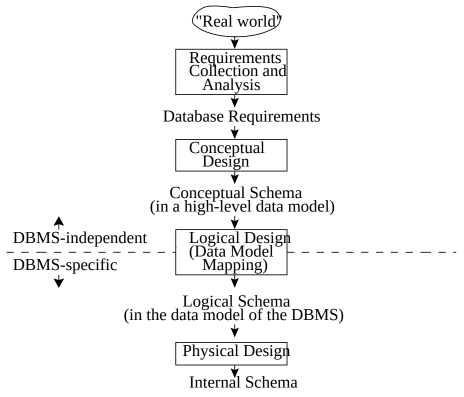
Course Notes on
Entity-Relationship Data Model

Entity-Relationship Data Model

- Classical, popular conceptual data model
- First introduced (mid 70's) as a (relatively minor) improvement to the relational model: pictorial diagrams are easier to read than relational database schemas
- Then evolved as a popular model for the first conceptual representation of data structures in the process of database design
- Today:
  - ◇ no ER standard, large variety of notations and concepts
  - ◇ simple versions in popular models, tools, and languages
  - ◇ richer versions in "semantic" data modeling
  - ◇ basis for the structural model of modern OO development methods

- Vocabulary:
  - ◇ "entity-relationship" is often translated as "entité-association" in French
  - ◇ English has "relation" and "relationship"
  - ◇ French only has "relation"
  - ◇ Spanish only has "relación"
  - ◇ Brasilian Portuguese has "relaçao" and "relacionamento", but mostly uses "relationship" :-)

Database Design Process


"Real world"
Requirements Collection and Analysis
Database Requirements
Conceptual Design
Conceptual Schema (in a high-level data model)
DBMS-independent
DBMS-specific
Logical Design (Data Model Mapping)
Logical Schema (in the data model of the DBMS)
Physical Design
Internal Schema

Basic Concepts of the ER Model

- Entities
- Relationships
  - ◇ Binary and N-ary
  - ◇ Cardinality constraints
  - ◇ Recursive relationships
- Attributes
  - ◇ Cardinality constraints
  - ◇ Simple versus composite
  - ◇ Stored versus derived
  - ◇ Identifiers
  - ◇ Weak entities
- Generalization
  - ◇ Total versus partial
  - ◇ Exclusive versus overlapping
  - ◇ Predicate- versus user-defined

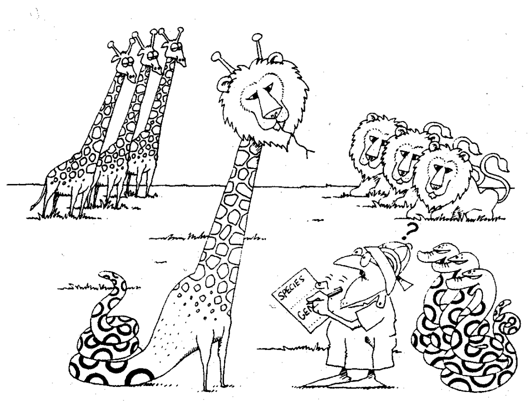
Entities and Entity Classes

- **Entity**:
  - ◇ important individual thing, object, concept in the real world of interest (e.g., the person called John, my red car, . . .)
  - ◇ **physical** (e.g., persons, cars, . . .) or **conceptual** (e.g., companies, jobs, . . .)
- **Entity class**:
  - ◇ concept, type, common prototype (**intension**), set of potential instances
  - ◇ current collection of instances (**extension**)
  - ◇ mechanism for **creating instances** (“object factory”, “cookie cutter”)

4

- **Vocabulary**:
  - ◇ the distinction bewteen type or class and their instances is classical in informatics (programming languages)
  - ◇ *entity* is often used for individuals (instance, occurrence), *entity* is also used for *entity type*, *entity set* and *entity class*
  - ◇ an important difference with the use of types in programming languages is that database management puts more emphasis on the extension of an entity class in a database, i.e., the current collection of instances of the entity type
  - ◇ in practice, for this chapter: “entity” ≈ “object”

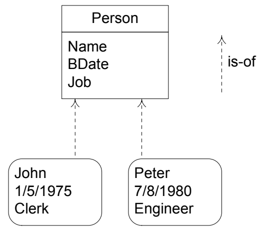
Grouping Entities into Classes



5

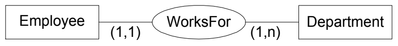
Class-Instance Relationship

- The relationship between a class or type and its instances is called **instantiation** / **classification**
- A class defines a common template for its instances
- Intances have a value for each attribute defined in their class

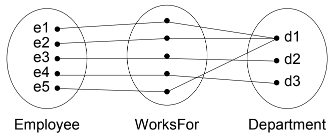


6

General Idea of ER Relationships



- A **relationship type** R among entity types $E_1, \dots, E_n$ models a structural (i.e., static) association between real-world entities
- **Relationship instances** $r_i = (e_1, \dots, e_n)$ of R, where each $e_i \in E_i$, model links between individual entities (instances of the real-world association)



