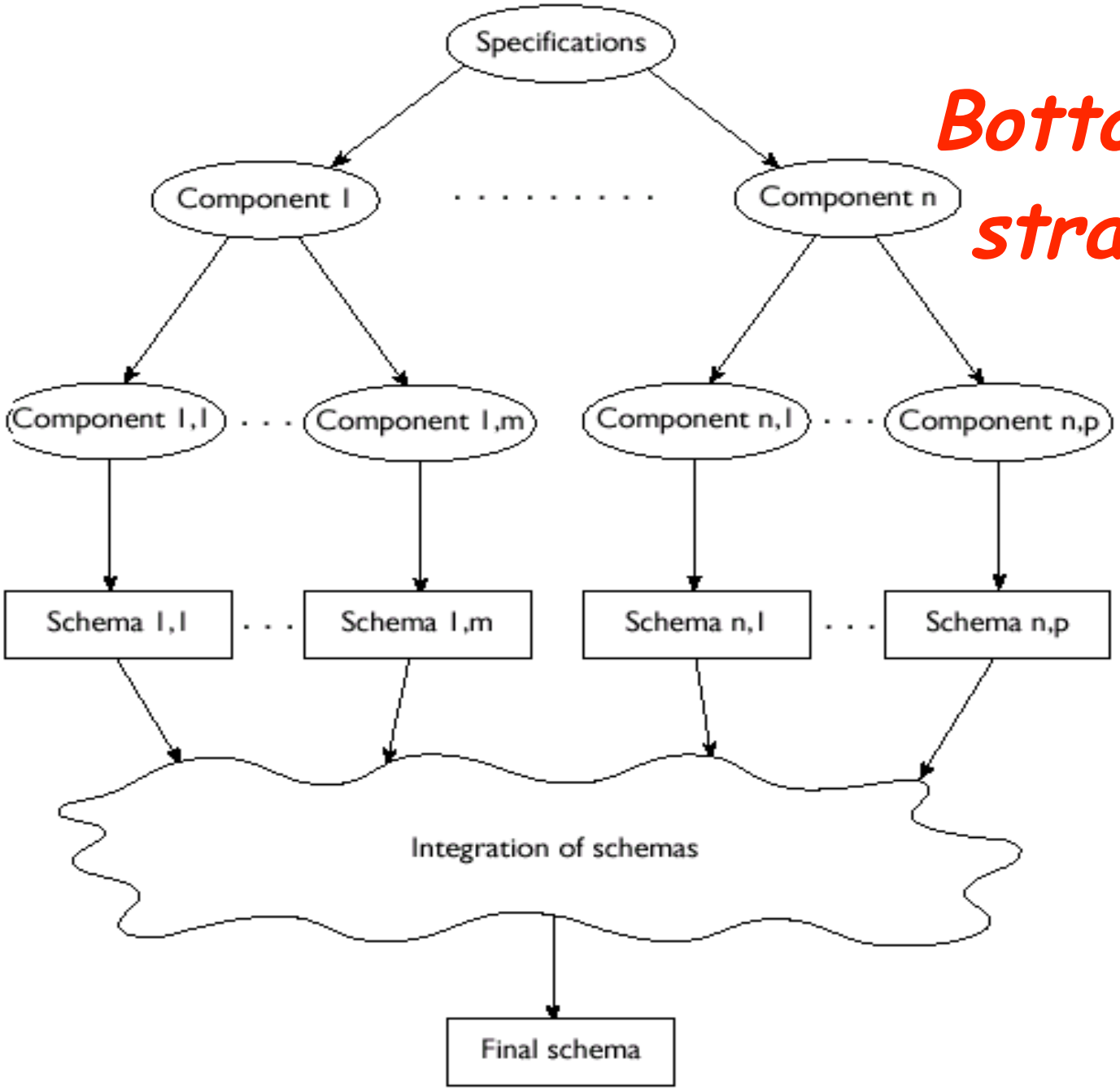
Top-Down Transformations II

T_4 From one relationship to an entity with relationships		
T_5 Adding attributes to an entity		
T_6 Adding attributes to a relationship		

