FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REAL WORLD MODEL
Entity-Relationship (ER)

DATABASES - lectures

Dr. Sonja Gievska

Outline

Databases are omnipresent!

- Because of their diversity, databases are the basis of different types of projects:
  - Web site for on-line shopping
  - Application for recording insured persons in a pension fund
  - Healthcare medical system
  - Personal address book for your e-mail client
  - iKnow working system for the University
  - Air ticket reservation system. . .

Basic definitions

- **Database (DB)**
  - A collection of related data.
- **Data**
  - Known facts that can be recorded and have an implicit meaning.
- **Mini-world**
  - Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.
- **Database Management System (DBMS)**
  - A software package/ system to facilitate the creation and maintenance of a computerized database.
- **Database System**
  - The DBMS software together with the data itself. Sometimes, the applications are also included.

Simplified database system environment

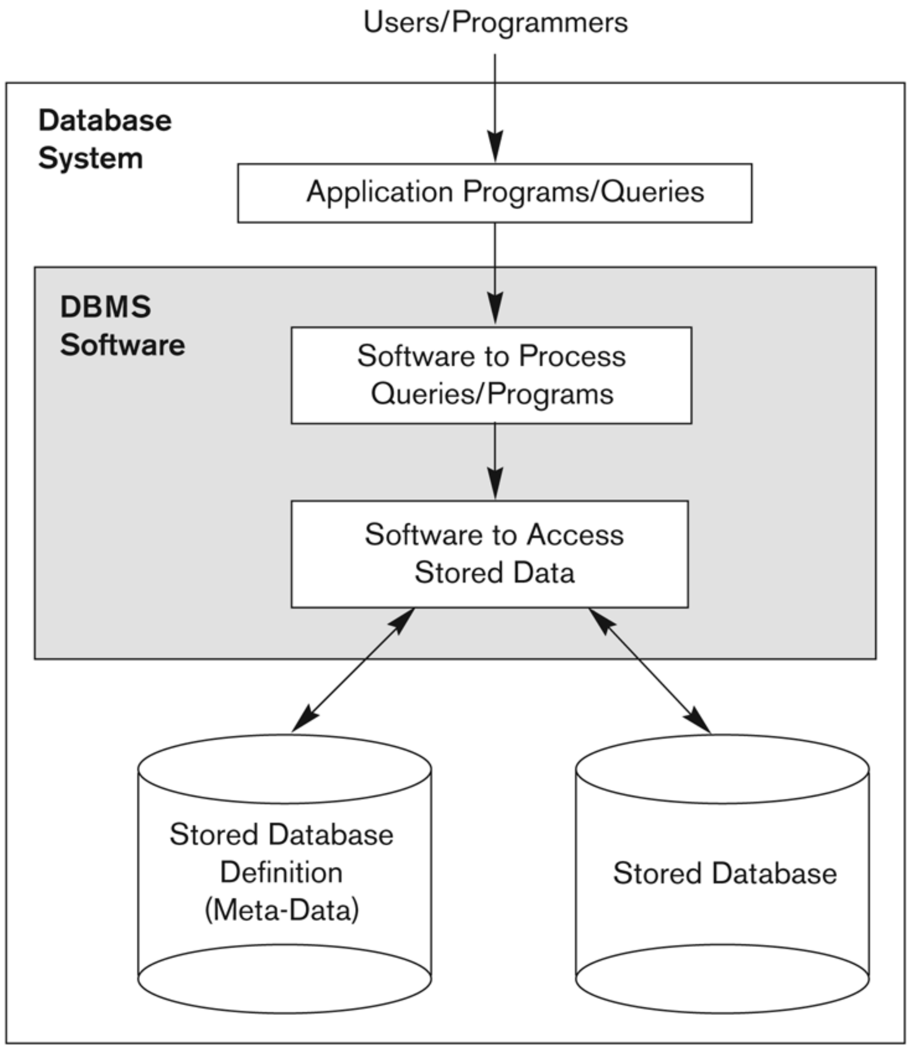

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure 1.1
A simplified database system environment.

Typical DBMS Functionality

- *Define* a particular database in terms of its data types, structures, and constraints
- *Construct* or Load the initial database contents on a secondary storage medium
- *Manipulating* the database:
  - Retrieval: Querying, generating reports
  - Modification: Insertions, deletions and updates to its content
  - Accessing the database through Web applications
- *Processing* and *Sharing* by a set of concurrent users and application programs – yet, keeping all data valid and consistent

Typical DBMS Functionality

- Other features:
  - Protection or Security measures to prevent unauthorized access
  - “Active” processing to take internal actions on data
  - Presentation and Visualization of data
  - Maintaining the database and associated programs over the lifetime of the database application
    - Called database, software, and system maintenance

Overview of Database Design Process

- Two main activities :
  - Database design
    - To design the conceptual schema for a database
  - Applications design
    - Focuses on the programs and interfaces that access the database
    - Generally considered part of software engineering

Overview of Database Design Process

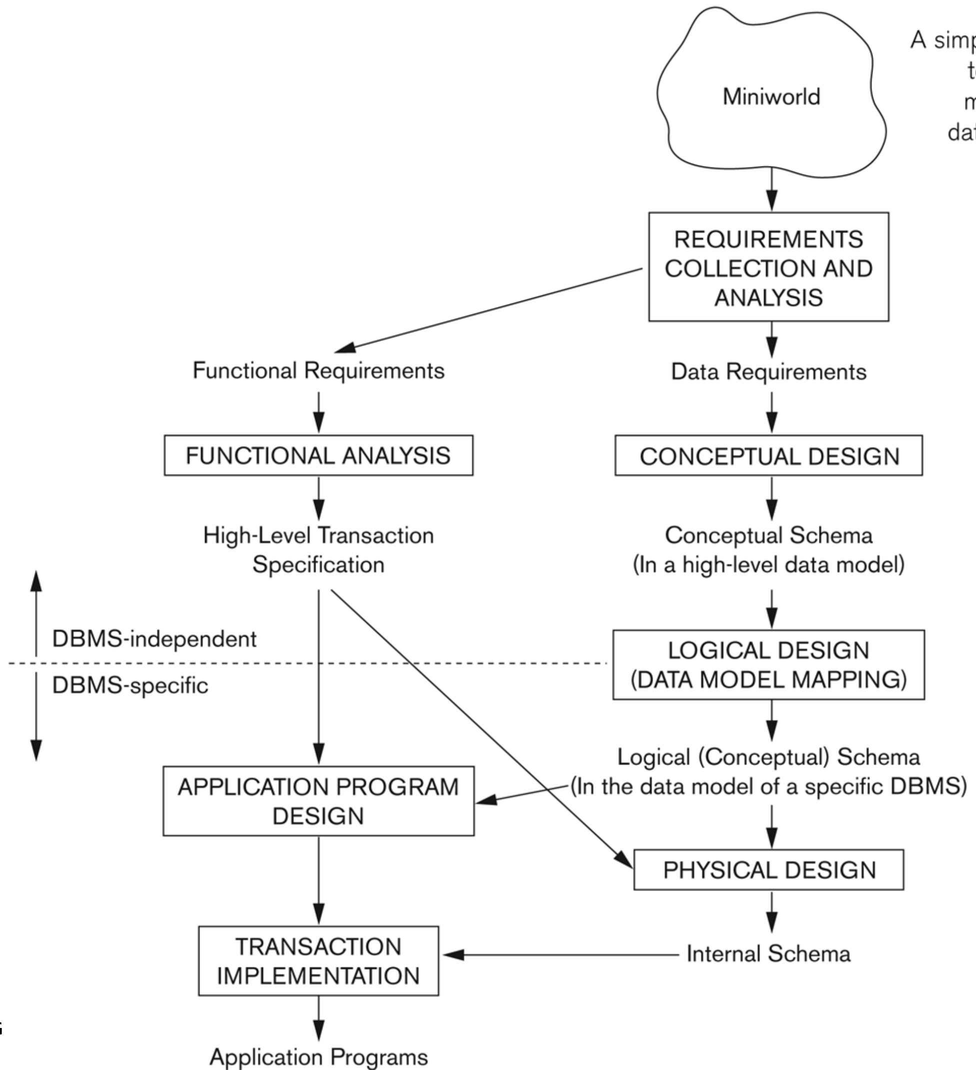

Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Database Schema

- Database Schema:
  - The ***description*** of a database
  - Includes descriptions of the:
    - database structure,
    - data types,
    - constraints on the database

It is common to give an illustrative display of (most aspects of) a database schema using a
DATABASE SCHEMA DIAGRAM

Example COMPANY Database (1)

- We need to create a database schema design based on the following (simplified) **requirements** of the COMPANY Database:
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
  - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

Example COMPANY Database (2)

- We store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
  - Each employee *works for* one department but may *work on* several projects.
  - We keep track of the number of hours per week that an employee currently works on each project.
  - We also keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of DEPENDENTs.
  - For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.

ER Model Concepts

- **Entities and Atributes**
  - Entities are specific objects or things in the mini-world that are represented in the database.
    - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
  - Attributes are properties used to describe an entity.
    - For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate
  - A specific entity will have a value for each of its attributes.
    - For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘
  - Each attribute has a *value set* (or data type) associated with it
    - For example integer, string, subrange, enumerated type, …

Entity examples

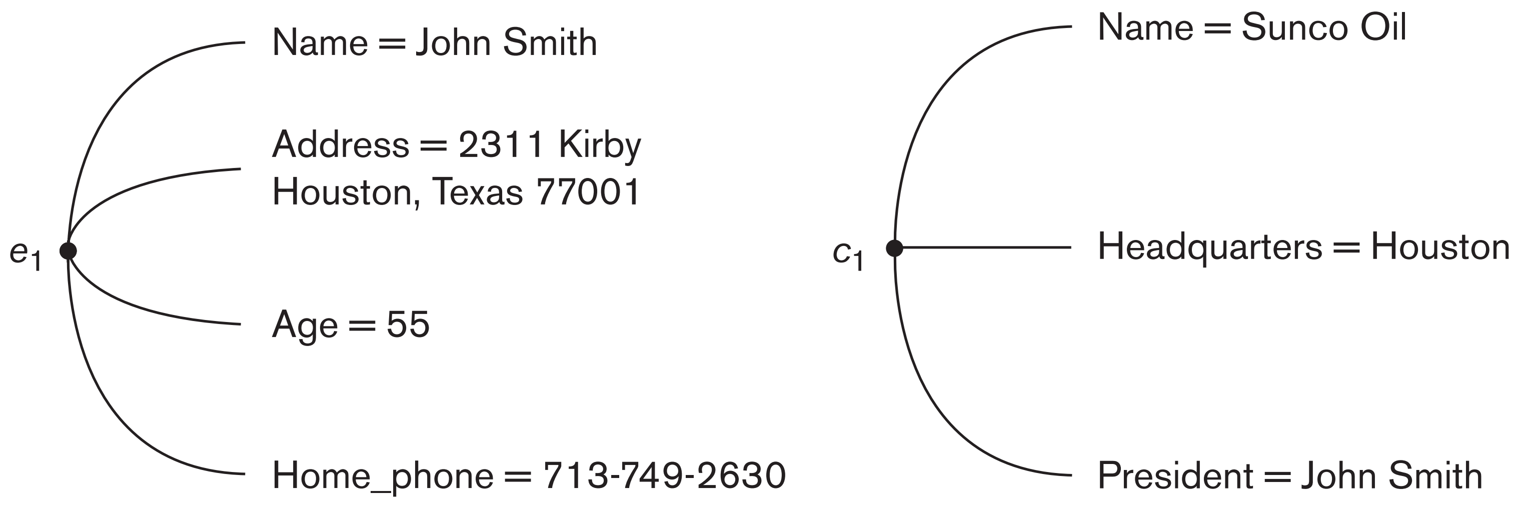


Figure 7.3
Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes.

Types of Attributes (1)

- **Simple**
  - Each entity has a single atomic value for the attribute. For example, SSN or Sex.
- **Composite**
  - The attribute may be composed of several components. For example:
    - Address(Apt#, House#, Street, City, State, ZipCode, Country), or
    - Name(FirstName, MiddleName, LastName).
    - Composition may form a hierarchy where some components are themselves composite.

Types of Attributes (2)

- **Multi-valued**
  - An entity may have multiple values for that attribute. For example:
    - Color of a CAR
    - PreviousDegrees of a STUDENT.
    - Denoted as {Color} or {PreviousDegrees}.

In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare!