7

- **Notation**: relationship types are sometimes drawn as diamonds, sometimes as simple lines (e.g., UML)
- **Vocabulary**: as for entities, “relationship” is often used for both types and instances (context usually makes things clear)
- **Degree** of a relationship type
  - degree = number of participating entity types (binary, ternary, . . .)
  - in practice, most relationships are binary (WHY?), but they are not sufficient in general (there is more information in a ternary relationship than in the 3 associated binary relationships, see later)

Relations and Functions in Classical Mathematics

- **Binary relation** between two sets A and B = {ordered pairs $(a, b) \mid a \in A$ and $b \in B$}
- **N-ary relation** between n sets A_1, ..., A_n = {ordered n-tuples $(a_1, ..., a_n) \mid a_i \in A_i$}
- More precisely: **N-ary relation** $\subseteq$ Cartesian product $A_1 \times ... \times A_n$
- Unordered tuples = add an index to identify each set in Cartesian product
- **Binary function** from set A (domain) to set B (range) = relation between A and B where each $a \in A$ is associated with at most one $b \in B$
- N-ary function: from a Cartesian product of sets A_1, ..., A_n to a set B
- Functions are special cases of relations, with an intuition of **direction**, **order**

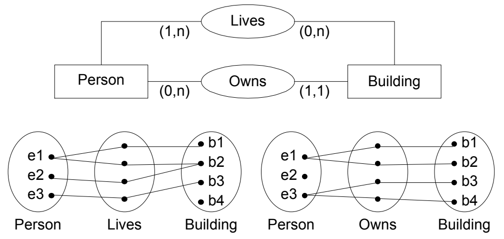
8

Relationship Semantics in the ER Model

- Early version of relationships (mid 70’s): not a really new and important concept, essentially an informal pictorial notation
- Current perception:
  - a relationship associates objects from n classes
  - simple semantics = set of n-tuples of objects (aggregation-style)
  - one-to-one, one-to-many, many-to-many
  - more precisely: multiplicity or cardinality
  - degree: binary, ternary, n-ary
  - can have attributes
  - roles = several names for preferred directions of relationship traversal (mandatory only for recursive relationships)

Relationship Cardinalities

- Minimum and maximum number of relationship instances in which each entity can participate
- The cardinality information belongs to the schema (i.e., it is valid for all future extensions of the database)



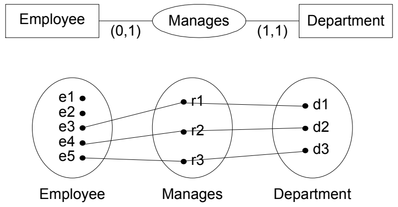
- “Multiplicity” is sometimes used for “cardinality”
- Cardinalities are positive integers (0,1,. . .) or “n” (no limit)
- (0,n) = no constraint (also noted as ’*’, e.g., in UML)
- Cardinality constraints may have to be relaxed in the initialization stages (the first employee entered in the database cannot be assigned to a department thru WorksFor if no department has been created yet)
- Two ways of noting cardinalities:
  1. on the side of the *origin* entity as above (e.g., each Building partipates in a number of instances of Owns comprised between 1 and 1, i.e., equal to 1)
  2. on the side of the *target* entity (e.g., starting from any Building, the number of Persons reached thru Owns is comprised between 1 and 1, i.e., equal to 1)
- The second notation is that of UML; the first one extends more smoothly to n-ary relationships
- The second notation suggests a view of relationships as two mappings

Other Vocabulary for Relationship Cardinalities

- Cardinalities: two pairs of integers say it all
- Optional and mandatory participation
  - ◇ E is **optional** in R: min-card(E,R) = 0
  - ◇ E is **mandatory** in R: min-card(E,R) $\geq$ 1
- Classification of a binary relationship R according to maximal cardinalities of participating entities E and F
  - ◇ **one-to-one**: max-card(E,R) = 1, max-card(F,R) = 1
  - ◇ **one-to-many**: max-card(E,R) = 1, max-card(F,R) = n
  - ◇ **many-to-many**: max-card(E,R) = n, max-card(F,R) = n

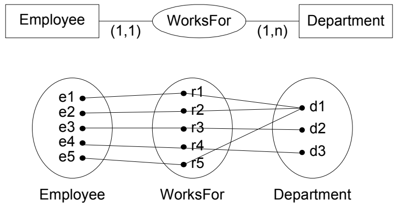
- Lives is is a many-to-many relationship
  - ◇ mandatory for Person
  - ◇ optional for Building
- Owns is is a one-to-many relationship
  - ◇ optional for Person
  - ◇ mandatory for Building