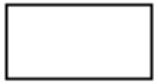
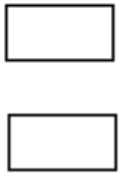
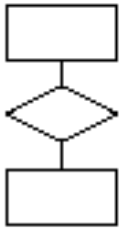
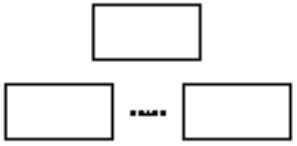
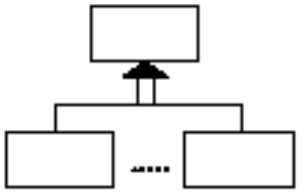
Bottom-Up Strategy

- The initial specification is decomposed iteratively into finer-grain specifications, until a specification becomes atomic, I.e., consists of simple elements.
- Each atomic specification is then represented by a simple conceptual schema.
- The schemas obtained from atomic specifications are then integrated into a final conceptual schema.
- The final schema is obtained by means of a set of *bottom-up transformations*.

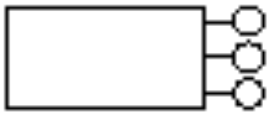
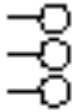
Bottom-up strategy



Bottom-Up Transformations I

Transformation	Initial concept	Result
T_1 Generation of an entity		
T_2 Generation of a relationship		
T_3 Generation of a generalization		

