# Week 10:
# The Entity-Relationship Model

***The Entity-Relationship Model***
***Entities, Relationships and Attributes***
***Cardinalities and Keys***
***Entity Hierarchies and Aggregation***
***Designing a Conceptual Schema, Design Choices***





## The Entity Relationship Model

- The ***Entity-Relationship*** (ER) ***model*** is a *conceptual* data model, capable of describing the meaning of the data in a database in an easy-to-understand graphical notation.
- The meaning is described in terms of a ***conceptual*** *(or,* ***ER****)* ***schema****.*
- ER schemas are comparable to class diagrams in UML.

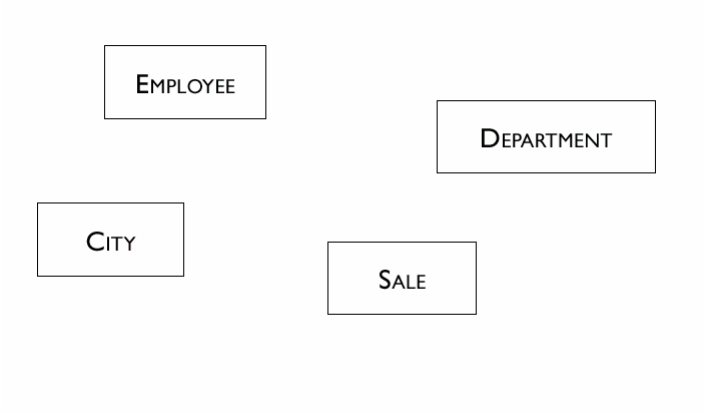
## Entities and Entities Sets

- An entity set represent classes of objects (facts, things, people,...) that have properties in common and an autonomous existence, e.g., **City, Department, Employee, Purchase** and **Sale**.
- An entity is an instance/member of an entity set, e.g., **Stockholm**, **Helsinki**, are examples of instances of the entity **City**; **Peterson** and **Johanson** are examples of instances of the **Employee** entity set.





## Examples of Entity Sets



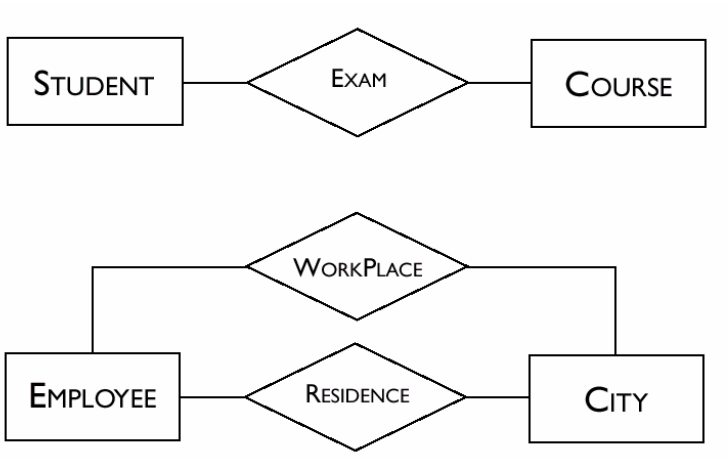
## Relationship Sets

- Represent associations between 2+ entity sets.
- ***Residence*** is a relationship set that can exist between entity sets ***City*** and ***Employee***; ***Exam*** is an example of a relationship that can exist between the entities ***Student*** and ***Course***.
- An instance of a relationship set is an n-tuple made up of entities, one for each of entity sets involved.
- The pair (**Johanssen,Stockholm**), or the pair (**Peterson,Oslo**), are examples of relationships that are instances of **Residence**.
- ***Note: The collection of instances of a relationship is by definition a set, not a bag; i.e., no duplicates!***