A one-to-one Relationship



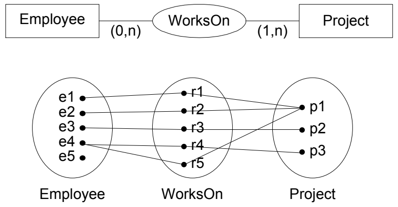
11

A one-to-many Relationship



12

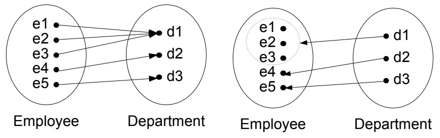
A many-to-many Relationship



13

Relationships as Mappings

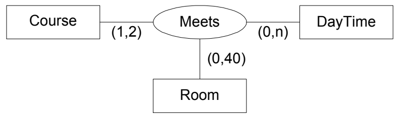
- Mappings generalize functions
- An object of a class is mapped to an object or to a set of objects of another class
- Mappings can be viewed as functions if sets can be viewed as objects
- Binary relationship = 2 mappings (one for each direction of traversal)

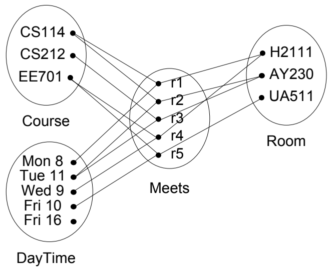


14

- Bidirectional or "oriented" relationships?
  - ◇ mathematical relations are not oriented (binary relations are inherently bidirectional)
  - ◇ most versions of the entity-relationship model treat relationships as not oriented; some versions view them as directed
  - ◇ asymmetry comes from
    - ∗ linguistics: relationship names rarely correspond to traversal in both directions
    - ∗ implementation techniques: a traditional implementation of relationships is as pointers from one entity to related entities
    - ∗ implementation efficiency: one direction of traversal may be more efficiently implemented (but this violates data independence)
    - ∗ application needs: it may be justified to privilege one direction of traversal

Cardinalities of n-ary Relationships



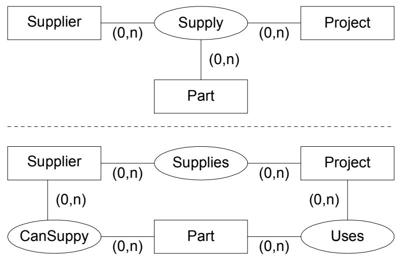


- 40 = 5 days × 8 time-slots; $n \leq 8\times$ number of rooms

15

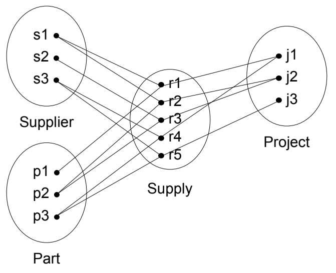
- The definition given for the cardinalities of binary relationships still holds: minimum and maximum number of instances of the relationship in which each entity can participate
- There is another definition of cardinalities, that extends differently to n-ary relationships

A Ternary Relationship and the 3 Binary Relationships Derived from it by Projection



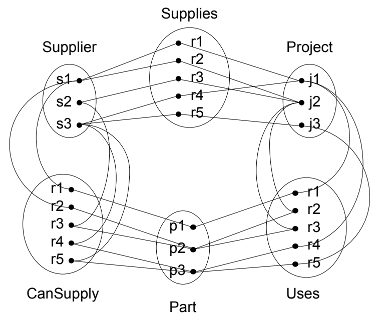
16

Examples of Instances of Supply



17

Instances of the Binary Relationships Derived from Supply



- The 3 binary relationships have the same number of instances as the ternary relationship (5)
- The binary relationships are derived by "projection" from the ternary relationship ⇒ there is certainly no more information in the 3 binary relationships than in the ternary relationship
- What about the reverse? Is there more information in the ternary relationship than in the 3 binary relationships?

Ternary Relationships Contain More Information

- There is less information in the 3 binary relationships derived form a ternary relationship than in the ternary relationship

Supply

Supplier	Client	Part
Dupont	Client1	pencil
Durand	Client1	pen
Dupont	Client2	pen

Supplies

Supplier	Client
Dupont	Client1
Durand	Client1
Dupont	Client2

Uses

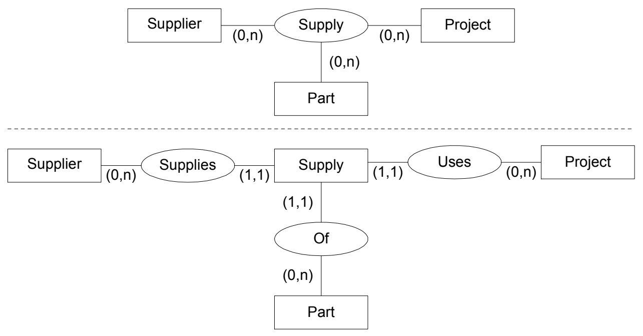
Client	Part
Client1	pencil
Client	pen
Client2	pen