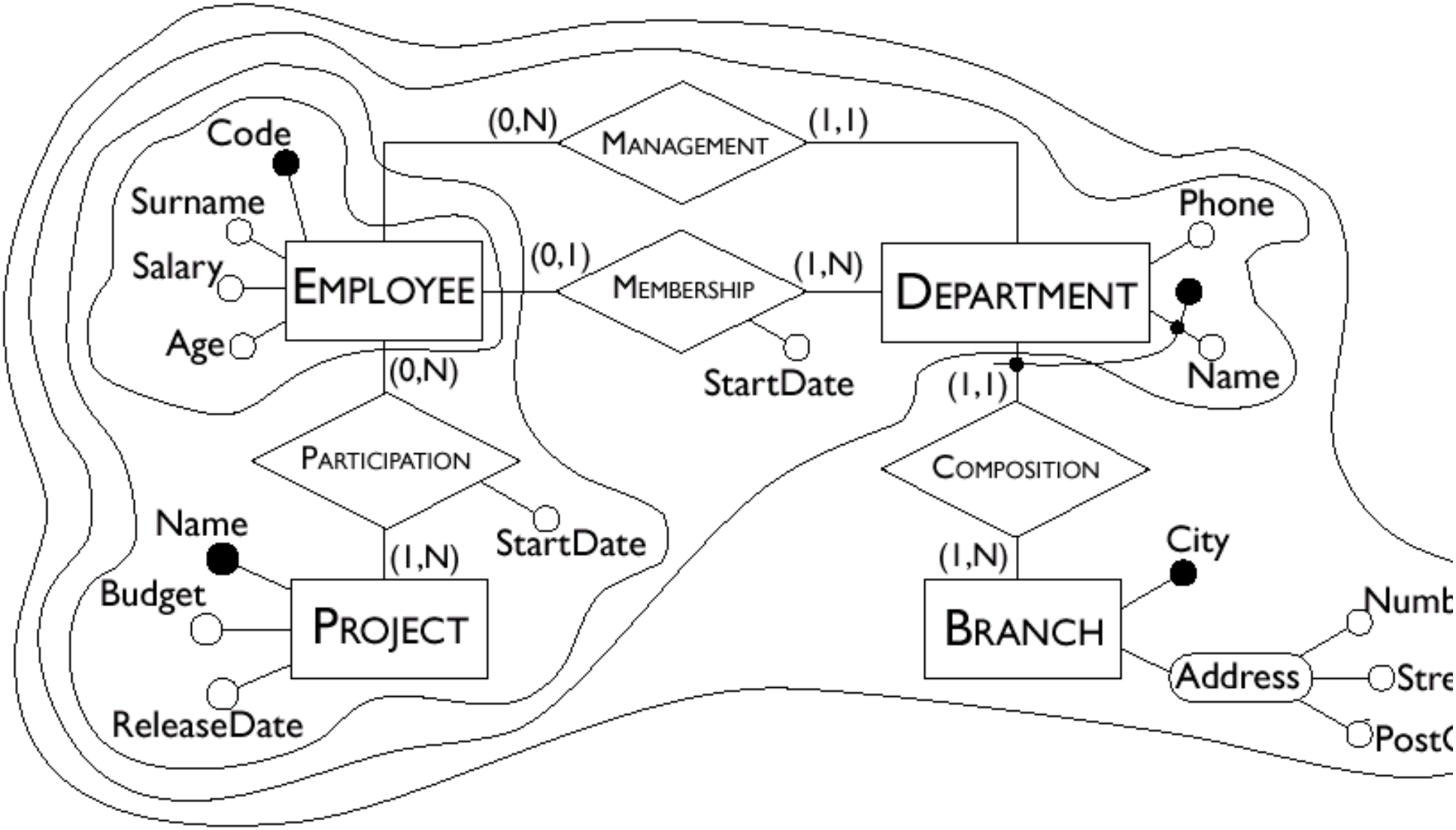
Bottom-Up Transformations II

T_4 Aggregation of attributes on an entity		
T_5 Aggregation of attributes on a relationship		

Middle-Out Strategy

- This strategy can be regarded as a special case of the 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.

Middle-Out Strategy Example



Mixed Strategy

- Here the designer decomposes the requirements into a number of components, as in the bottom-up strategy, but not to as fine-grain a level.
- Designer also 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 a top-down strategy) or extending them with concepts that are not yet represented (following a bottom-up strategy).

Qualities for a Conceptual Schema

- *Correctness*. Conceptual schema uses correctly the constructs made available by the conceptual model. As with programming languages, the errors can be *syntactic* or *semantic*.