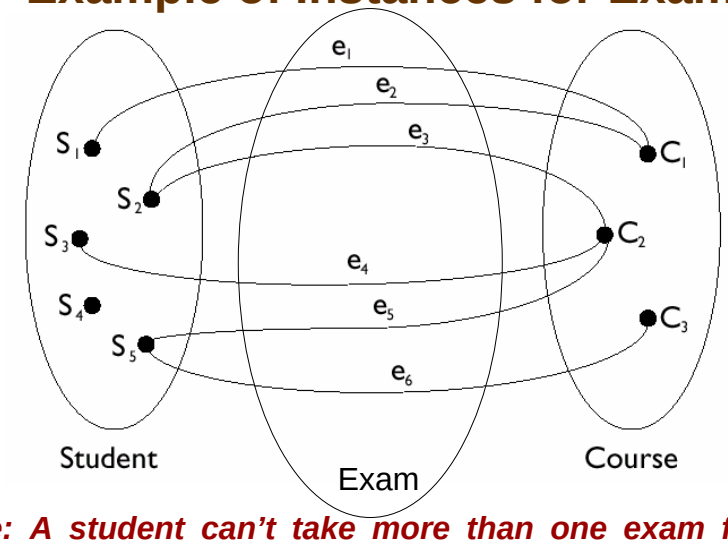




## Examples of Relationship Sets



# Example of Instances for Exam

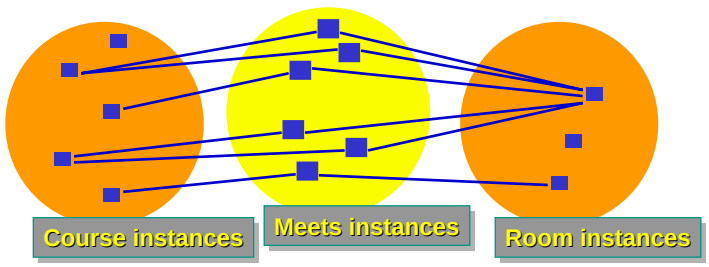



***Note: A student can't take more than one exam for a particular course!***



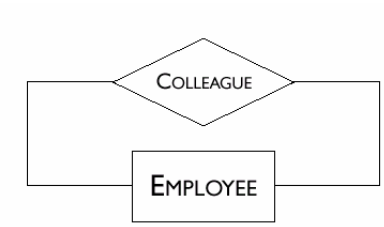


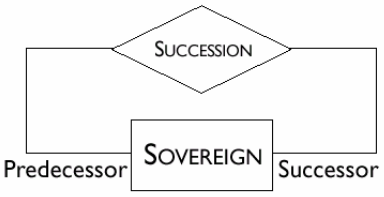
# *What Does A Schema Really Mean?*

# 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.

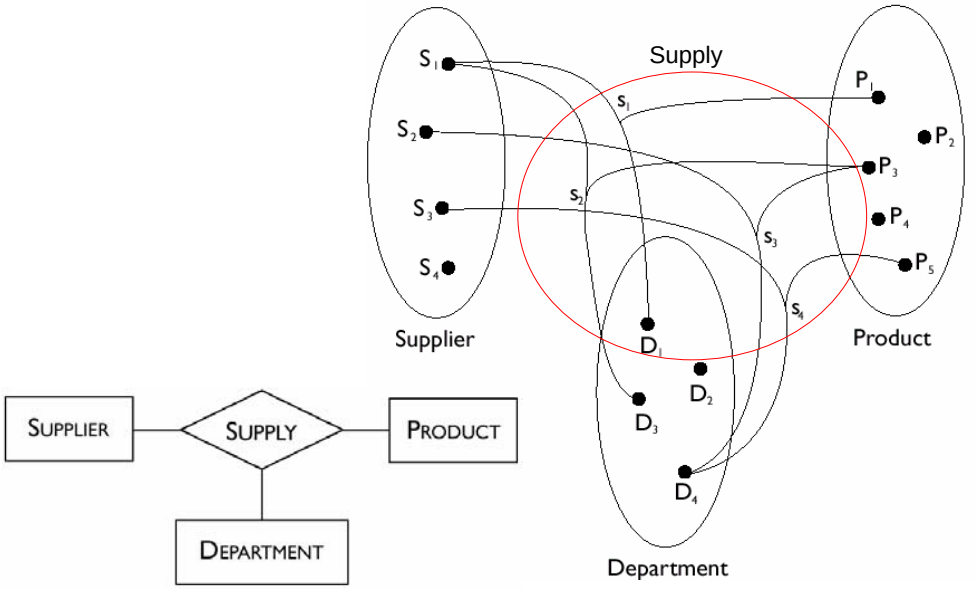







# Ternary Relationships



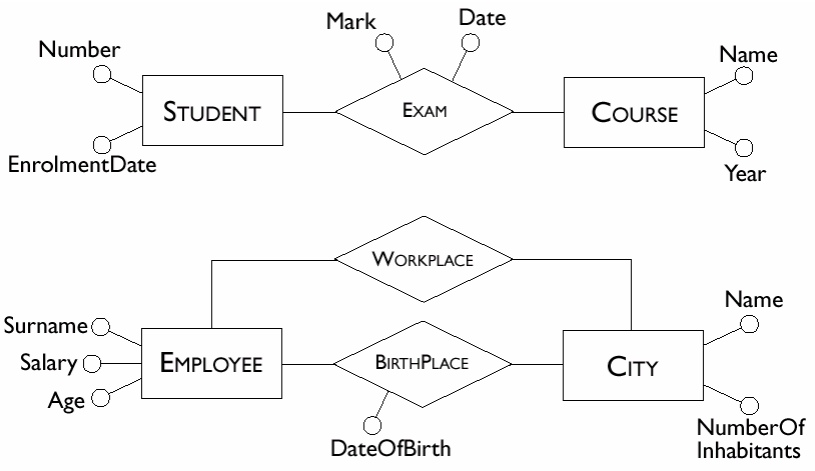
# Attributes

- Describe elementary properties of entities or relationships.
- For example, **Surname**, **Salary** and **Age** are **attributes** of **Employee**, while **Date** and **Mark** are **attributes** of relationship **Exam** between **Student** and **Course**.