Composite multi-valued attribute example

- For example, PreviousDegrees of a STUDENT can be a composite multi-valued attribute denoted as {PreviousDegrees (College, Year, Degree, Field)}
  - There can be multiple values of PreviousDegrees
  - Each value has 4 component attributes :
    - College, Year, Degree, Field

Example of a composite attribute

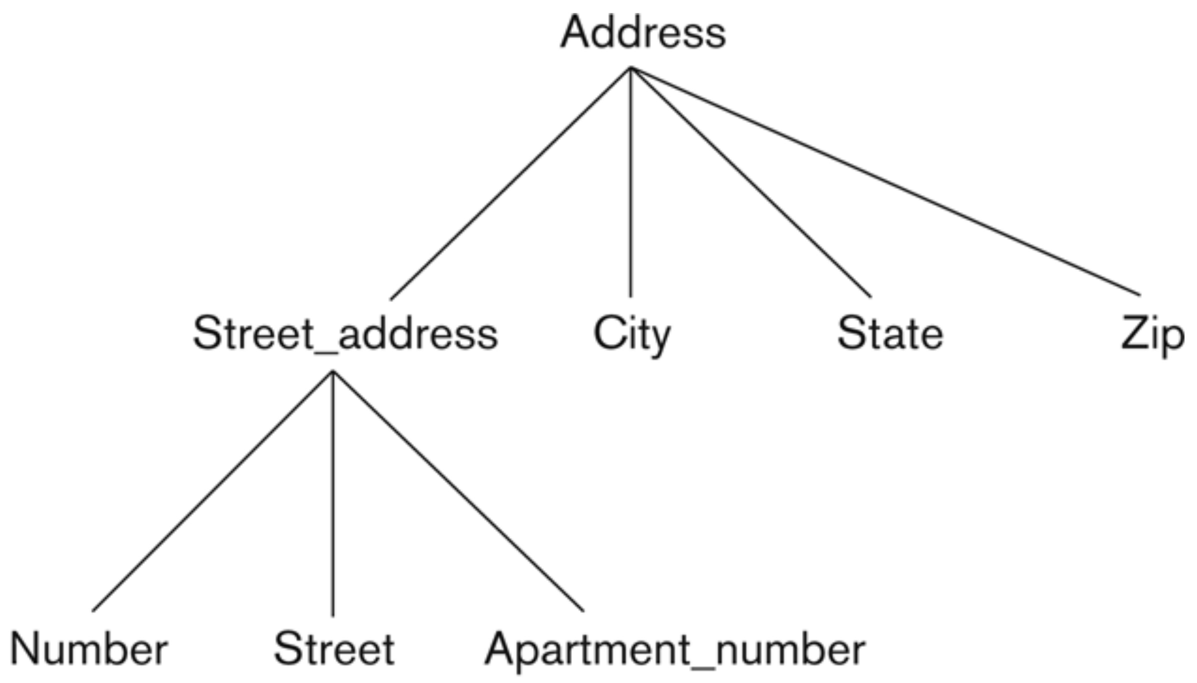
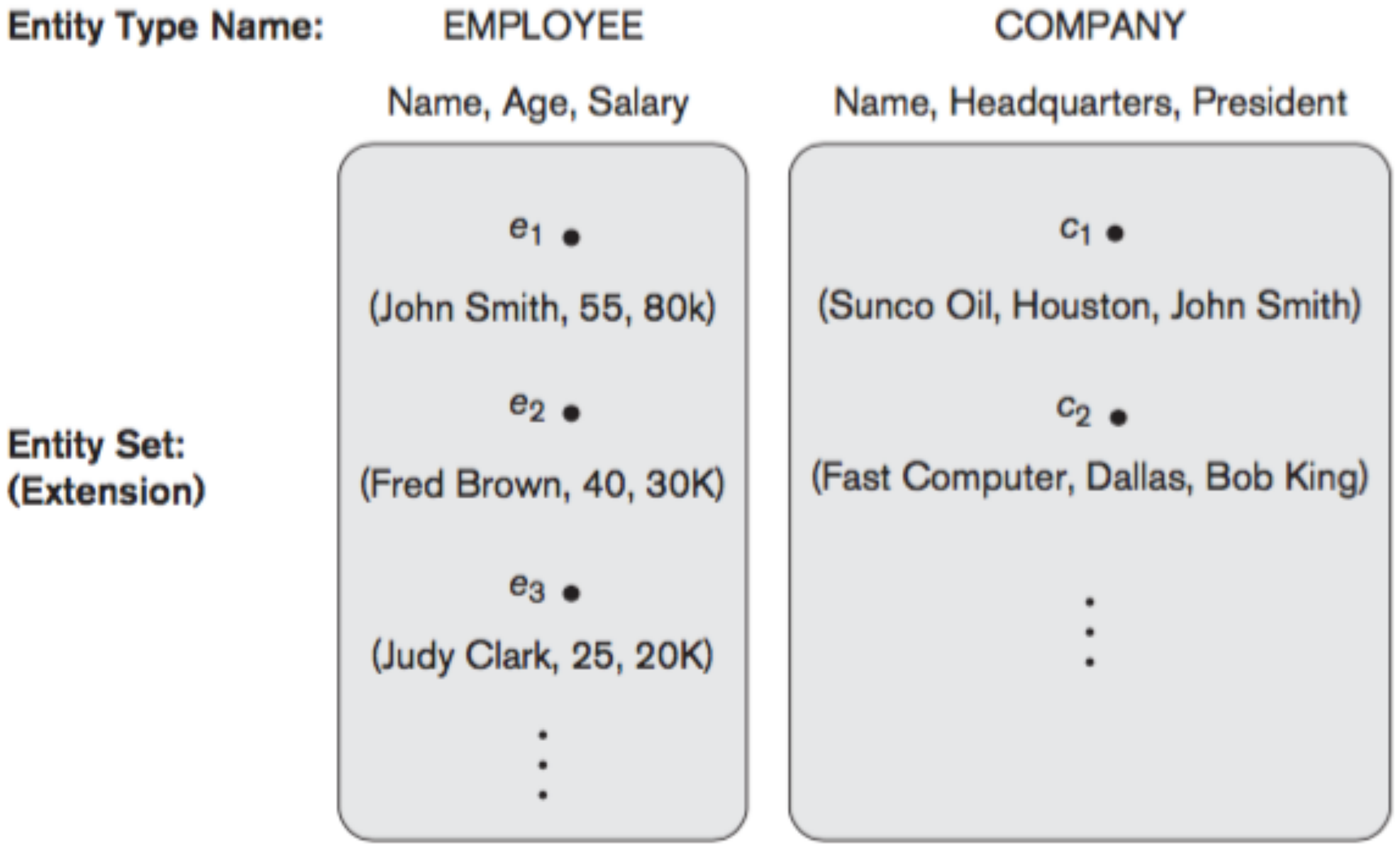


Figure 3.4
A hierarchy of composite attributes.

Entity Types and Key Attributes (1)

- Entities with the same basic attributes are grouped or typed into **an entity type**.
  - For example, the entity type EMPLOYEE and PROJECT.
- An attribute of an entity type for which each entity must have a unique value is called a **key attribute** of the entity type.
  - For example, SSN of EMPLOYEE.

Entity Types and Key Attributes (1)



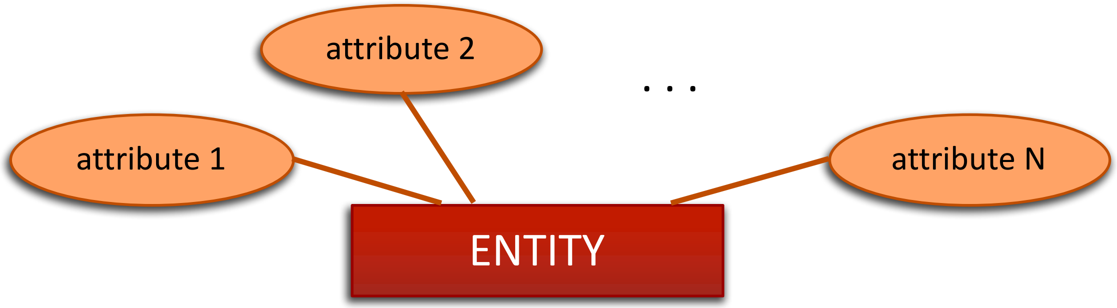
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

Entity Types and Key Attributes (2)

- A key attribute may be composite.
  - VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
  - The CAR entity type may have two keys:
    - VehicleIdentificationNumber (popularly called VIN)
    - VehicleTagNumber (Number, State), aka license plate number.
- Each key is <u>underlined</u>

Displaying an Entity type

- In ER diagrams, an entity type is displayed in a *rectangular box*
- Attributes are displayed in *ovals*
  - Each attribute is connected to its entity type
  - Components of a composite attribute are connected to the oval representing the composite attribute
  - Each key attribute is underlined
  - Multivalued attributes displayed in double ovals



Entity Type CAR with two keys and a corresponding Entity Set

Example

(a)

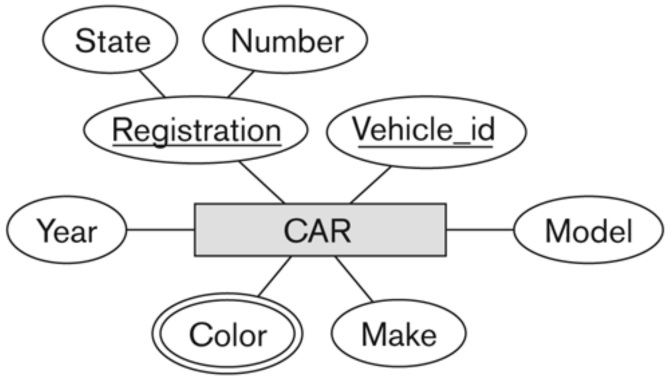


Figure 3.7
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(b)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

entity set

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

entity 1

entity 2

entity 3

Entity set

- Each entity type will have a collection of entities stored in the database
  - Called the **entity set**
  - Previous slide shows three CAR entity instances in the entity set for CAR
  - Same name (CAR) used to refer to both the entity type and the entity set

Entity set is the current state of the entities of that type that are stored in the database

Initial Design of Entity Types for the COMPANY Database Schema (1)

The company is organized into ***DEPARTMENT***s. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department may have several locations.

Each department controls a number of ***PROJECT***s. Each project has a unique name, unique number and is located at a single location.

We store each ***EMPLOYEE***'s social security number, address, salary, sex, and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.

Each employee may have a number of ***DEPENDENT***s. For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.

Initial Design of Entity Types for the COMPANY Database Schema (2)

- Based on the requirements, we can identify four initial entity types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT
- Their initial design is shown on the following slide
- The initial attributes shown are derived from the requirements description

Initial Design of Entity Types: EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