CanSupply

Supplier	Part
Dupont	pencil
Durand	pen
Dupont	pen

- There is no way to tell from the binary relationships that supplier Dupont does not supply pen to Client1

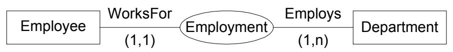
Correct Modeling of N-ary Relationships with Binary Relationships



- How to correctly do without n-ary relationships:
  - ◇ “objectify” or “reify” the n-ary relationship into a **weak** entity type (whose identifier is composite and made of the identifiers of the n participating entities, see later), and
  - ◇ link the the new entity type to each original entity type with a binary relationship
- In the example above, the new entity type could model orders, invoices, ..., according to the semantics of the original Supply relationship
- Advantage of using binary relationships only: the data model is simpler
- Advantage of using n-ary relationships:
  - ◇ they may express the most natural model (if the underlying reality is naturally perceived as an n-ary association)
  - ◇ the “objectified relationship” and the accompanying binary relationships may or may not be natural in the perception of the application domain
  - ◇ schemas with only binary relationships are typically larger: they have more entity types and more relationships than the equivalent schemas with n-ary relationships
- Summary: doing away with n-ary relationship yields a simpler data model (binary relationships only) at the expense of larger schemas and possibly less natural models
- Both views (an n-ary relationship and its “objectified” entity type) could coexist in a rich and redundant data model (this may be natural, as both perceptions may be reasonable in the underlying reality)
- Further discussion: what about the cardinalities of binary relationship obtained by projection, by reification?

Roles in Relationships

- Each entity type plays a **role** or function in a relationship

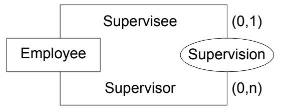


- Employee plays role WorksFor and Department plays role Employs
- Role names are convenient carriers for alternative (redundant) relationship names: relationships can thus have
  - ◇ a neutral name (Affiliation)
  - ◇ names (WorksFor, Employs) from the point of view of each entity involved, suggesting directed traversal of the relationship
- Role names are similar to class attributes (attributes whose value is the same for all entities in an entity type)

21

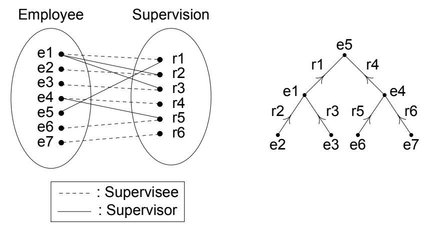
Role Names and Recursive Relationships

- **Recursive** (or cyclic) relationship: involves the same entity type more than once
- Roles are **mandatory** in recursive relationships



22

Example Instances for Supervision

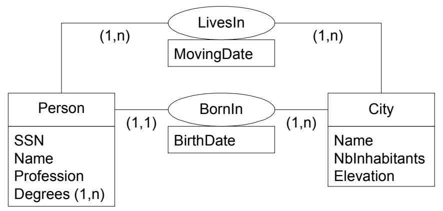


23

- The graph defined by the cardinalities of Supervision is not a tree nor a DAG (directed acyclic graph)
- Explicit constraints must restrict the schema to the desired kind of graph

Attributes

- Attributes model details of entities or relationships



- An entity or relationship instance has a **value** for each attribute
- Attributes have a **value set** or **domain** (set of possible values)
- Domain values are distinct, more abstract than their interpretation as lower level data (name of person, profession of person, name of city are distinct domains although represented as strings of characters)

24

Attribute of an entity or of a relationship?

- Relationship attributes are really necessary only for n-ary and many-to-many binary relationships
- Otherwise, attributes can be attached to entities, although it may be natural to attach attributes to binary relationships
  - ◇ MovingDate can be an attribute of relationship LivesIn or of entity type Person
  - ◇ if the history of residences becomes relevant (i.e., the maximal cardinality of Person in LivesIn is ≥ 1), then MovingDate is necessarily an attribute of LivesIn
  - ◇ schema evolution (from modeling a single residence to modeling their history) is made easier

Attribute Cardinalities

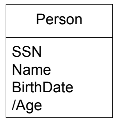
- Like relationships, attributes have minimal and maximal cardinalities
- Optional ⇔ Mandatory
  - ◇ **mandatory**: min-card ≥ 1
  - ◇ **optional**: min-card $= 0$
- Several possible meanings for "**null values**":
  - ◇ **not applicable:** (a person does not have a university degree)
  - ◇ **unknown but known to exist:** (phone number exists, unknown to the database)
  - ◇ **unknown, not known to exist:** (phone number may exist)
- Single-valued ⇔ Multivalued
  - ◇ **single-valued**: max-card $= 1$ (one value for each entity, e.g., Age)
  - ◇ **multivalued**: max-card > 1 (a set of values for each entity, e.g. Degrees)