- An attribute associates with each instance of an entity (or relationship) one or more values belonging to its ***domain***.
- Attributes may be single-valued, or multi-valued.



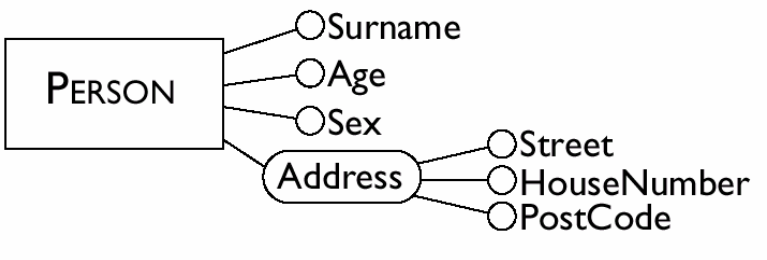


# Attribute Examples



## Composite Attributes

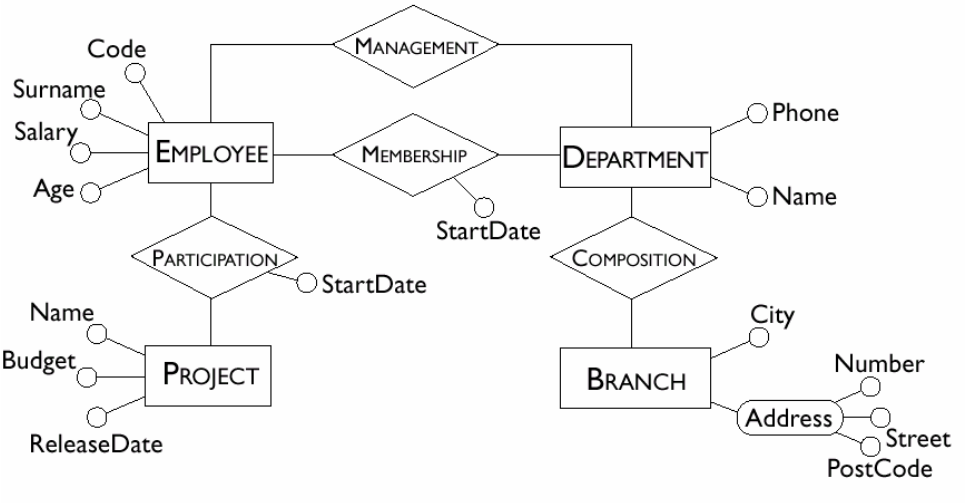
- It is sometimes convenient to group attributes of the same entity or relationship that have closely connected meanings or uses. Such groupings are called ***composite attributes***.








## Schema with Attributes



# Cardinalities

- These are specified for each entity participating in a relationship and describe the **maximum** and **minimum** number of relationship instances that an entity instance can participate in.