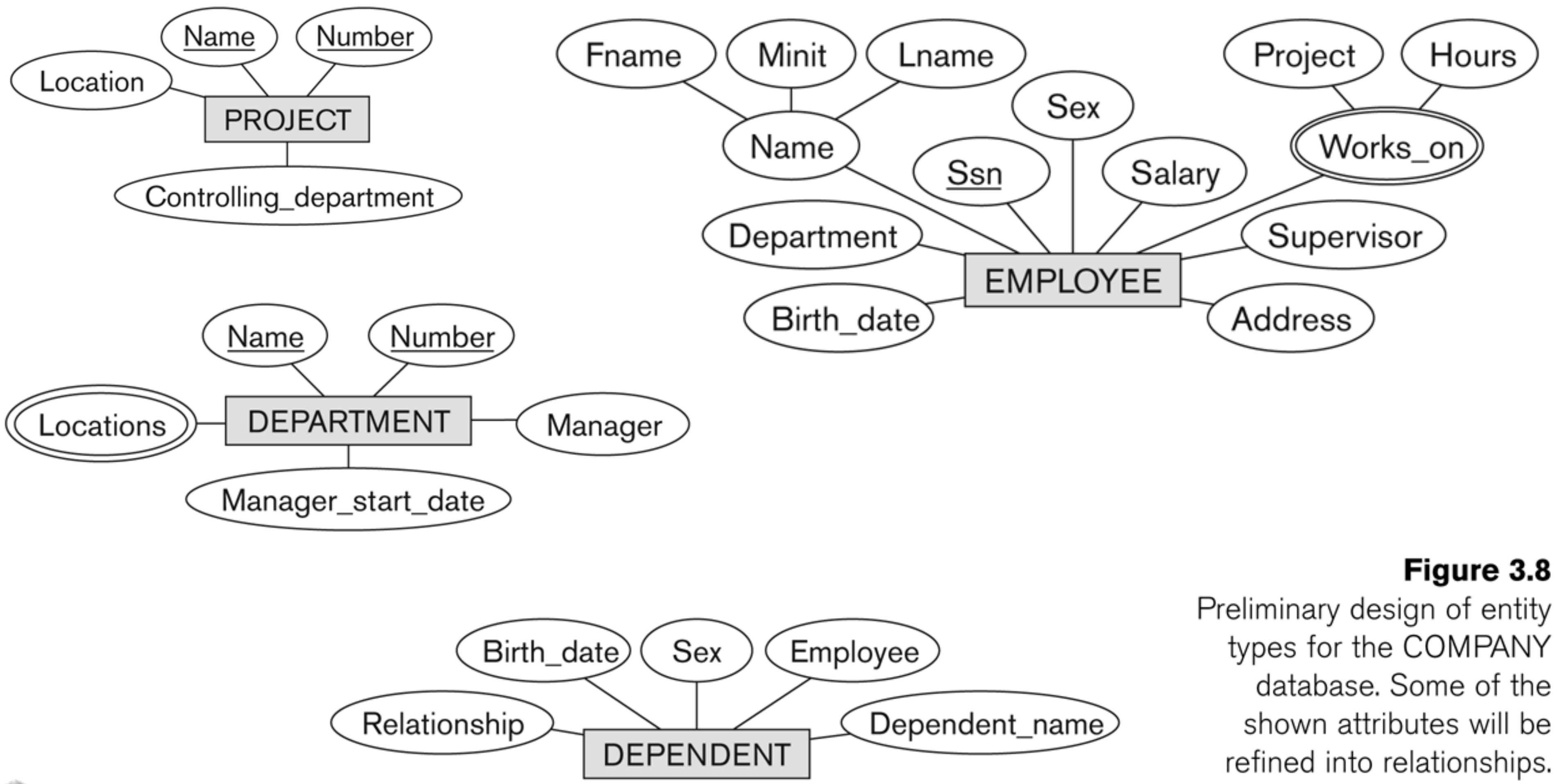


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

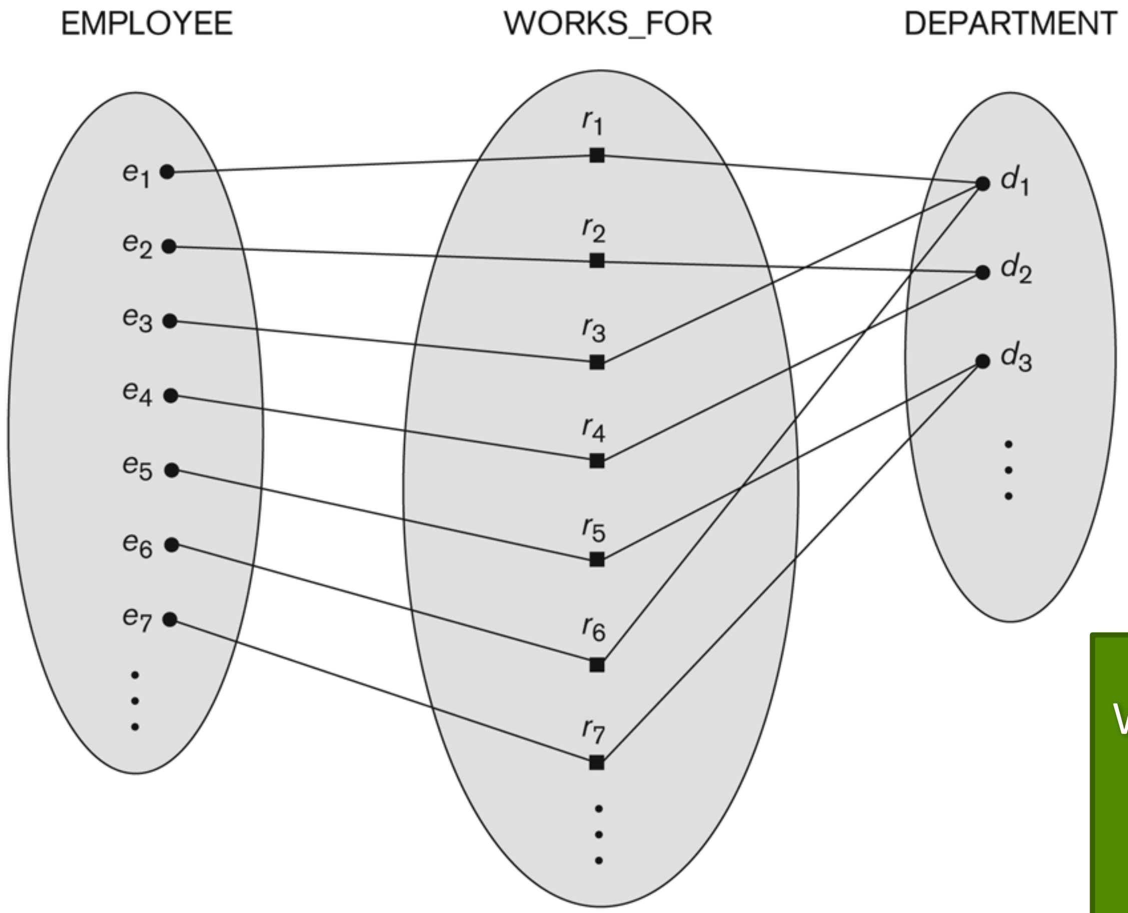
Refining the initial design by introducing **relationships**

- The initial design is typically not complete
- Some aspects in the requirements will be represented as **relationships**
- ER model has three main concepts:
  - Entities (and their entity types and entity sets)
  - Attributes (simple, composite, multivalued)
  - Relationships (and their relationship types and relationship sets)

Relationships and Relationship Types

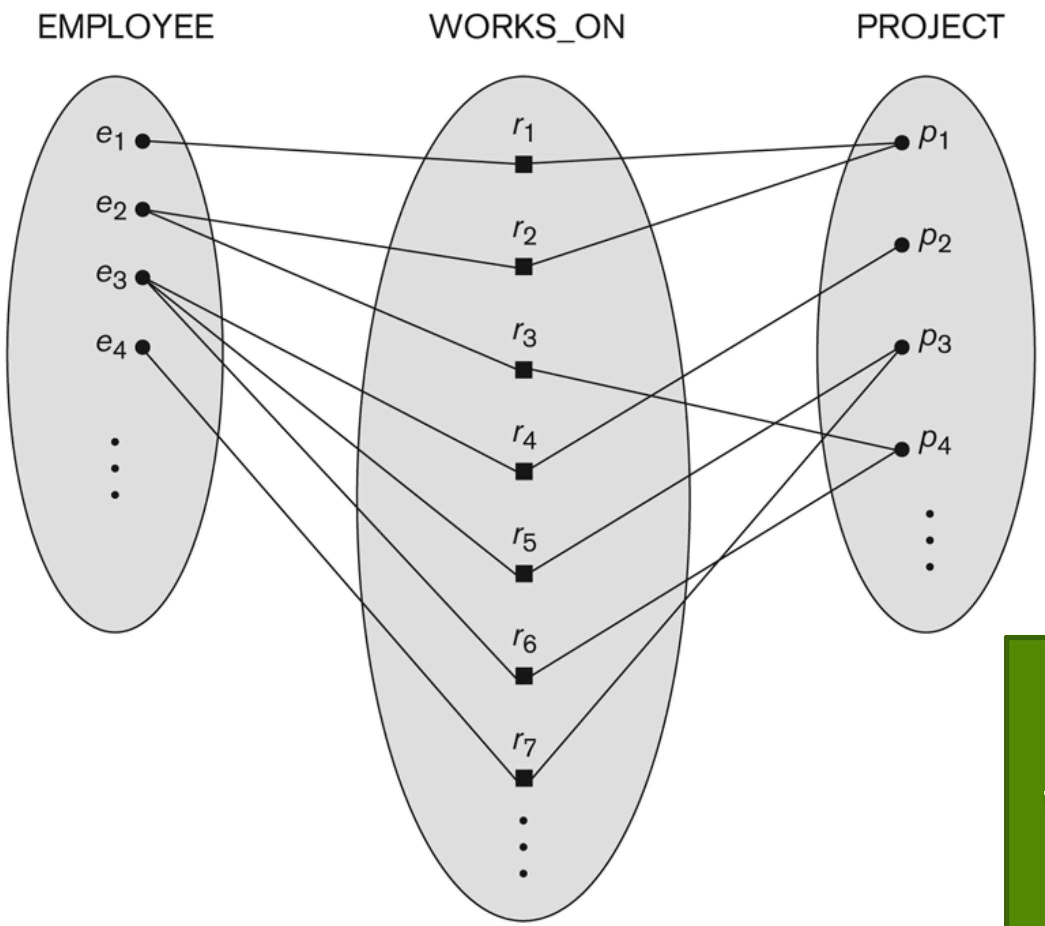
- A **relationship** relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith works on the ProductX PROJECT, or EMPLOYEE Franklin Wong manages the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
  - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- **The degree** of a relationship type is the number of participating entity types.
  - Both MANAGES and WORKS_ON are binary relationships.

Relationships and Relationship Types Example (1)



Some instances in the WORKS_FOR relationship set, which represent a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT

Relationships and Relationship Types Example (2)



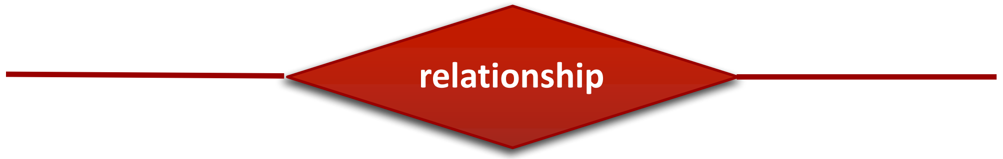
Some instances in the WORKS_ON relationship set between EMPLOYEE and PROJECT

Relationship type vs. relationship set (1)

- **Relationship Type**:
  - Is the schema description of a relationship
  - Identifies the relationship name and the participating entity types
  - Also identifies certain relationship constraints
- **Relationship Set**:
  - The current set of relationship instances represented in the database
  - The current *state* of a relationship type

Relationship type vs. relationship set (2)

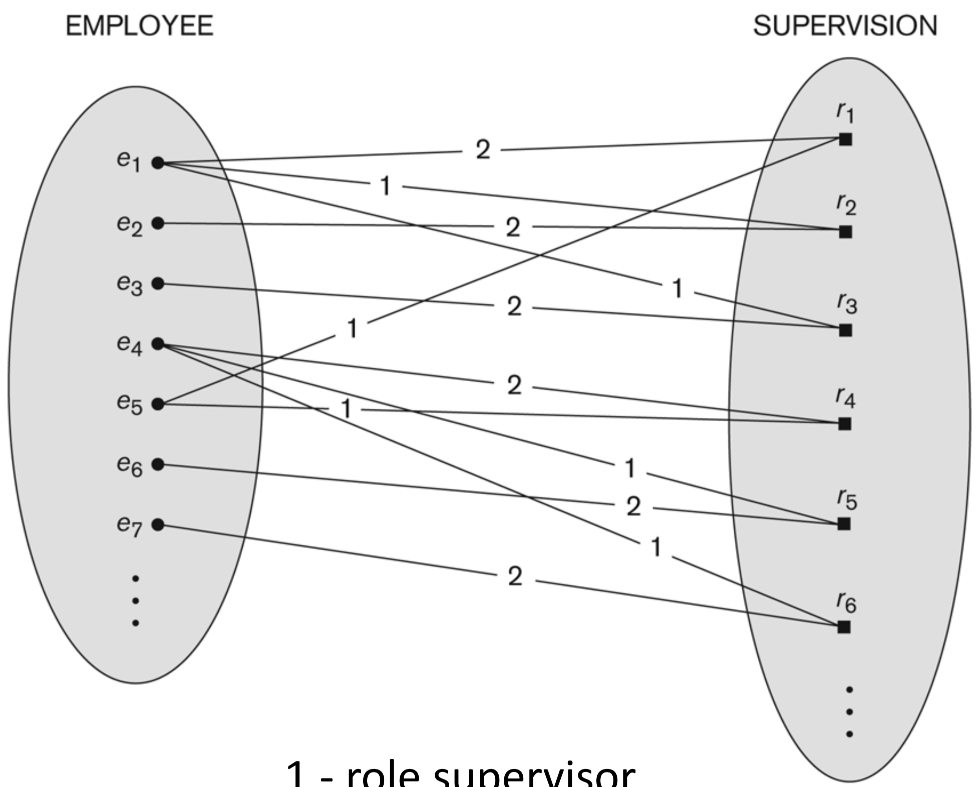
- Previous figures displayed the relationship sets
  - Each instance in the set relates individual participating entities – **one from each participating entity type**
- In ER diagrams, we represent the *relationship type* as follows:
  - Diamond-shaped box is used to display a relationship type
  - Connected to the participating entity types via straight lines



Recursive Relationship Type

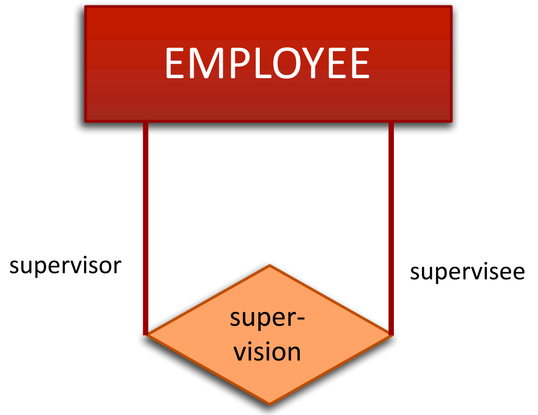
- Same entity type participates more than once in a relationship type
  - Both participations are same entity type in **different roles**.
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role

A Recursive Relationship Supervision`



1 - role supervisor
2 - role supervisee

- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in *supervisor* role
  - One employee in *supervisee* role



Notation for Constraints on Relationships

- **Cardinality ratio** (of a binary relationship):
  - Shown by placing appropriate numbers on the relationship edges(1:1, 1:N, N:1, or M:N)
- **Participation constraint** (on each participating entity type):
  - **Total** (called existence dependency) shown by double line
  - **Partial** shown by single line