25

- Notations
  - ◇ the most frequent case of attribute cardinality is (1,1), i.e., mandatory monovalued attributes; this is often taken as the default notation (i.e., when no cardinality is indicated explicitly, then (1,1) is meant)
  - ◇ the cardinality (0,n) for optional multivalued attributes is sometimes noted as a $*$ (e.g., attribute Degree of Person)

Derived Attributes

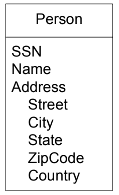
- Two or more attributes can be related



- The value of Age may be determined from the current date and the BirthDate: Age is a **derived** attribute
- The constraint must be noted in the schema (and the redundancy properly managed)
- How derived attributes are implemented is another issue

26

Composite Attributes



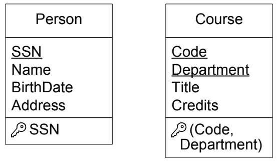
- **Composite** attribute: formed by aggregating related attributes
- Nondivisible attributes are called **simple** or **atomic**
- Composite attribute:
  - ◇ its value is a tuple of values for its constituent simple attributes
  - ◇ introduces a third level of structure (entity, composite attribute, simple attribute)
  - ◇ represents groups of values that go together but are not thought important enough to make up an entity

27

Attribute or Entity?

- There is no fundamental difference between an attribute and an entity, just a matter of emphasis
- Choose an entity instead of an attribute if the piece of data to be modeled has a well defined identity and is naturally related to other attributes, for example:
  - ◇ the hometown of students
    - ∗ if the hometown name is the only attribute of interest, then model hometown as an attribute
    - ∗ if some demographic information is of interest (like population or primary industry), then model hometown as an entity
  - ◇ color
    - ∗ the color of a book cover or of a bicycle is probably best modeled as an attribute
    - ∗ for a paint company, color may be an important entity with attributes, and linked to other entities (e.g., dyes necessary to manufacture the color)
  - ◇ job skills can be simply an attribute of employees or they can be important corporate ressources modeled as entities
- Composite attributes are not present in many versions of the ER model
- Deciding between an attribute and an entity is thus not an absolute decision; this illustrates that
  - ◇ modeling is a creative activity that cannot be automated
  - ◇ there are many ways to represent the same "real world" in a database

Identifiers



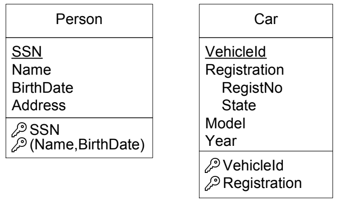
- **Identifier** or **key** of entity type E: attribute or set of attributes of E whose values uniquely determine an entity of E
- **Alternative notations**: underlined attributes or in a constraint box
- Attributes involved in an identifier must be of cardinality (1,1)
- Two facets
  - ◇ **unique identification**: an identifier value selects at most one entity
  - ◇ **unicity constraint**: no two entities of the same type with the same value for an identifier
- Identifier definition belongs to the schema

28

Beware of "Key"!

- *Key* is overloaded in informatics:
  - ◇ identifying key (here)
  - ◇ search key (in indexes, for example, see later)

More on Identifiers

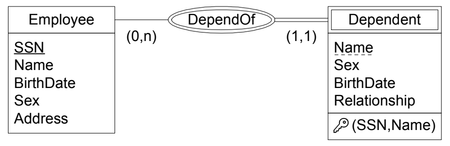


- Some entity types have **more than one** identifier
- **Notation**: Only one of them can be represented using the "underlined" notation, the others must be represented in the constraint box
- An identifier with several attributes can be made a composite attribute

29

Weak Entity Types

- Have no identifier consisting exclusively of their own attributes
- Identification involves another entity type related with (1,1) cardinality

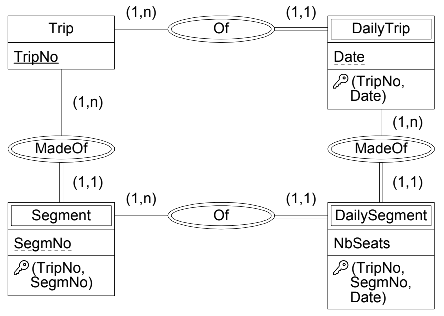


- A weak entity type has a **partial identifier** uniquely identifying entities within those related to the same owner
  - ◇ Name is a partial identifier of Dependent

- In this example, the various dependents of a given employee can be distinguished by their name, but dependents of distinct employees can have the same name; to distinguish a dependent from all other dependents, the name of the dependent and the identification of an employee are needed
- Alternative to a weak entity: include in the dependent entity a redundant identifier of the owner entity (⇒ redundancy has to ne managed with an appropriate constraint in the schema)
- Suppress the weak entity in this example:
  - ◇ add a SSN attribute to Dependent
  - ◇ add the following constraint: for each dependent d, the value of its SSN attribute is equal to the SSN value of the employee linked to d by an instance of relationship DepOf