- *Completeness*. Conceptual schema represents all data requirements and allows for the execution of all the operations included in the operational requirements.
- *Readability*. Conceptual schema represents the requirements in a way that is natural and easy to understand. Therefore, the schema must be self-explanatory; for example, by choosing suitable names for concepts.
- *Minimality*. Schema avoids *redundancies*, e.g., data that can be derived from other data.

A Comprehensive Method for Conceptual Design (I)

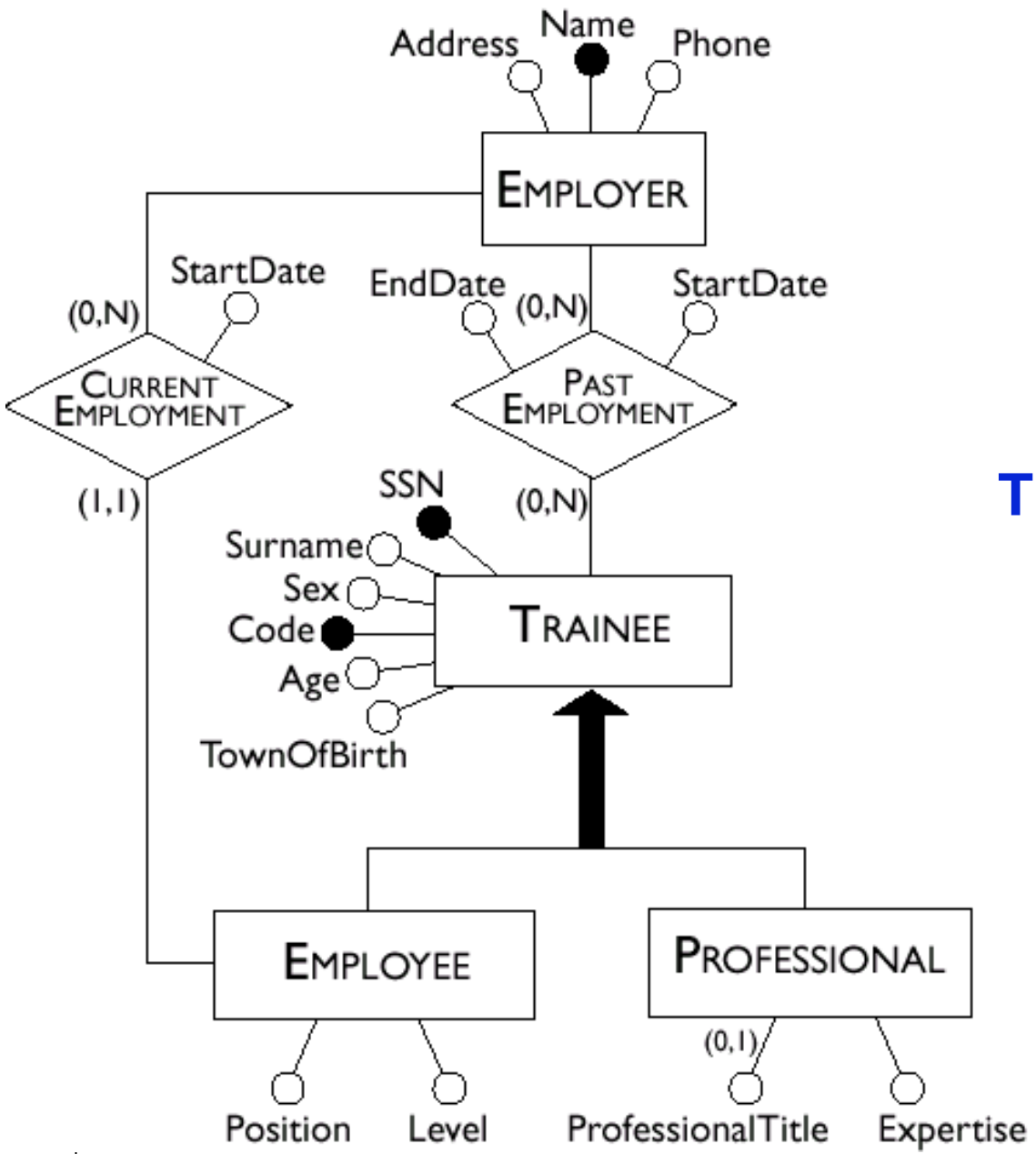
1. Analysis of requirements
   (a) Construct a glossary of terms.
   (b) Analyze the requirements and eliminate all ambiguities.
   (c) Arrange the requirements in groups.
2. Basic step
   (a) Identify the most relevant concepts and represent them in a skeleton schema.
3. Decomposition step (to be used 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 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, completeness, minimality and readability of the schema.