NOTE:
These are easy to specify for Binary Relationship Types

Many-to-one (N:1) Relationship

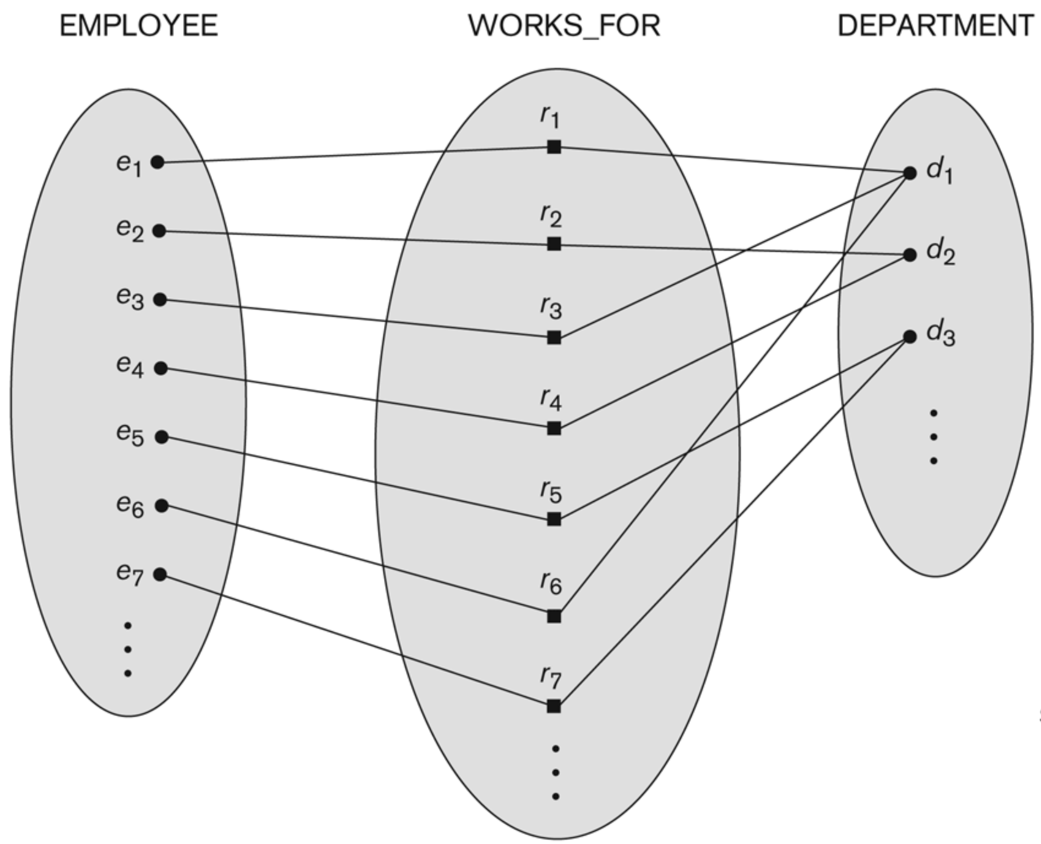
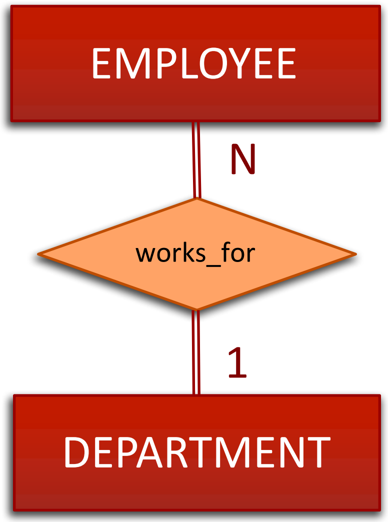
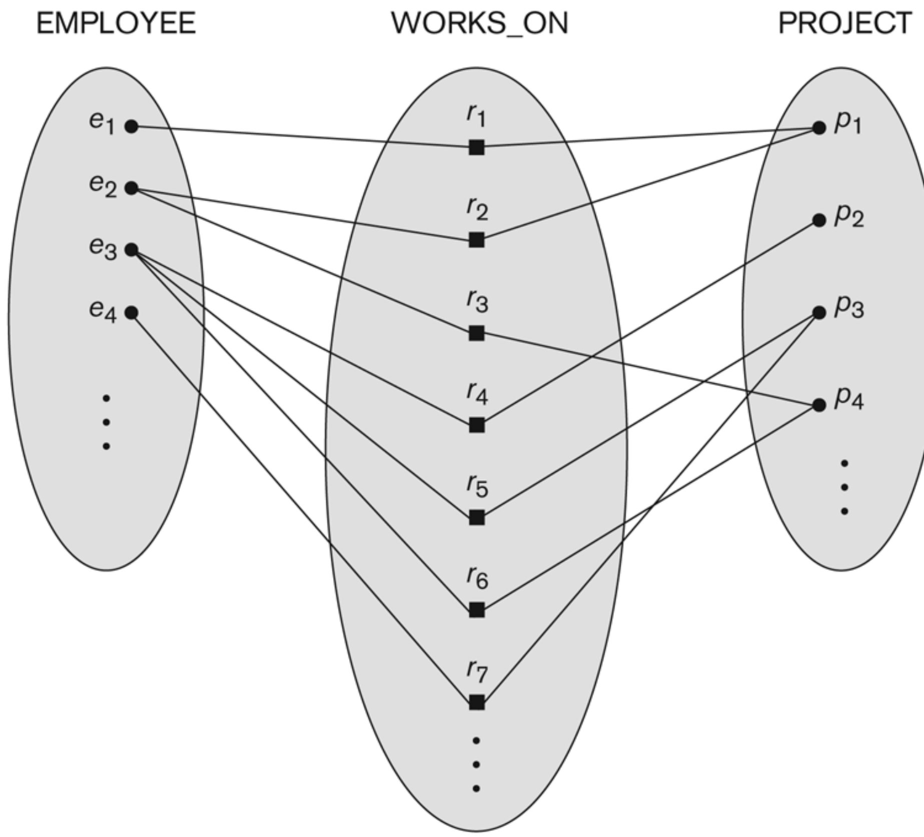


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.



Many-to-many (M:N) Relationship



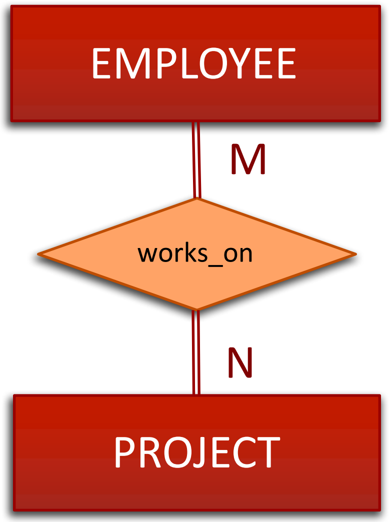


Figure 3.13
An M:N relationship, WORKS_ON.

Constraints on Relationships

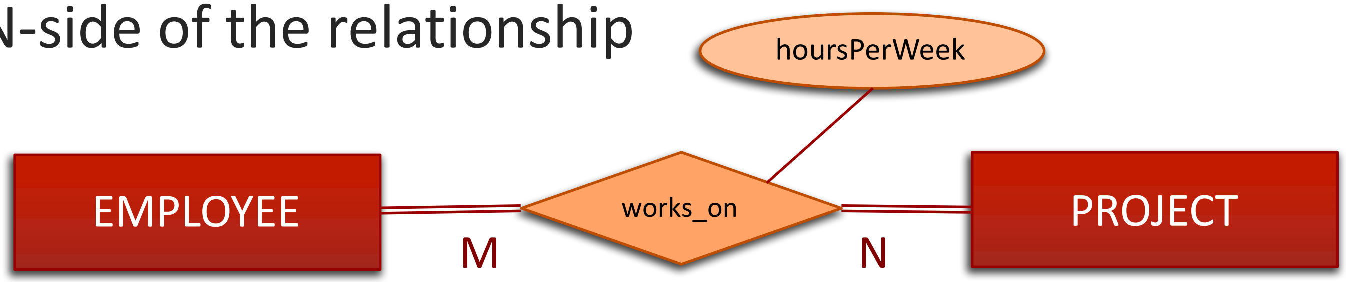
- Constraints on Relationship Types
  - (Also known as ratio constraints)
  - Cardinality Ratio (specifies *maximum* participation)
    - One-to-one (1:1)
    - One-to-many (1:N) or Many-to-one (N:1)
    - Many-to-many (M:N)
  - Existence Dependency Constraint (specifies *minimum* participation) (also called participation constraint)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory participation, existence-dependent)

Examples for constraints on relationships

- Example for total participation (existence dependency):
  - If company policy imposes that *every* employee must work for some department, then the EMPLOYEE entity can exist only if it participates in at least one instance of the WORKS_FOR relationship
- Example for partial participation (existence not dependent):
  - We don’t expect every employee to manage some department. Therefore, EMPLOYEE has partial participation in the MANAGES relationship

Attributes of Relationship types

- Some relationship types can have attributes
  - For example, HoursPerWeek of WORKS_ON
  - Its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.
    - A value of HoursPerWeek depends on a particular (employee, project) combination
- Most relationship attributes are used with M:N relationships
  - In 1:N relationships, they are transferred to the entity type on the N-side of the relationship

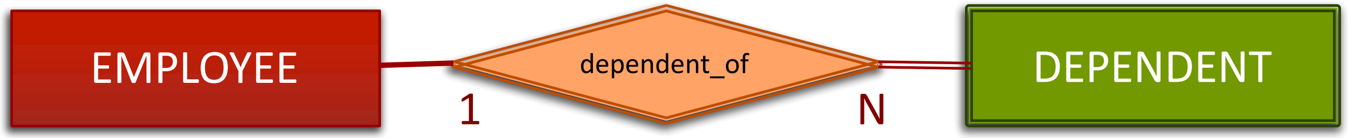


Weak Entity Types

- An entity that does not have a key attribute
- A weak entity must participate in **an identifying relationship type** with an **owner or identifying entity type**
- Entities are identified by the combination of:
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying entity type
- In the ER diagram represented with **a double rectangular box** connected with an identifying relationship type with the identifying entity type (represented with **a double diamond-shaped box**)

Weak Entity Type Example

- A DEPENDENT entity is identified by the dependent's first name, and the specific EMPLOYEE with whom the dependent is related
  - Name of DEPENDENT is the partial key
  - DEPENDENT is a weak entity type
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF



Refining the COMPANY database schema by introducing relationships (1)

The company is organized into departments. Each department has a name, number and an employee who ***MANAGES*** the department. We keep track of the start date of the department manager. A department may have several locations.

Each department ***CONTROLS*** a number of projects. Each project has a unique name, unique number and is located at a single location.

We store each employee’s social security number, address, salary, sex, and birthdate. Each employee ***WORKS FOR*** one department but may ***WORK ON*** several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the ***DIRECT SUPERVISOR*** of each employee.

Each employee may ***HAVE A NUMBER OF DEPENDANTS***. For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.

Refining the COMPANY database schema by introducing relationships (2)

- By examining the requirements, six relationship types are identified
- All are *binary* relationships (degree 2)
- Listed below with their participating entity types:
  - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
  - MANAGES (also between EMPLOYEE, DEPARTMENT)
  - CONTROLS (between DEPARTMENT, PROJECT)
  - WORKS_ON (between EMPLOYEE, PROJECT)
  - SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
  - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