Weak Entity Types (cont'd)

- A weak entity type may have several identifying owners

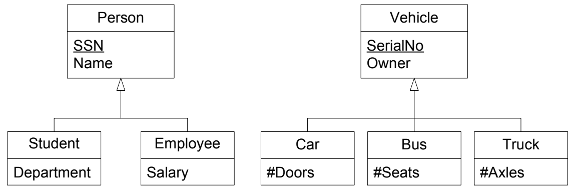


- Interest of weak entity types:
  - ◇ avoid redundant attributes (and associated constraint management)
  - ◇ do without n-ary relationships
  - ◇ similar to inheritance in generalizations

Generic Relationships

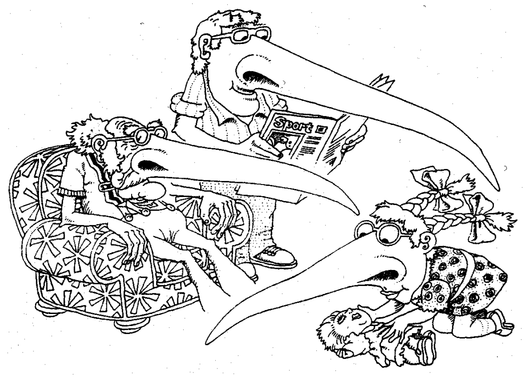
- Generic relationships = templates to be specialized for relating classes
  - ◇ generalization: Superclass ⟵ Subclass
  - ◇ classification: Class<- - -Instance
  - ◇ aggregation: Whole◇—Part
  - ◇ materialization: Abstract—∗Concrete
- Generic relationships abstract specific relationships: they can be viewed as metarelationships
  - ◇ they are part of the language for defining ER schemas (specific relationships are part of schemas)
  - ◇ a specific relationship (e.g., Person ⟵ Employee is an instance of the corresponding generic relationship

Generalization



- Special relationship between several **subentities (subclasses)** and a higher-level, more abstract **superentity (superclass)**
- **Specialization**: the relationship viewed from the superclass
- Two aspects
  - ◇ **inheritance**: built-in mechanism
  - ◇ **is-a** relationship: relationship with specific semantics; **substitution principle**: instances of subclasses can also be viewed as instances of their superclasses

Generalization: Inheritance

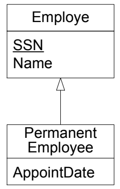


Intuitive definition of inheritance

- built-in object-oriented and ER **mechanism**
- whose effect is as if all properties of the superclass had also been defined for all subclasses

Subsets

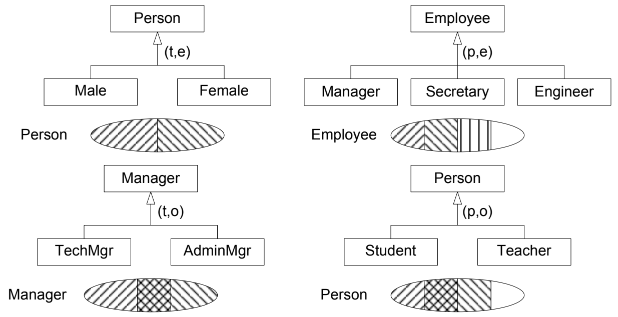
- Special case of generalization with a single subentity



35

Coverage Properties of Generalization

- Total (t) or Partial (p)
- Exclusive (e) or Overlapping (o)

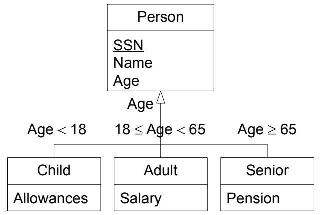


36

- The more general (less constraining) case (partial + overlapping) is equivalent to one subset relation per subentity
- Total and exclusive generalizations are easiest to handle
- Exercise (easy): transform any generalization into a total and exclusive generalization

Specialization Defined by a Predicate

- Participation in subentities is sometimes specified by a condition on attribute values in the superentity

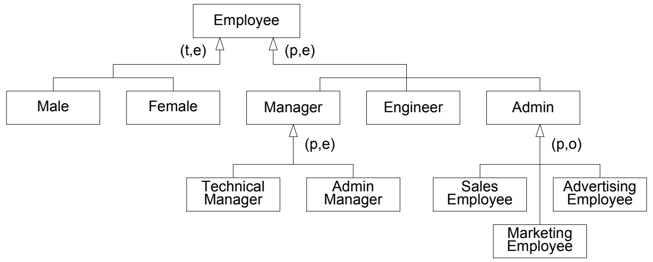


37

- If specialization is defined by a predicate, then the participation of an entity in a subentity can be decided automatically
- Otherwise, membership in subclasses must be specified explicitly (or “manually”)