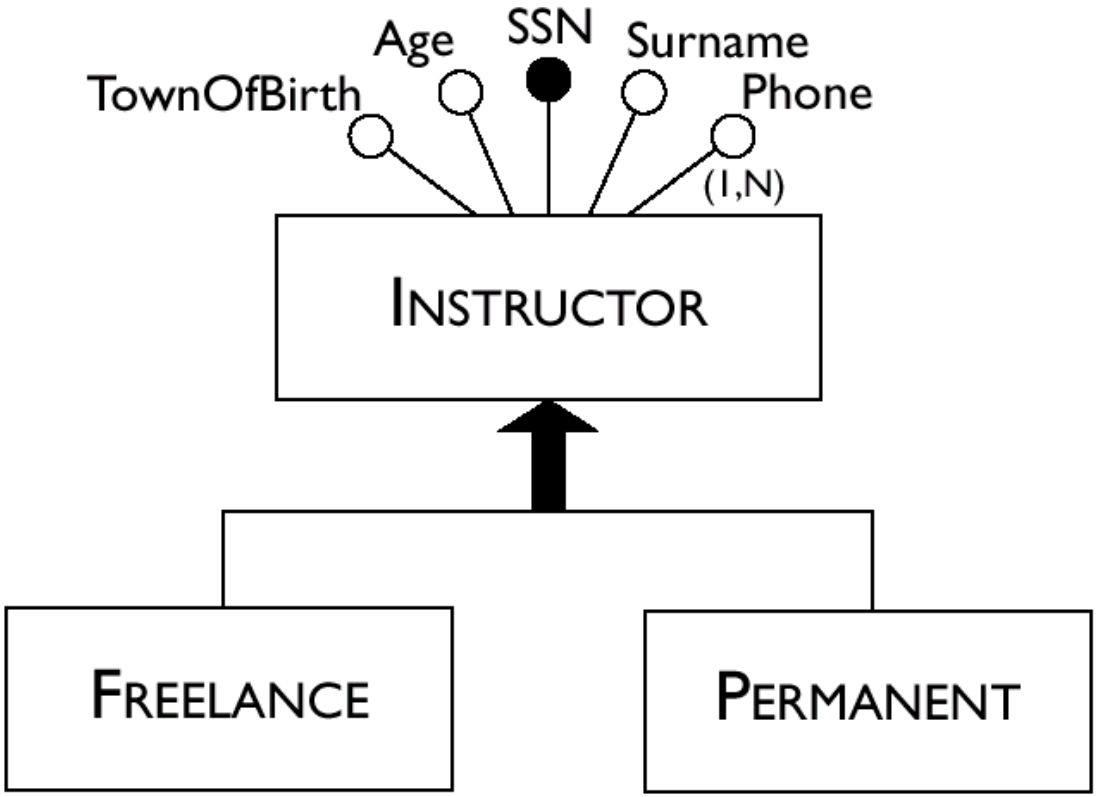
The Training Company Example

Skeleton schema for the training company

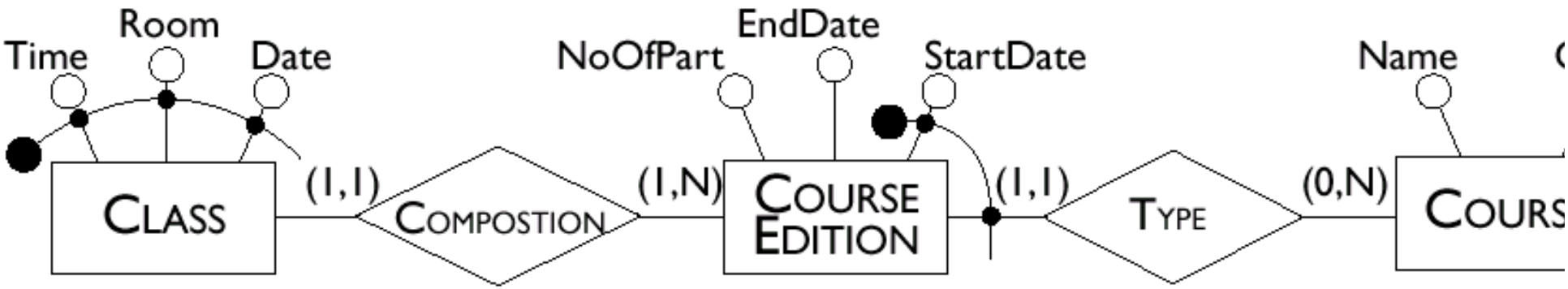