ER Diagram for the COMPANY Example

Example

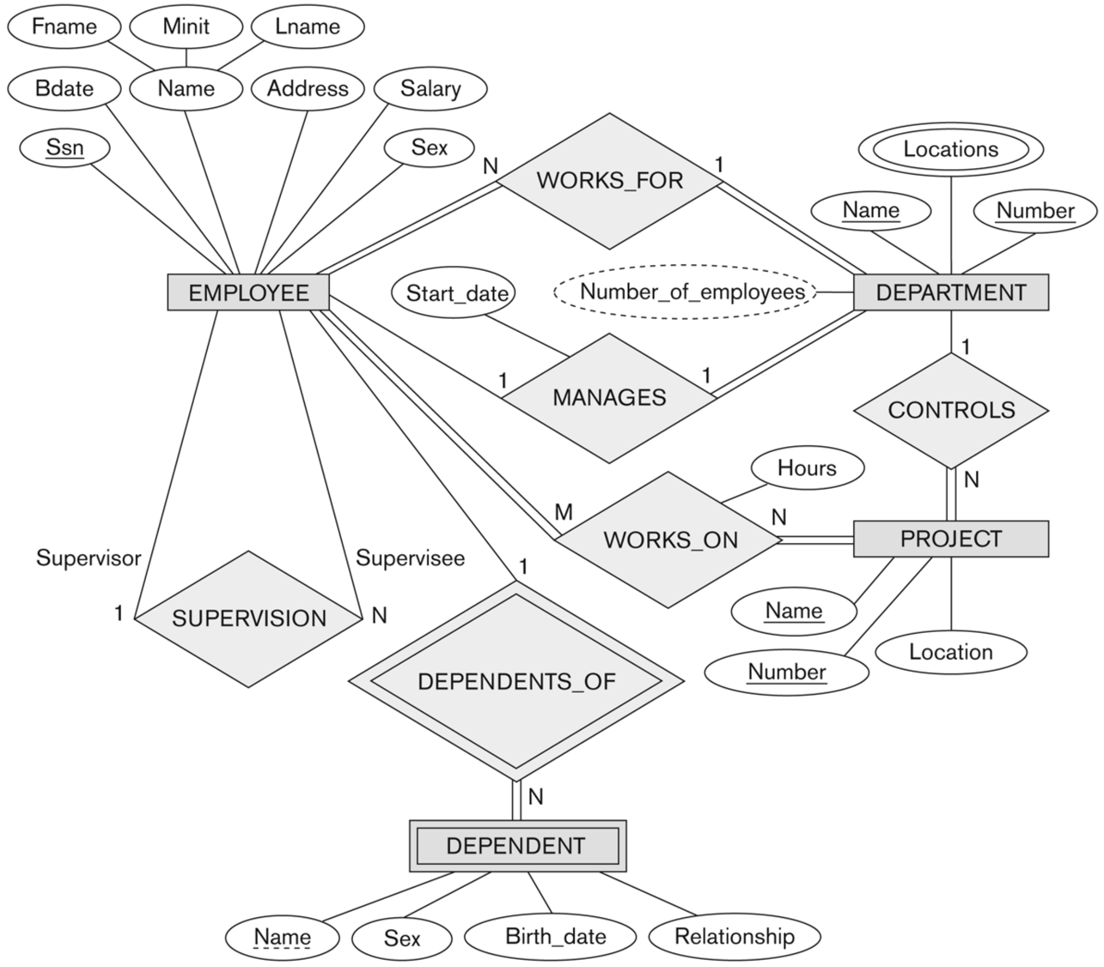


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

entity types are:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

relationships types are:
WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF

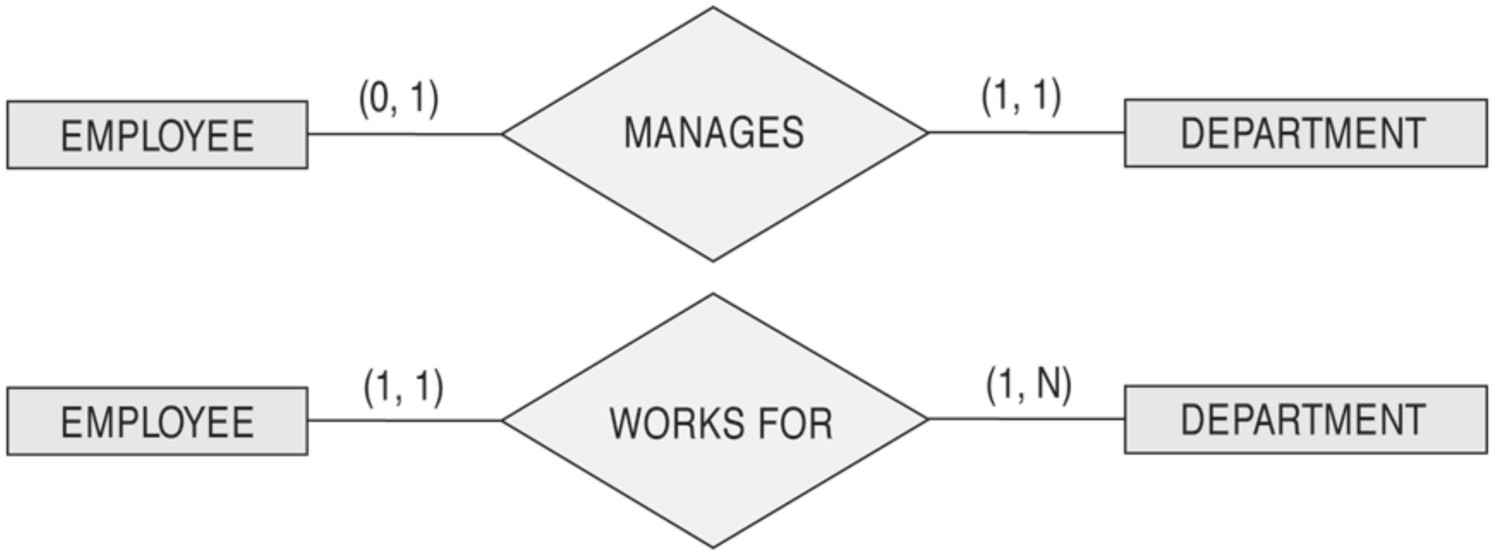
Discussion on Relationship Types

- In the refined design, some attributes from the initial entity types are refined into relationships:
  - Manager of DEPARTMENT -> MANAGES
  - Works_on of EMPLOYEE -> WORKS_ON
  - Department of EMPLOYEE -> WORKS_FOR
  - etc
- In general, more than one relationship type can exist between the same participating entity types
  - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
  - Different meanings and different relationship instances.

Alternative (min, max) notation for relationship structural constraints

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints
- Examples:
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

The (min,max) notation for relationship constraints

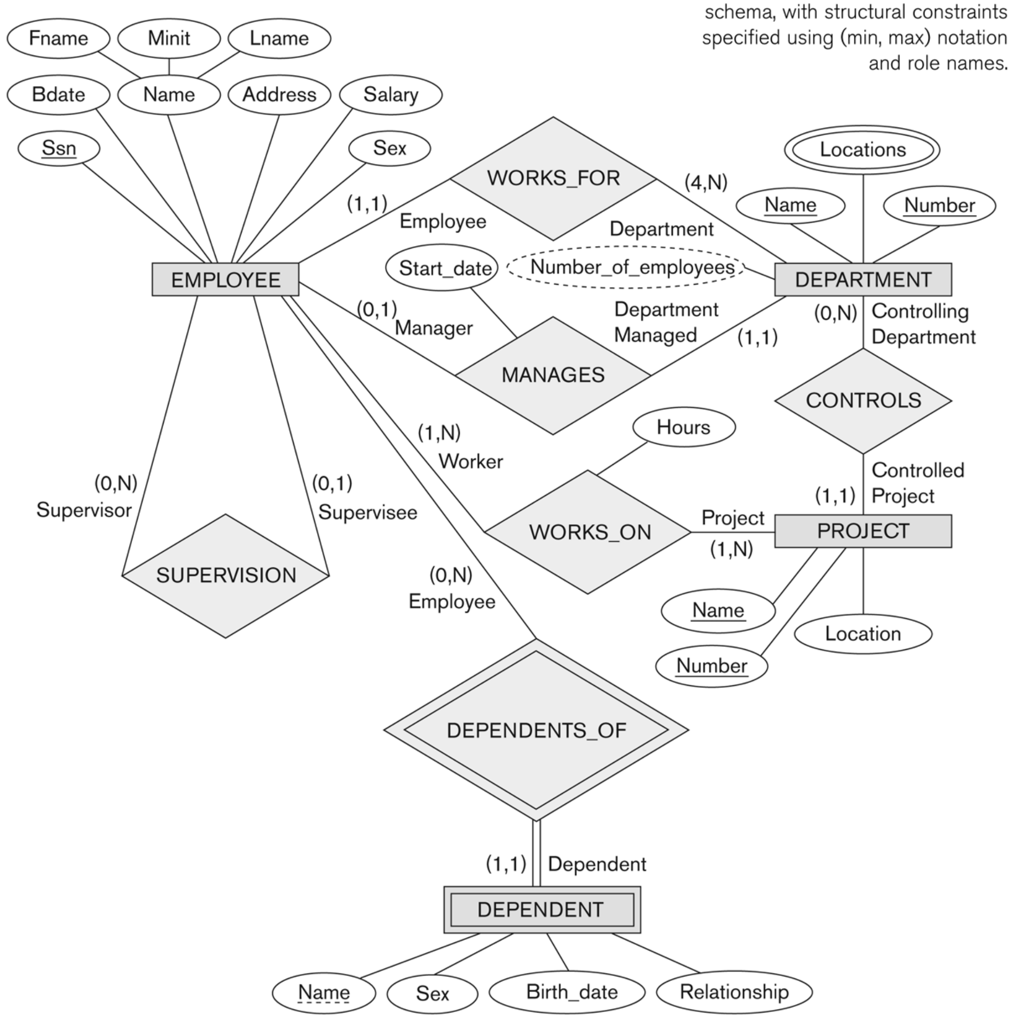


NOTE:

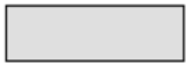
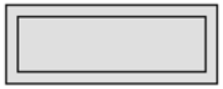
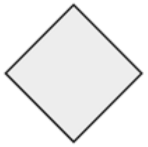
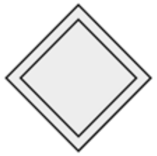
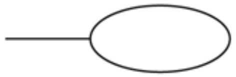
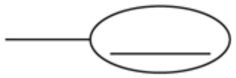
Be careful when you read the min,max constraints. The numbers next to the entity type refer to the entity type on the other side of the relationship. In other notations, like for example UML (upcoming slides), these numbers are used in an opposing logic (number are written down on the other side of the relationship).

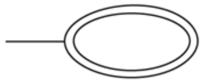
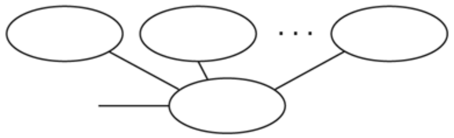
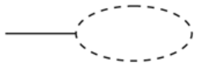
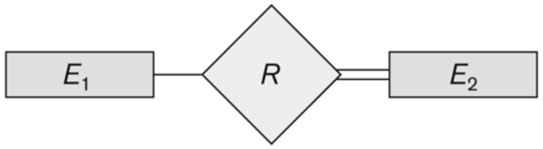
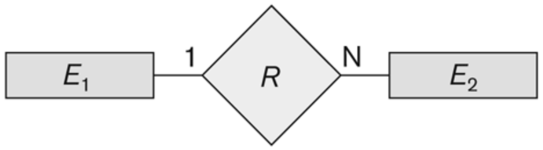
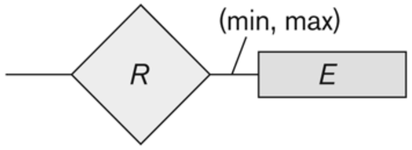
COMPANY ER Schema Diagram using (min, max) notation

Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.



Summary of notation for ER diagrams

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute

Symbol	Meaning
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for E_1:E_2 in R
	Structural Constraint (min, max) on Participation of E in R

Alternative diagrammatic notation

- ER diagrams is one popular example for displaying database schemas
- Many other notations exist in the literature and in various database design and modeling tools
  - Appendix A illustrates some of the alternative notations that have been used
  - UML class diagrams is representative of another way of displaying ER concepts that is used in several commercial design tools