- In this example:
  - ✓ an employee has at minimum [ ] assignment task and no more than [ ]
  - ✓ a task has a minimum of [ ] employees assigned and no more than [ ]

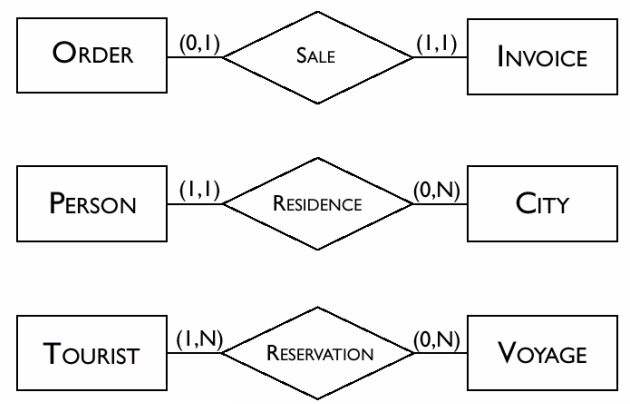




# Cardinalities (cont'd)

- 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, entity participation in a relationship is [ ]. If minimum cardinality is 1, entity participation in a relationship is [ ].
- 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 [ ] number of instances of the relationship.

# Cardinality Examples



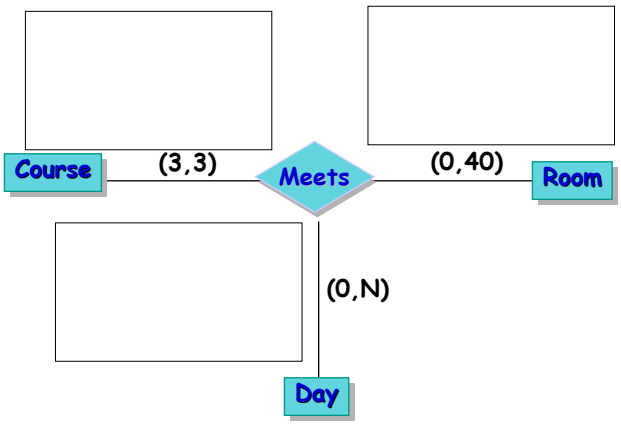





# Textbook Notation

- …for (1,1) cardinalities (*key constraints*)