The refinement of a portion of the skeleton schema

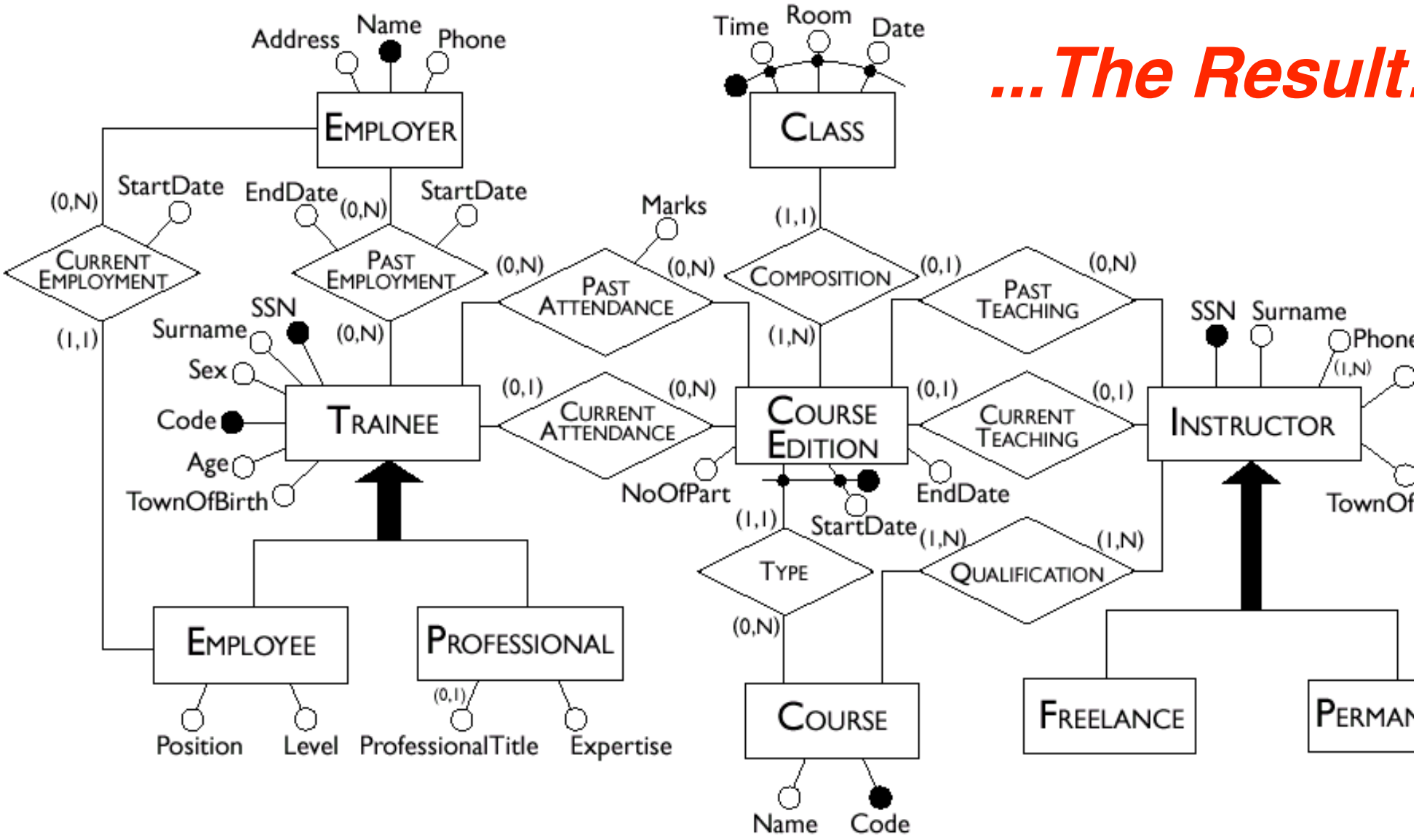
The refinement of another portion of the skeleton schema