Generalization Hierarchies

- An entity can be involved in **multiple parallel generalizations**
- The same entity can play the role of generic entity and specific entity for different generalizations

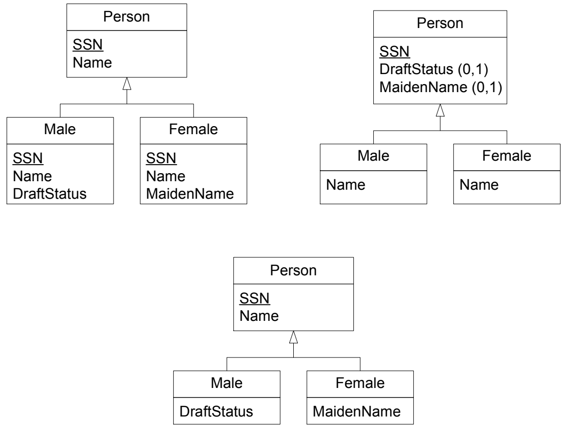


- A subentity **inherits all the abstractions** not only of its direct superentities but also of all its predecessor superentities all the way to the root

38

- Without generalization, models comprise many "intersection entities" that are more or less natural

Good Use and Bad Use of Generalization

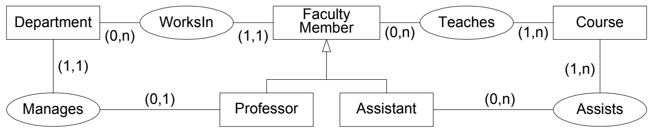


39

- Attributes must be placed at the appropriate level (as high as possible to avoid redundancy, as in the second schema)
- The first schema is incorrect: multiple inheritance is not defined in this simple ER model
- The second schema will accept more instances than the third, unless an integrity constraint for the second makes them equivalent (the only persons with a draft status are males and the only persons with a maiden name are female; all males have a draft status and all females have a maiden name)

Inheritance of Relationships

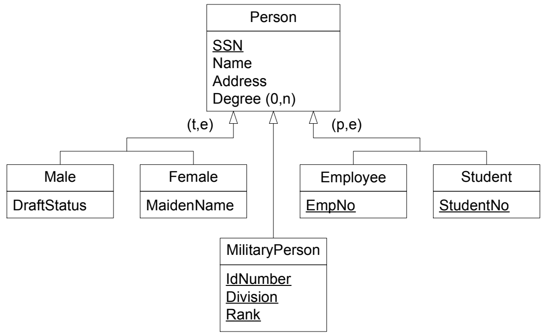
- **Relationships are also inherited**: they must be placed at the highest applicable level



- Relationships should be placed at the appropriate level (as high as possible to avoid redundancy)
- They should not be placed too high
- Here Professors and Assistants can both teach Courses, while only Assistants assist Courses

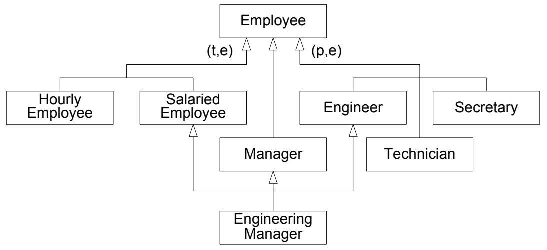
Inheritance of Identifiers

- **Identifiers are inherited** like the other attributes
- A generalization may introduce **additional specific identifiers**



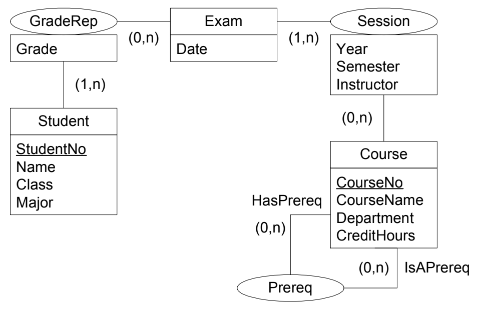
Multiple Inheritance/Shared Subentities

- **Shared subentity**: a subentity with more than one superentity
- **Multiple inheritance**: subentities inherit from all of its superentities

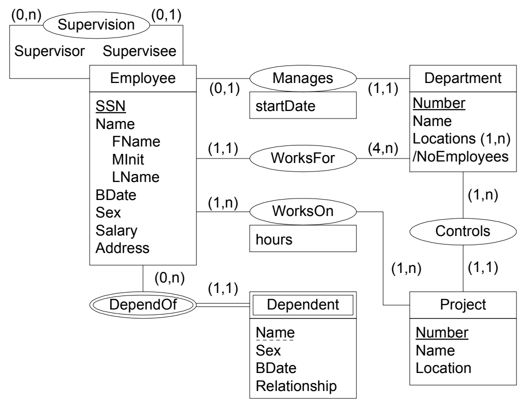


- Shared subentity induces generalization lattices
- Multiple inheritance **may generate conflicts**, that must be solved one way or another

ER Schema for the University Example



ER Schema for the Company Example



Requirements for the Company example

(1) The company is organized into departments. Each department has a name, a number, and an employee who manages the department. We keep track of the start date when that employee started managing the department. A department may have several locations.