# Cardinality Example

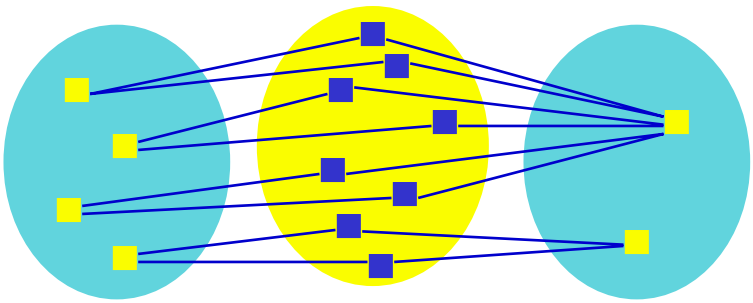




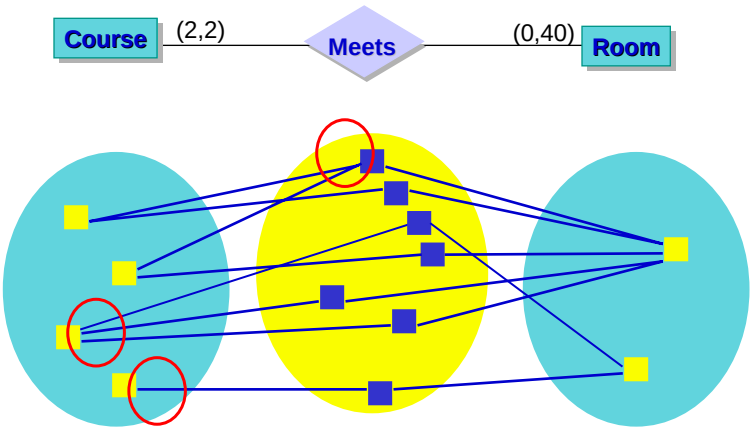



# Instantiating ER schemas

- An ER schema specifies what states are possible in the world being modeled

# Illegal Instantiations

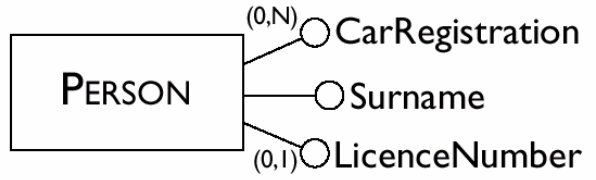






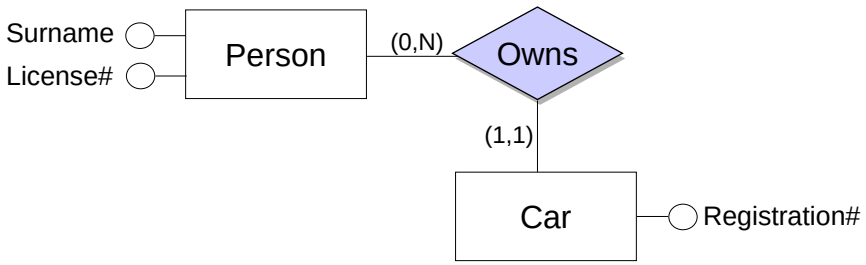

# Cardinalities of Attributes

- Describe the minimum and maximum number of values an attribute can have.
- When the cardinality of an attribute is (1,1), it can be omitted (***single-valued*** attributes)
- The value of an attribute, however, may also be null, or there may be several values (***multi-valued*** attributes)



## Cardinalities (cont'd)

- Multi-valued attributes often represent situations that can be modelled with additional entities.
- For example, the ER schema of the previous slide can be revised into:



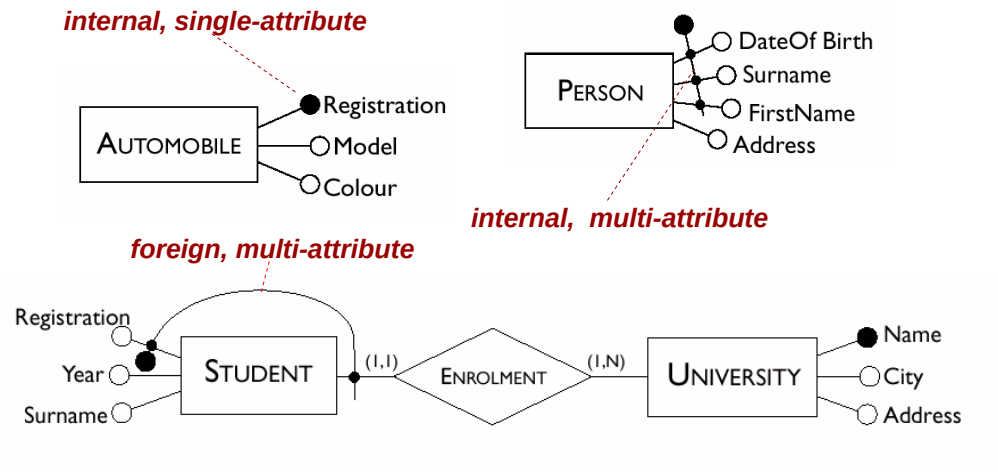





## Keys

- ***Keys*** consist of minimal sets of attributes which identify uniquely instances of an entity set.
- For example, **`socialInsurance#`** may be a key for **`Person`**. Alternatively, **`firstName`**, **`middleName`**, **`lastName`**, **`address`** may be a key.
- In most cases, a key is formed by one or more attributes of the entity itself (***internal*** keys).
- Sometimes, other entities are involved in the identification (***foreign keys***, ***weak entities***).
- A key for a relationship consists of keys of entities it relates.