Other alternative diagrammatic notations

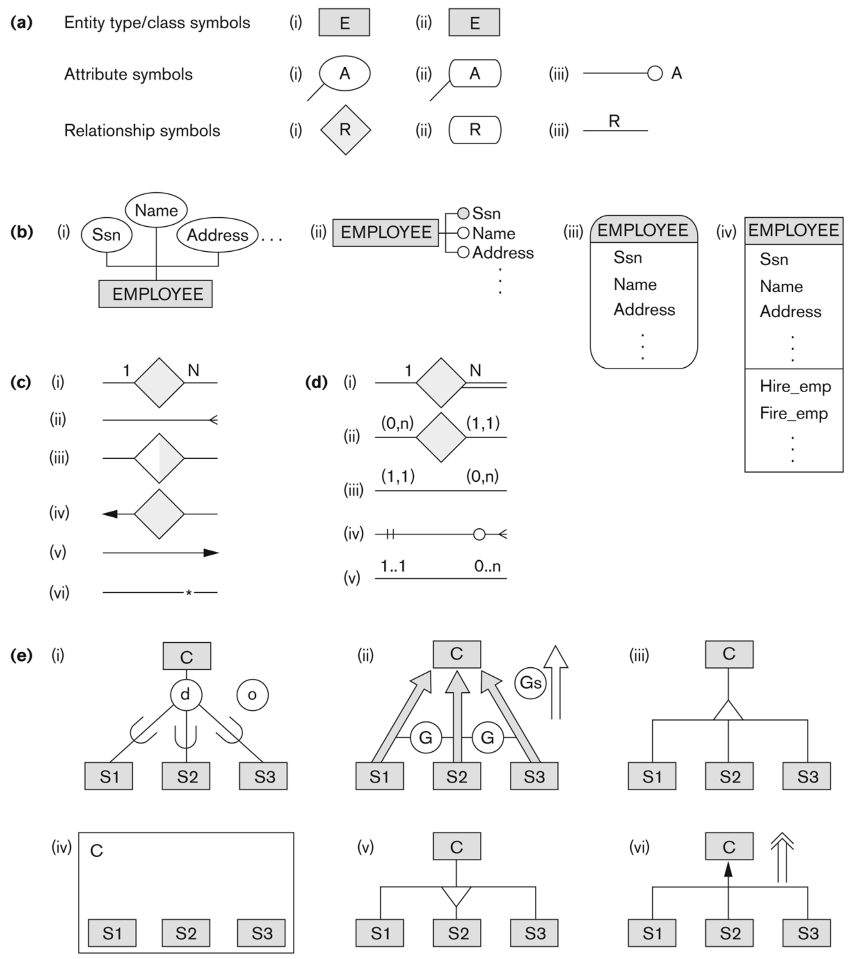


Figure A.1
Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

In Appendix A in the book

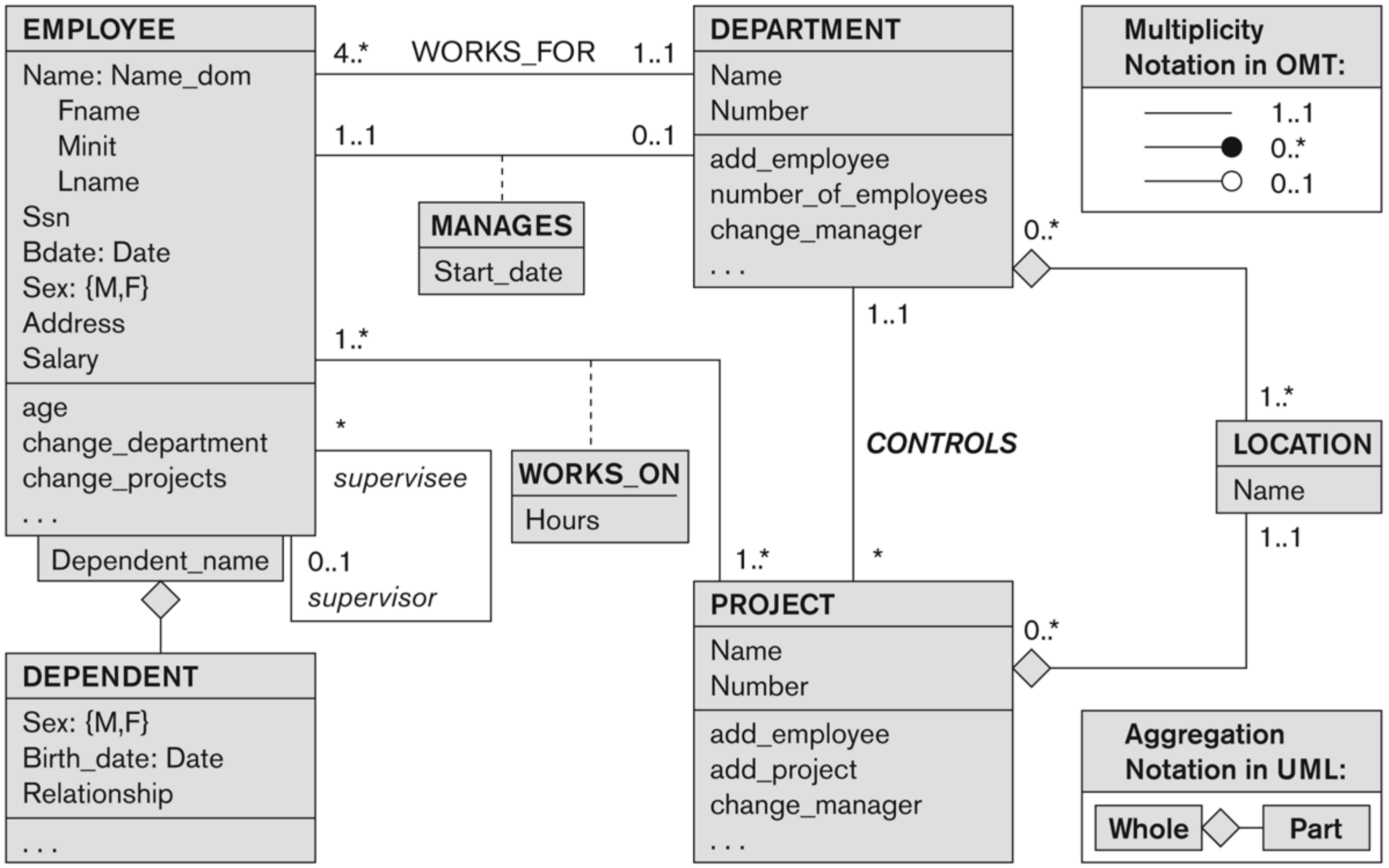
UML class diagrams

- Represent classes (similar to entity types) as large rounded boxes with three sections:
  - Top section includes entity type (class) name
  - Second section includes attributes
  - Third section includes class operations (operations are not in basic ER model)
- Relationships (called associations) represented as lines connecting the classes
  - Other UML terminology also differs from ER terminology
- Used in database design and object-oriented software design
- UML has many other types of diagrams for software design

UML class diagram for COMPANY database schema

Example

Figure 3.16
The COMPANY conceptual schema in UML class diagram notation.



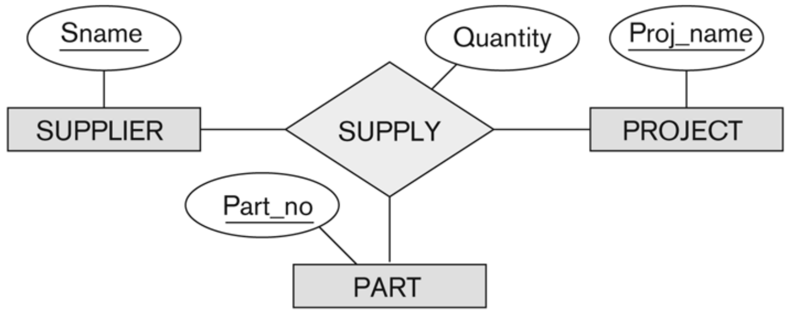
Relationships of Higher Degree

- Types of relationships according to their degree:
  - degree 2 - binary
  - degree 3 - ternary
  - degree n - n-ary
- In general, an n-ary relationship is not equivalent to n binary relationships
- Constraints are harder to specify for higher-degree relationships (n > 2) than for binary relationships

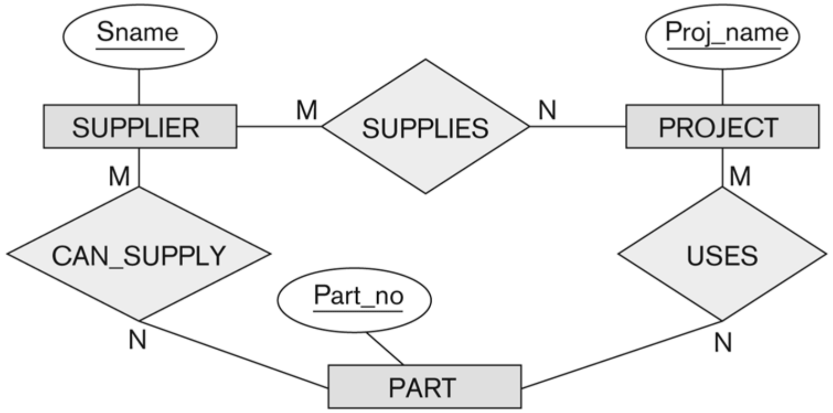
Discussion of n-ary relationships (n > 2) (1)

- In general, 3 binary relationships can represent different information than a single ternary relationship and if needed, all can be included in the schema design

(a)

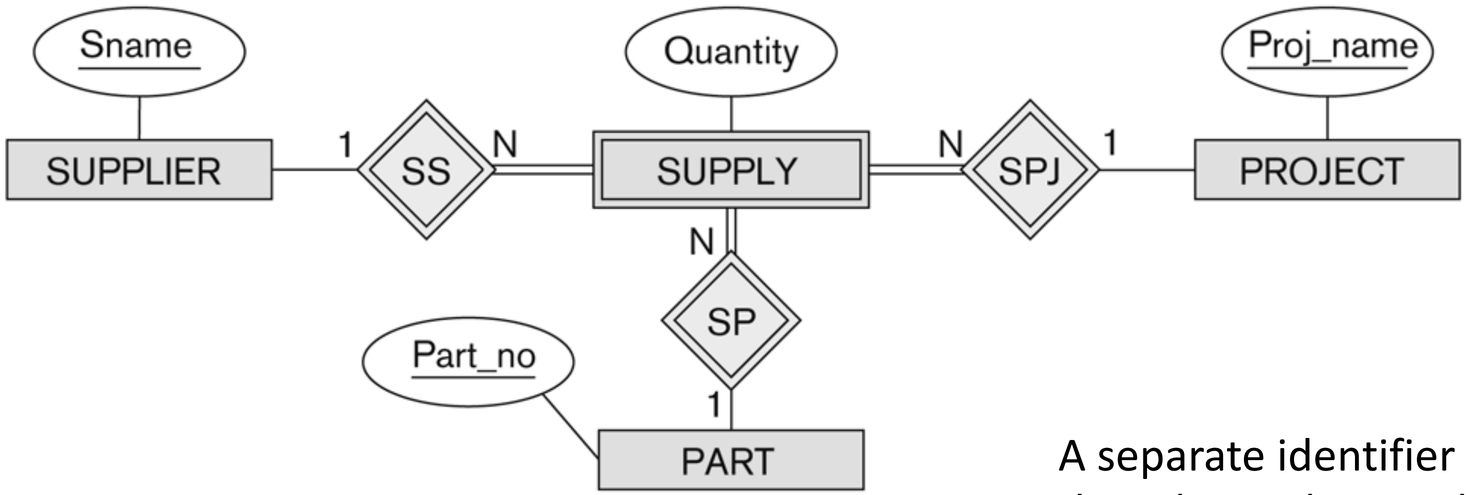


(b)



Discussion of n-ary relationships (n > 2) (2)

- In some cases, a ternary relationship can be represented as a weak entity if the data model allows a weak entity type to have multiple identifying relationships (and hence multiple owner entity types)

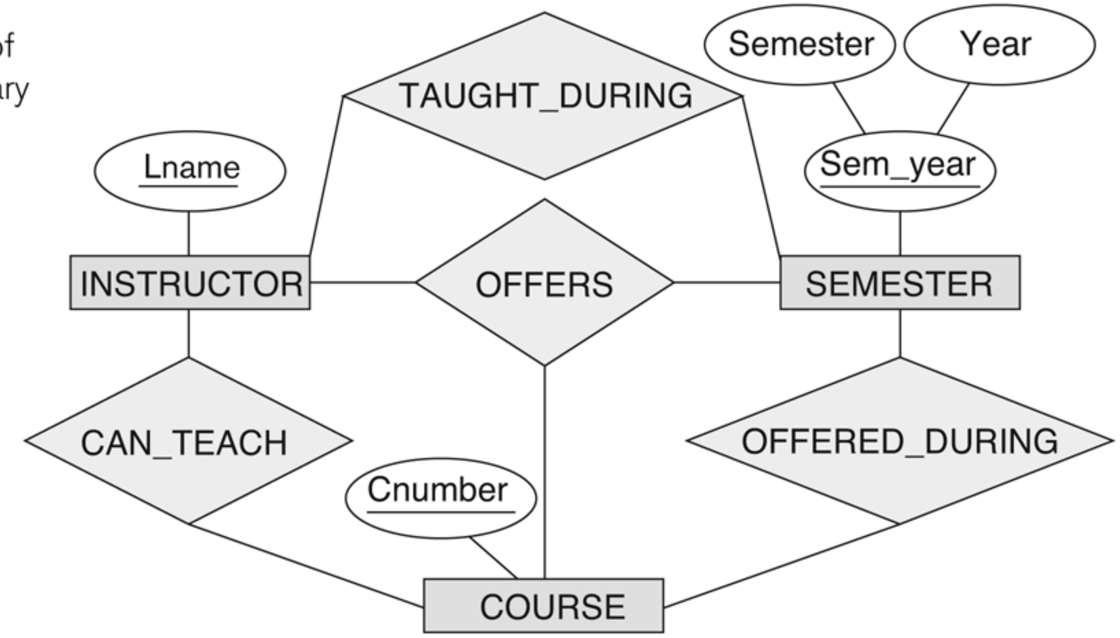


A separate identifier for the relationship can be introduced

Discussion of n-ary relationships (n > 2) (3)

- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant

Figure 3.18
Another example of ternary versus binary relationship types.



Discussion of n-ary relationships (n > 2) (4)

- These ternary and binary relationships represent different information, but certain constraints should hold among the relationships.

- For example, a relationship instance (*i, s, c) should not exist in OFFERS unless an instance (i, s) exists in* TAUGHT_DURING, an instance (*s, c) exists in OFFERED_DURING, and an instance (i, c) exists in CAN_TEACH.*

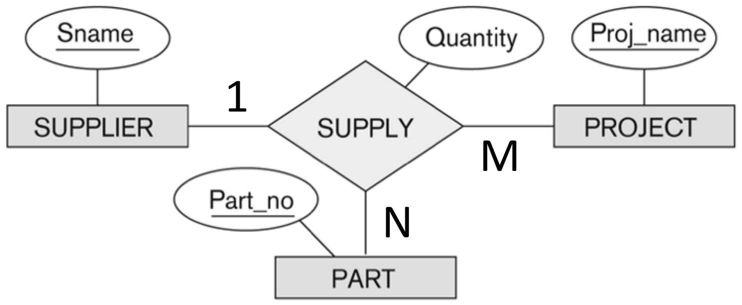
Discussion of n-ary relationships (n > 2) (5)

- Although in general three binary relationships *cannot replace a ternary relationship,* they may do so under certain *additional constraints.*

- In our example, if the CAN_TEACH relationship is 1:1 (an instructor can teach one course, and a course can be taught by only one instructor), then the ternary relationship OFFERS can be left out because it can be inferred from the three binary relationships CAN_TEACH, TAUGHT_DURING, and OFFERED_DURING.