The refinement of another portion of the skeleton schema

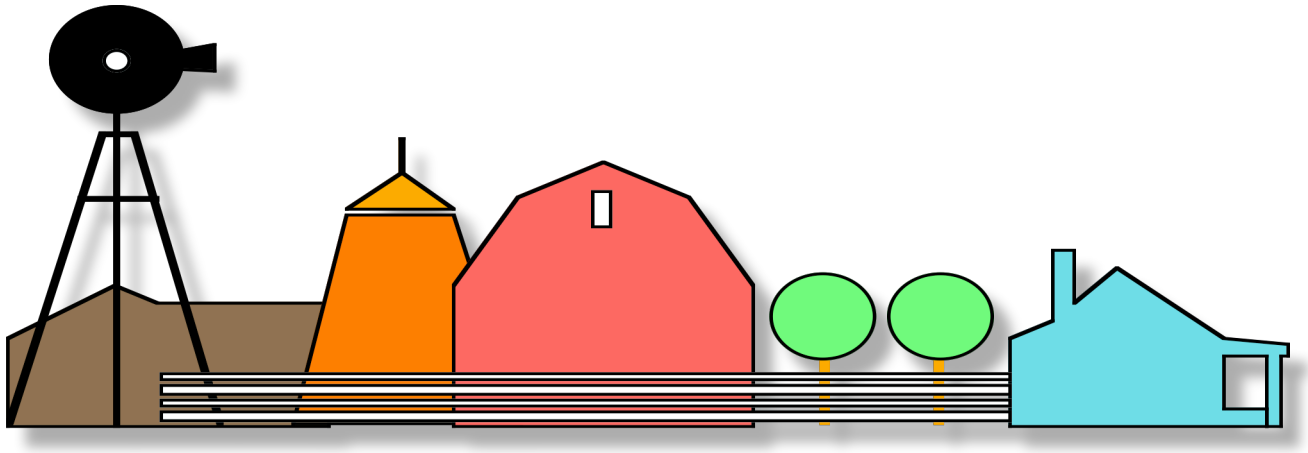


...The Result

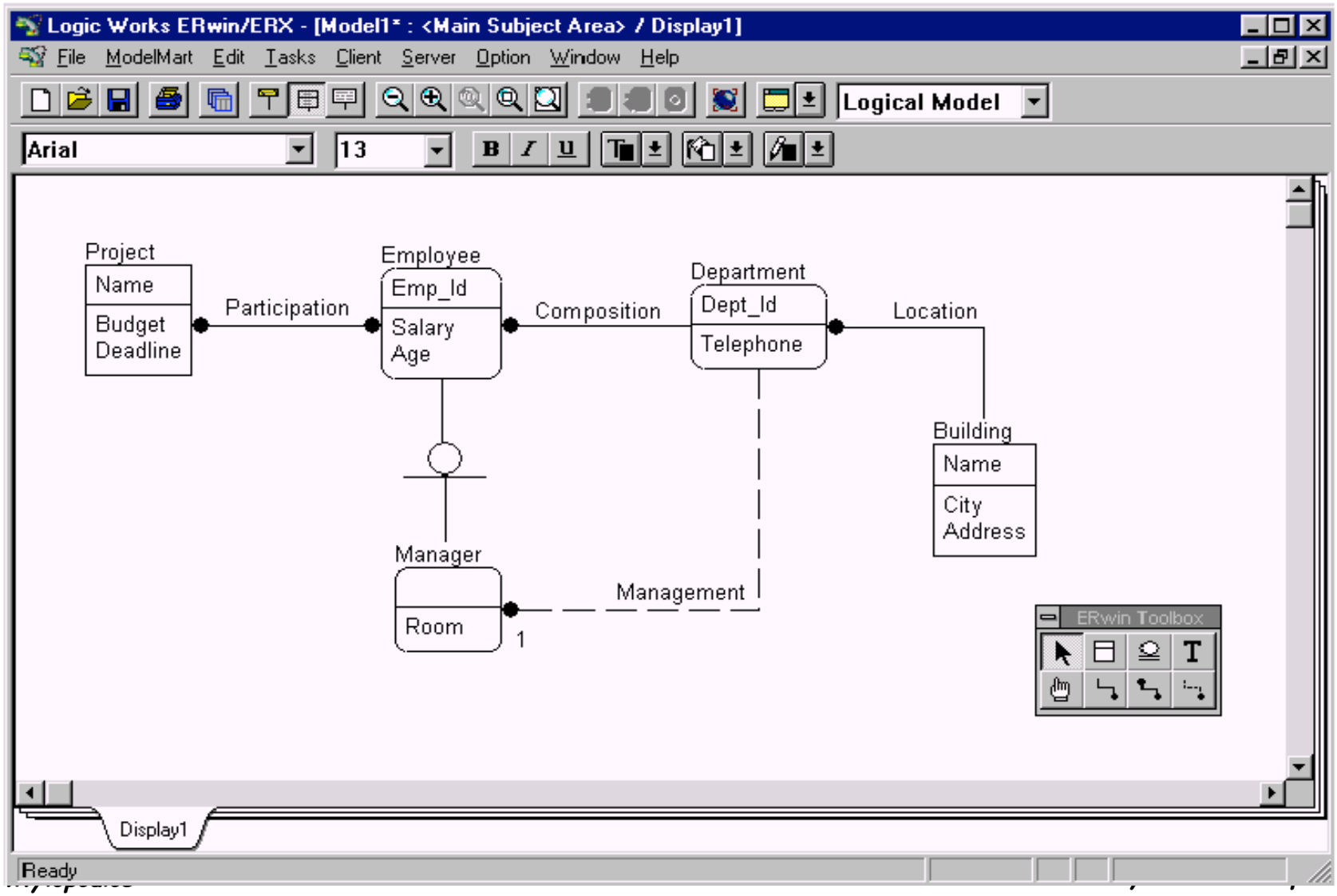

Address
Name
Phone
EMPLOYER
Time
Room
Date
CLASS
(0,N)
StartDate
CURRENT EMPLOYMENT
EndDate
(0,N)
StartDate
PAST EMPLOYMENT
Marks
(0,N)
PAST ATTENDANCE
(1,1)
COMPOSITION
(0,1)
(0,N)
PAST TEACHING
(1,1)
SSN
Surname
(0,N)
Sex
Code
Age
TownOfBirth
TRAINEE
(0,1)
(0,N)
CURRENT ATTENDANCE
(1,N)
COURSE EDITION
NoOfPart
(1,1)
StartDate
EndDate
(0,1)
(0,1)
CURRENT TEACHING
SSN
Surname
Phone
(1,N)
INSTRUCTOR
TownOf
(1,1)
TYPE
(1,N)
(1,N)
QUALIFICATION
(0,N)
EMPLOYEE
PROFESSIONAL
(0,1)
Position
Level
ProfessionalTitle
Expertise
COURSE
Name
Code
FREELANCE
PERMAN

CASE Tools for Database Design

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



Conceptual design using a CASE tool (ER-Win)



References

- [Atzeni99] Atzeni, P., Ceri, S., Paraboschi, S., and Torlone, R.,. *Database Systems*, McGraw-Hill, 1999.
- [Chen76] P. Chen, P., "The Entity-Relationship Model: Towards a Unified View of Data," *ACM Transactions on Database Systems 1(1)*, 1976.