# Examples of Keys



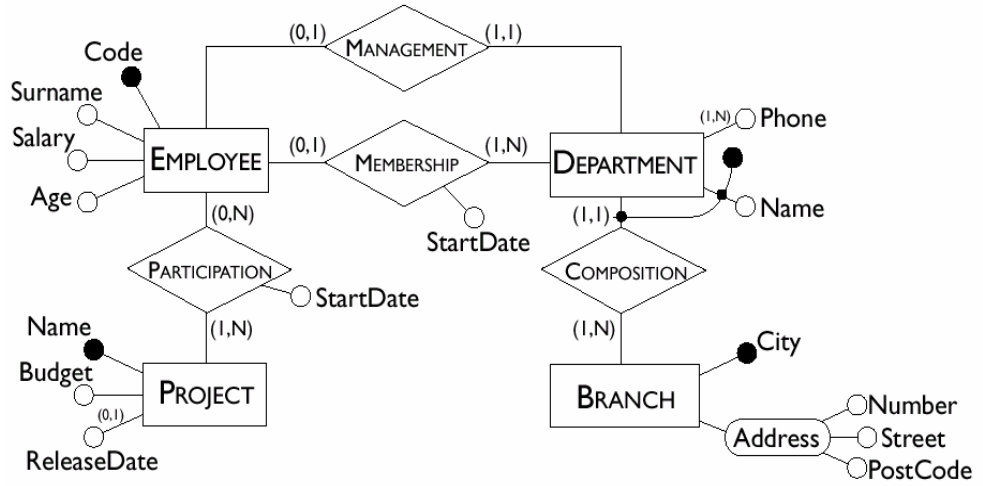





# General Observations on Keys

- A key may consist of one or more attributes, provided that each of them has (1,1) cardinality.
- A foreign key can involve one or more entities, provided that each of them is member of a relationship to which the entity to be identified participates with cardinality equal to (1,1).
- A foreign key may involve an entity that has itself a foreign key, as long as cycles are not generated.
- Each entity must have at least one (internal or foreign) key. If there is more than one key, one of them is labelled as ***primary***.

# Schema with Keys





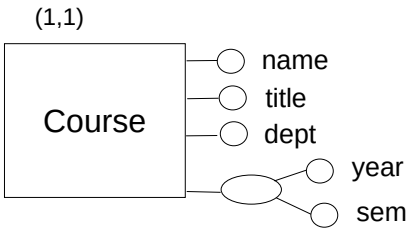



# Modeling an Application with Keys

Keys constitute a powerful mechanism for modeling an application. Assume we want a database storing information about course lectures.

- Suppose first that we use the key **`coursename`**, **`day`**, **`hour`** for the **`Lecture`** entity. What does this say about the application?
- Suppose now we use only **`coursename`** as identifier for **`Lecture`**. What does this say about the application??

# Consider...



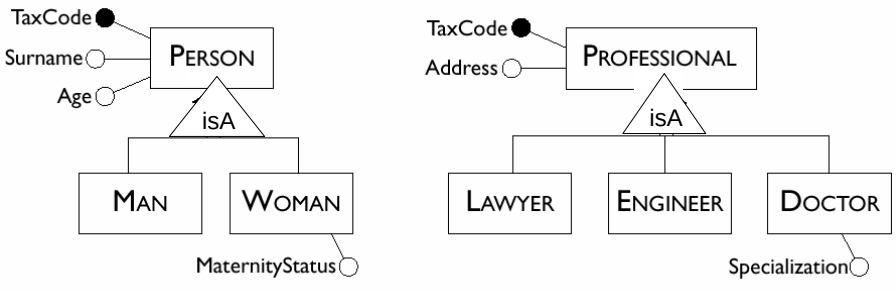

- **Suppose we want information on course offerings over the years. What does each of the following identifiers say about the application?**
  - ✓ **name;**
  - ✓ **name, sem, year;**
  - ✓ **sem, year;**
  - ✓ **name, dept;**
  - ✓ **dept, year.**





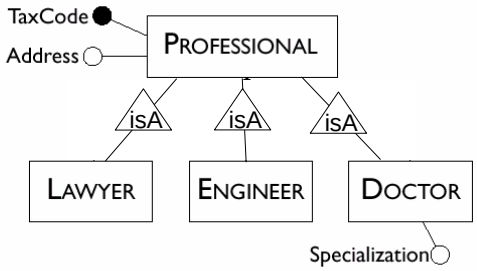
# Entity (Class) Hierarchies

- Entity hierarchies (called class hierachies in the textbook) organize a group of entity sets into a parent/child hierarchy using as criterion their generality/specificity. (Also called ***generalization hierarchies***)



# Overlap Constraints

- Sometimes we want to specify that the children of an entity do/don't overlap.
- For example in previous slide, **`Man/Woman`** don't overlap, neither do **`Lawyer/Engineer/Doctor.`**
- To indicate overlap, we use separate isA links. E.g.,

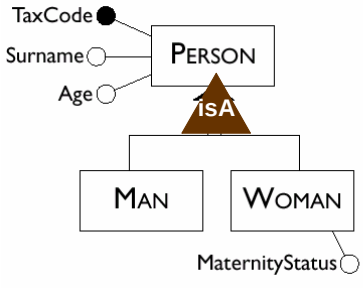


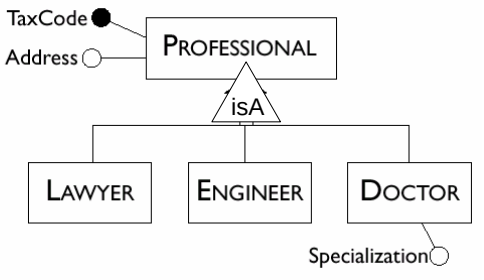





# Covering Constraints

- Now we want to be able to say that instances of the children of an entity include all instances of their parent (I.e., cover it). We use a dark triangle notation