Displaying constraints on higher-degree relationships

- The (min, max) constraints can be displayed on the edges – however, they do not fully describe the constraints
- Displaying a 1, M, or N indicates additional constraints
  - An M or N indicates no constraint
  - A 1 indicates that an entity can participate in at most one relationship instance *that has a particular combination of the other participating entities*



For example, if only one SUPPLIER, *supplies* a certain PART for a certain PROJECT

Enhanced Entity Relationship (EER) model

- EER Model Concepts
  - Includes all modeling concepts of basic ER
  - Additional concepts:
    - subclasses/superclasses
    - specialization/generalization
    - categories (UNION types)
    - attribute and relationship inheritance
  - These are fundamental to conceptual modeling
- The additional EER concepts are used to model applications more completely and more accurately
  - EER includes some object-oriented concepts, such as inheritance

Subclasses and Superclasses (1)

- An entity type may have additional meaningful subgroupings of its entities
  - Example: EMPLOYEE may be further grouped into:
    - SECRETARY, ENGINEER, TECHNICIAN, …
      - Based on the EMPLOYEE’s Job
    - MANAGER
      - EMPLOYEEs who are managers
    - SALARIED_EMPLOYEE, HOURLY_EMPLOYEE
      - Based on the EMPLOYEE’s method of pay
- EER diagrams extend ER diagrams to represent these additional subgroupings, called ***subclasses*** or ***subtypes***

Subclasses and Superclasses (2)

Example

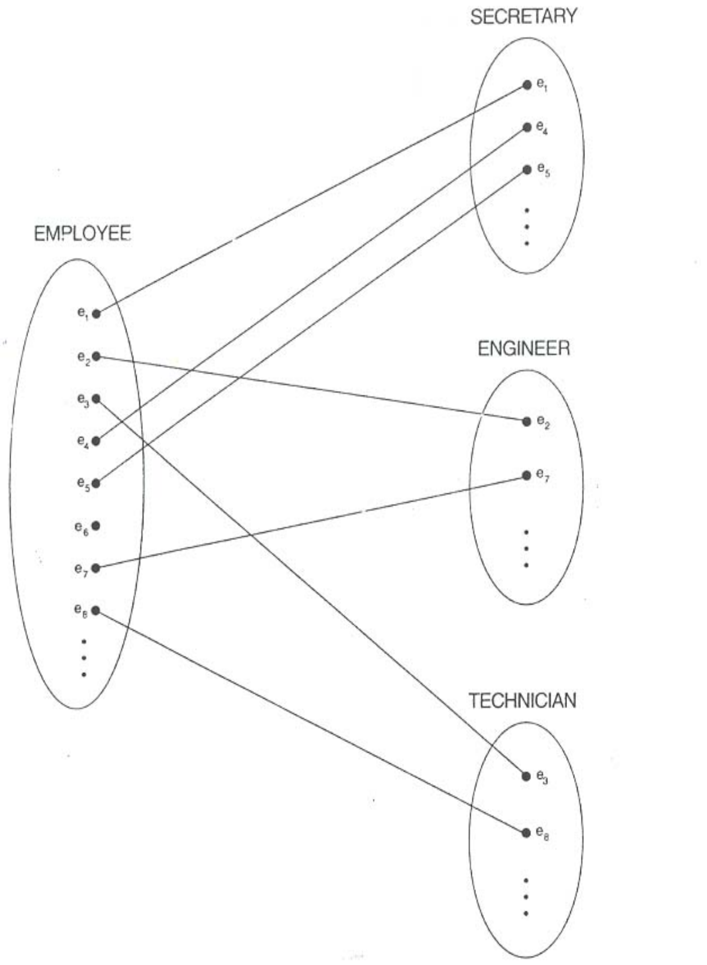
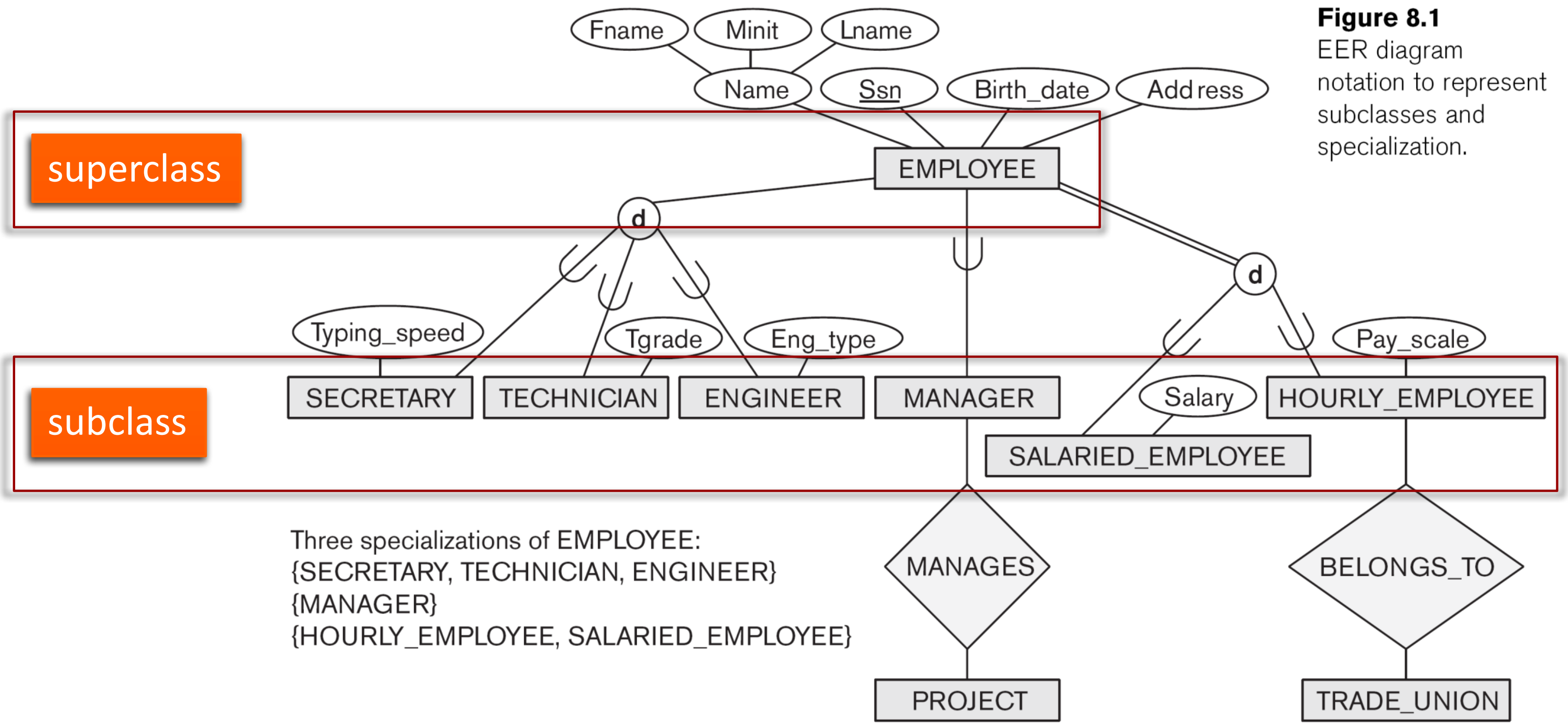


Figure 4.2 Some instances of the specialization of EMPLOYEE into the {SECRETARY, ENGINEER, TECHNICIAN} set of subclasses.

What is the difference between this relationship and the 1:1 relationship?

Subclasses and Superclasses (3)



Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}

Figure 8.1
EER diagram notation to represent subclasses and specialization.

[3]A class/subclass relationship is often called an **IS-A** (or **IS-AN**) **relationship** because of the way we refer to the concept. We say a SECRETARY *is an* EMPLOYEE, a TECHNICIAN *is an* EMPLOYEE, and so on.

Subclasses and Superclasses (4)

- Each of these subgroupings is a subset of EMPLOYEE entities
- Notations used in the EER diagram:
  - The symbol for the subclass-superclass relationship is the subset symbol ⊂
  - If there are multiple subclasses then a circle (**O**) is used to join them with the superclass
  - Each subclass can define its own (local) attributes
  - Each subclass can participate in its own relationships

Subclasses and Superclasses (5)

- These are also called **IS-A relationships**
  - SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, ….
- Note:
  - An entity that is member of a subclass represents the same real-world entity as some member of the superclass:
  - The subclass member is the same entity in a *distinct specific role*
  - An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass
  - A member of the superclass can be optionally included as a member of any number of its subclasses

Subclasses and Superclasses (6)

- Examples:
  - A salaried employee who is also an engineer belongs to the two subclasses:
    - ENGINEER, and
    - SALARIED_EMPLOYEE
  - A salaried employee who is also an engineering manager belongs to the three subclasses:
    - MANAGER,
    - ENGINEER, and
    - SALARIED_EMPLOYEE

Is it necessary that every entity in a superclass be a member of some subclass?

Generalization / Specialization

- **Generalization** is the process of defining a generalized entity type from a set of given entity types by identifying their common features

- **Specialization** is the process of defining a set of subclasses of an entity type (superclass of the specialization)

Generalization example

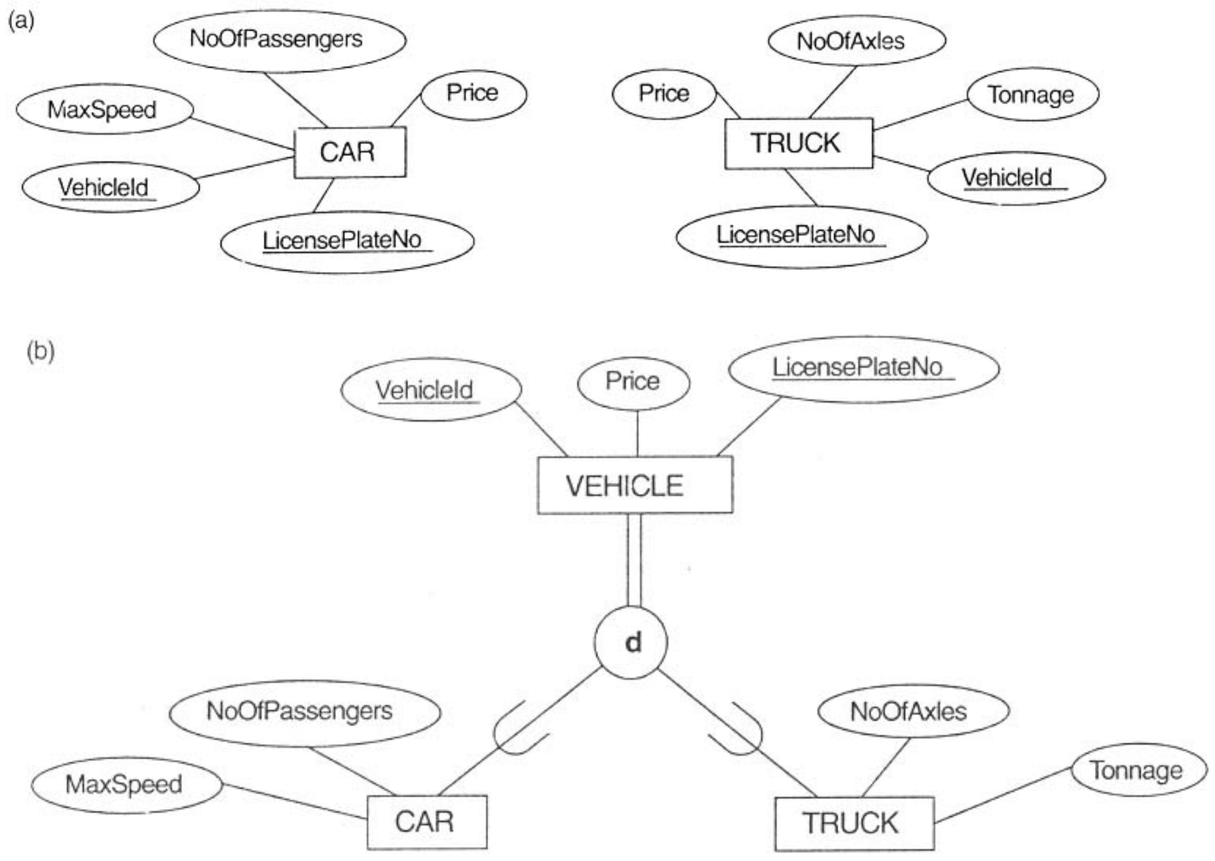


Figure 4.3 Examples of generalization. (a) Two entity types CAR and TRUCK. (b) Generalizing CAR and TRUCK into VEHICLE.