(2) A department controls a number of projects, each of which has a name, a number, and a single location.

(3) We store each employee's name, social security number, address, salary, sex, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.

(4) We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's name, sex, birth date, and relationship to the employee.

Cycles in ER Schemas

- A cycle in an ER schema is often associated with a constraint
- Example (university database):
  - ◇ there is a cycle Course – HasPrereq – Prereq – IsAPrereq - Course
  - ◇ a reasonable constraint states that there is no cycle in the instances of relationship Prereq, i.e., a course cannot be (directly or indirectly) a prerequisite for itself
  - ◇ in other words, the graph of relationship Prereq is acyclic (it is a partial order)

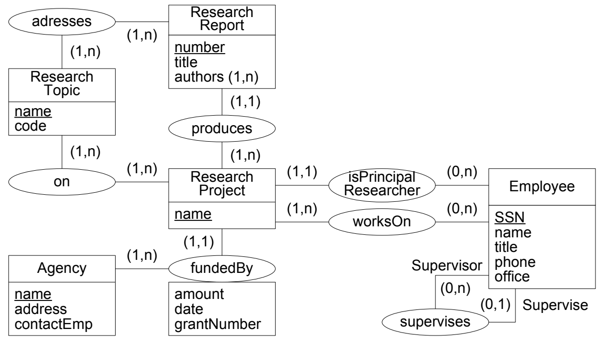
Other examples of cycles in the company database:

- Example 1:
  - ◇ cycle Employee – Supervisor – Supervision – Supervisee – Employee
  - ◇ same structure as Example 1: no cycle in the instances of relationship Supervision
- Example 2 :
  - ◇ cycle Employee – Manages – Department – WorksFor – Employee

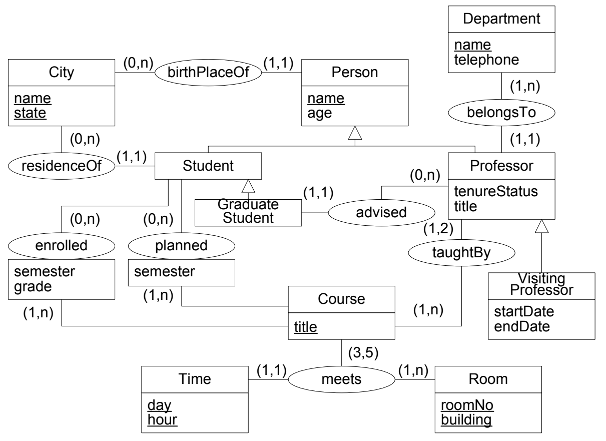
    ◇ plausible constraint: every employee who manages a department also works for that department

- Example 3 :
    ◇ cycle Employee – WorksFor – Department – Controls – Project – WorksOn – Employee
    ◇ there are 2 sets of projects associated to each employee: the set of projects on which the employee works on (relationship WorksOn and the set of projects controled by the department for which the employee works for (link form Employee to Department via WorksFor and from Department to Project via Controls
    ◇ a constraint may constrain these 2 sets (includes, is included in, identical, no intersection)
- Note that not saying anything means that there is no constraint on the instances

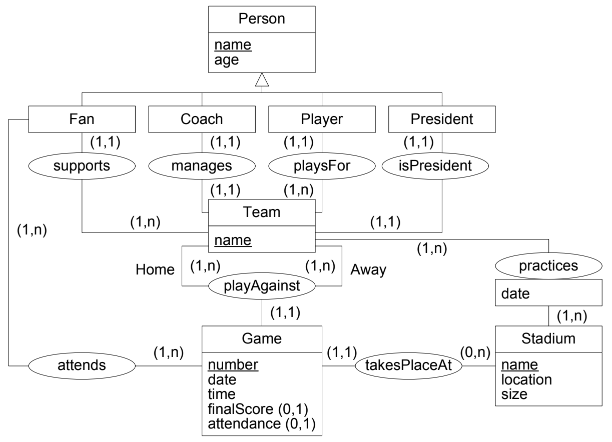
ER Schema: Research Database



ER Schema: University Database



ER Schema: Football Database


Person
name
age
Fan
Coach
Player
President
(1,1)
(1,1)
(1,1)
(1,1)
supports
manages
playsFor
isPresident
(1,1)
(1,n)
Team
name
(1,n)
(1,n)
(1,1)
(1,n)
Home
(1,n)
(1,n)
Away
playAgainst
practices
date
(1,1)
(1,n)
Game
number
date
time
finalScore (0,1)
attendance (0,1)
attends
(1,n)
(1,1)
takesPlaceAt
(0,n)
Stadium
name
location
size