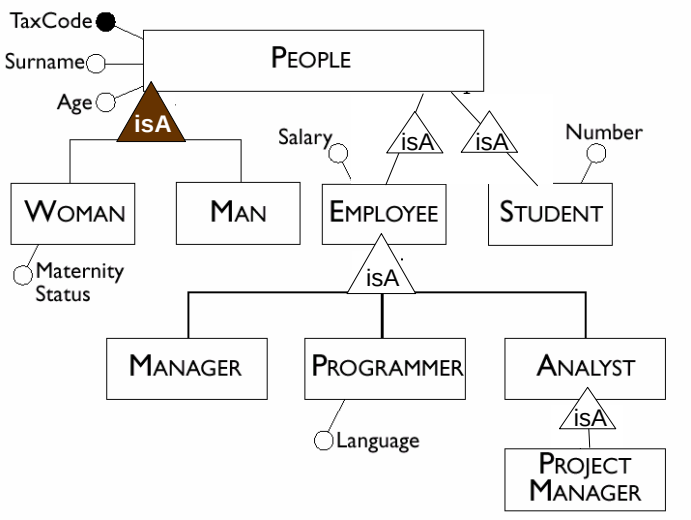
## Properties of Generalization

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





## An Entity Hierarchy



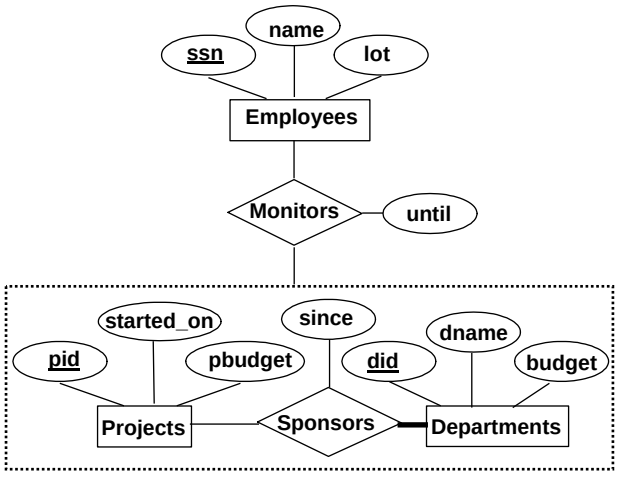
# Aggregation

- Used when we have to model a relationship involving (entity sets and) and a *relationship set*.
- Aggregation allows us to treat a relationship set as an entity set for purposes of participation in other relationships.





# An Example



## Conceptual Design with the ER Model

- **Design choices:** Should a concept be modeled as an entity, an attribute, or a relationship?
- **Constraints on the ER Model:** A lot of data semantics can (and should) be captured; but some constraints cannot be captured by ER diagrams.



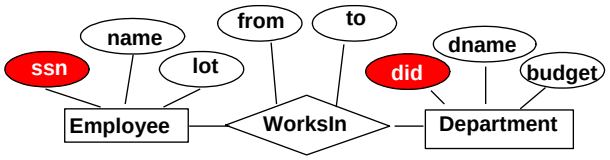


## Some Rules of Thumb

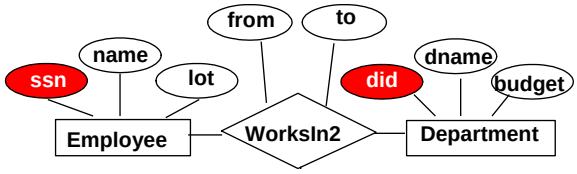
- If a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it as an entity.
- If a concept has a simple structure, and no relevant properties associated with it, it can be represented with attributes of other concepts to which it refers.
- If a concept provides a logical link between 2+ entities, it is convenient to represent it with a relationship.
- If 1+ concepts are particular cases of another concept, it is convenient to represent them in terms of a generalization relationship.

# Entity vs. Attribute

WorksIn does not allow an employee to work in a department for two or more periods.




**How can this be fixed?**

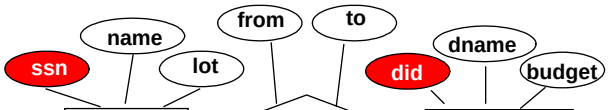


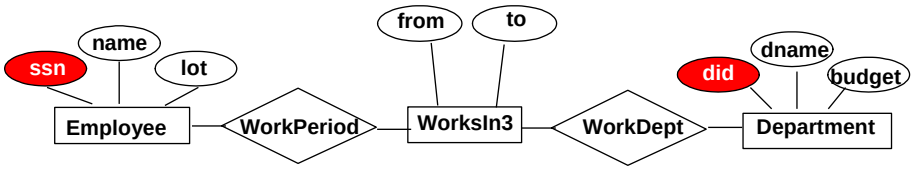





# Entity vs. Relationship

WorksIn can also be turned into an entity to avoid the problem mentioned earlier



## Database Design -- Early Phases

- To design a database, we need to identify requirements, generate a conceptual design, using an ER schema, and from it generate a logical design using a relational schema.
- In the following we discuss how can one go from an informal description of requirements to a conceptual design.





## From Text to an ER Schema

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

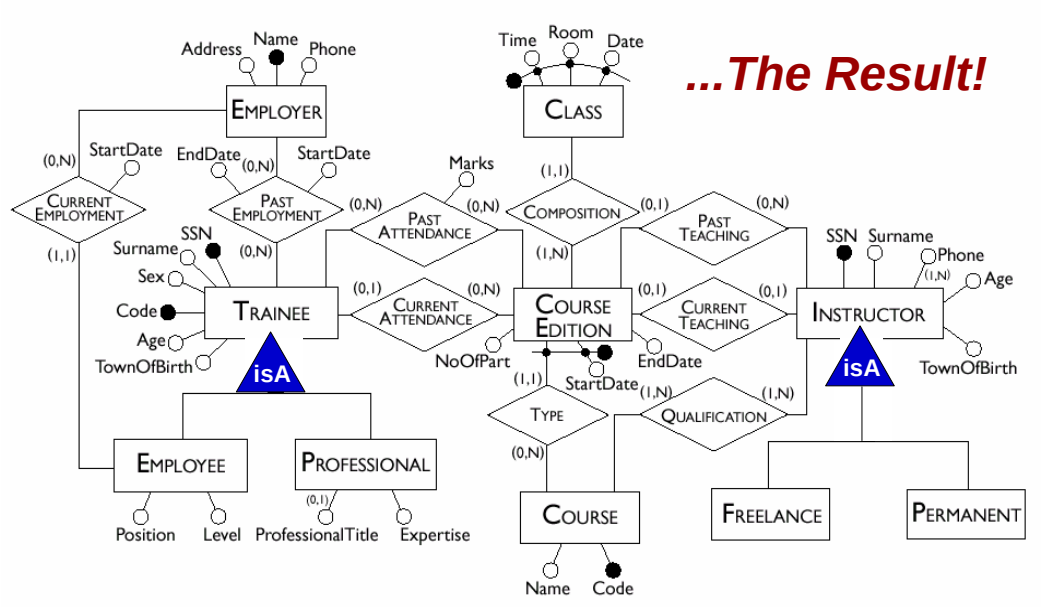
- operation 1: insert new trainee, including her data (40/day);
- operation 2: assign a trainee to an edition of a course (50/day);
- operation 3: insert a new instructor, including all his or her data and the courses he or she is qualified to teach (2/day);
- operation 4: assign an instructor to a course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





## Design Decisions

- Consider **address** of a trainee. Is it an entity or relationship?
- 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.
- How do we represent employment of a trainee by a particular employer?
- How do we represent a course edition?








# Documentation of an E-R Schema — The Data Dictionary

- An ER schema is rarely sufficient by itself to represent all aspects of an application in detail.
- It is therefore important to complement every E-R schema with support documentation, that facilitates the interpretation of the schema and describes 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.





## The Data Dictionary

- 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.<br>**(BR2)** An employee must not have a salary greater than that of the manager of the department to which he or she belongs.<br>**(BR3)** A department of the Rome branch must be managed by an employee with more than 10 years' employment with the company.<br>**(BR4)** An employee who does not belong to a particular department must not participate in any project.<br>.... |
| **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.<br>.... |

# ER Schemas vs UML Class Diagrams

- The ER model allows N-ary relationships, N≥2; Class diagrams only allow binary relationships.
- ER schemas allow multi-valued attributes, class diagrams do not.
- ER schemas include keys (an often-encountered type of constraint), class diagrams do not.
- Class diagrams allow dynamic classification, but ER diagrams do not.
- Class diagrams can have associated methods, constraints and pre/post-conditions.