Constraints on Specialization and Generalization (1)

- Disjointness Constraint:
  - ***Disjoint*** :
    - Specifies that the subclasses of the specialization must be disjoint: an entity can be a member of at most one of the subclasses of the specialization
    - Specified by ***d*** in EER diagram
  - ***Overlapping*** :
    - If not disjoint, specialization is overlapping: that is the same entity may be a member of more than one subclass of the specialization
    - Specified by ***o*** in EER diagram

Constraints on Specialization and Generalization (2)

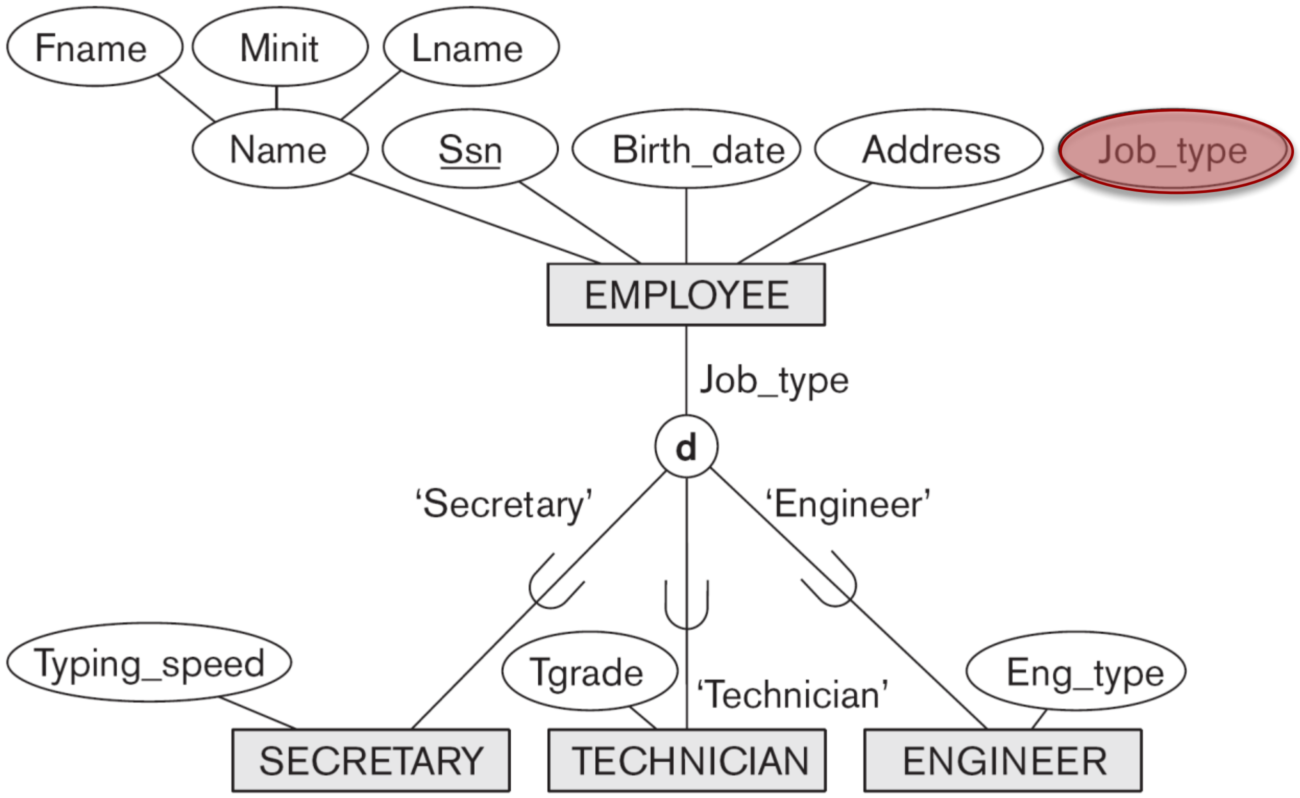
- Completeness Constraint:
  - ***Total*** specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
    - Shown in EER diagrams by a ***double line***
  - ***Partial*** allows an entity not to belong to any of the subclasses
    - Shown in EER diagrams by a single line

Constraints on Specialization and Generalization (3)

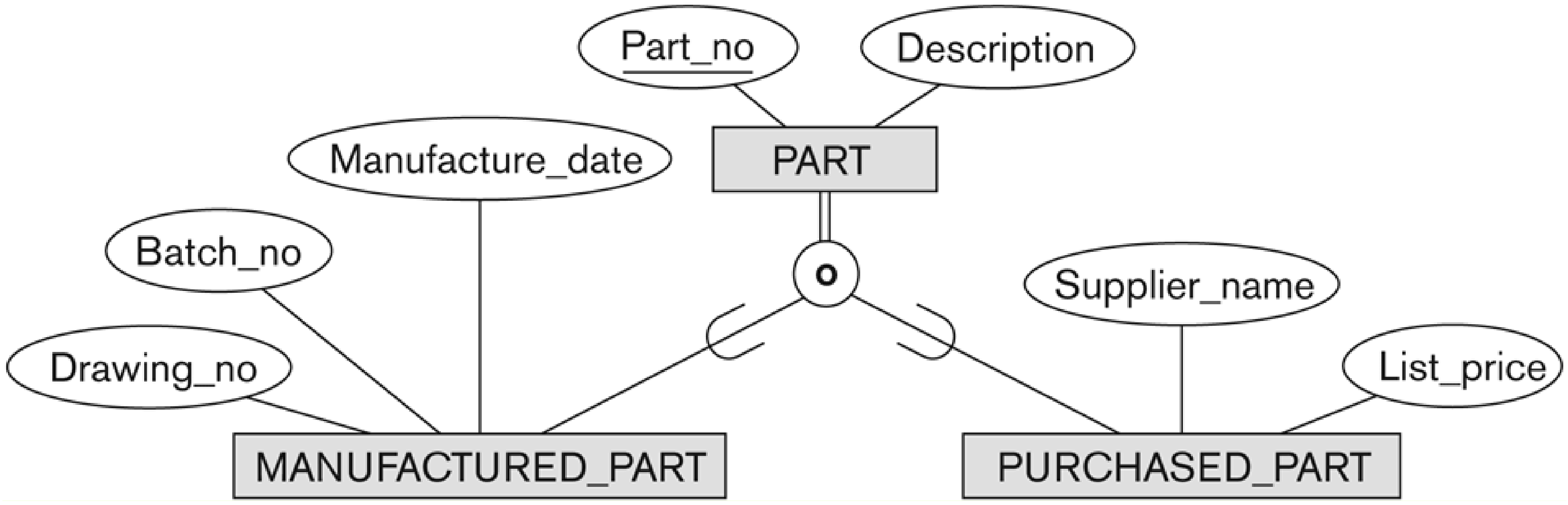
- Hence, we have four types of specialization/generalization:
  - Disjoint, total
  - Disjoint, partial
  - Overlapping, total
  - Overlapping, partial
- Note:
  - Generalization usually is total because the superclass is derived from the subclasses.

Example of disjoint partial Specialization

Figure 8.4
EER diagram notation for an attribute-defined specialization on Job_type.



Example of overlapping total Specialization



Specialization/Generalization Hierarchies, Lattices (1)

- A subclass may itself have further subclasses specified on it
  - forms a hierarchy or a lattice
- ***Hierarchy*** has a constraint that every subclass has only one superclass (called ***single inheritance***); this is basically a ***tree structure***
- In a ***lattice***, a subclass can be subclass of more than one superclass (called ***multiple inheritance***)

Specialization/Generalization Hierarchies, Lattices (2)

Example

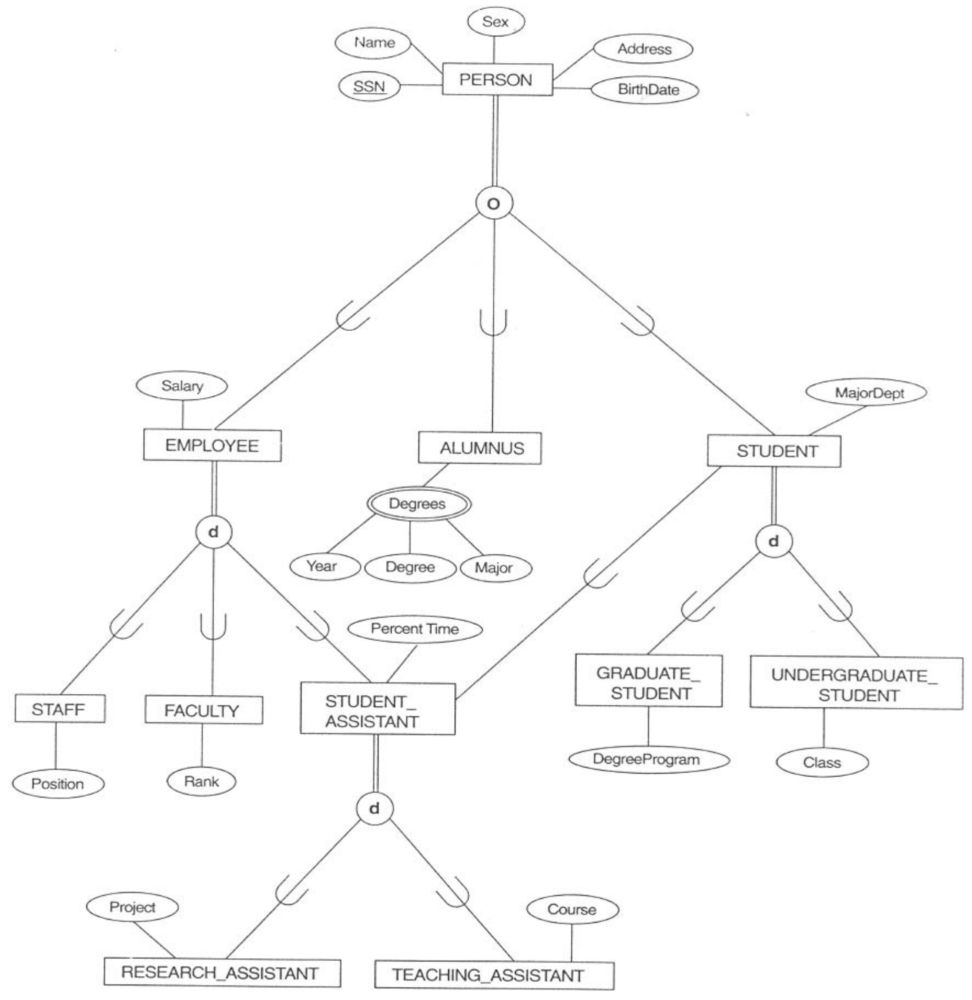


Figure 4.7 A specialization lattice (with multiple inheritance) for a UNIVERSITY database.

Homework:

Using your current knowledge of the organization and functioning of a faculty, draw the corresponding EER diagram

Specialization/Generalization Hierarchies, Lattices (2)

- Specialization - *top down* conceptual refinement process
  - Person
  - Employee, Alumnus, Student
  - …
- Generalization - *bottom up* conceptual synthesis process
  - Staff, Faculty, Res. Ass, Teach. Ass., Grad. Stud., Undergrad. Stud
  - Res.Ass + Tech. Ass. -> Student_Atistant…

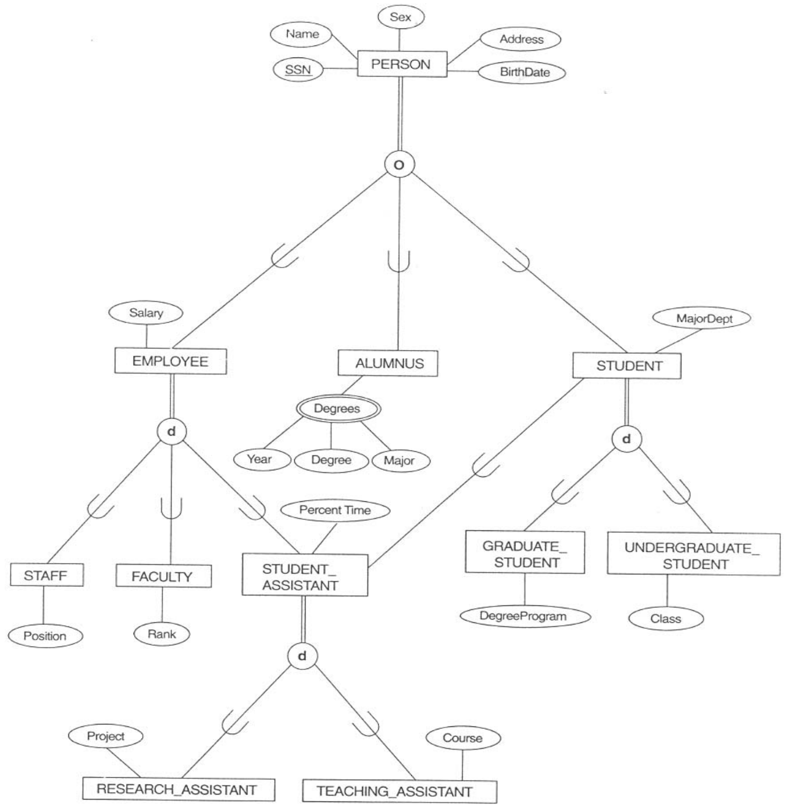


Figure 4.7 A specialization lattice (with multiple inheritance) for a UNIVERSITY database.

Categories (UNION TYPES) (1)

- All of the *superclass/subclass relationships* we have seen thus far have a single superclass
- A shared subclass is a subclass in:
  - *more than one* distinct superclass/subclass relationships
  - each relationships has a single superclass
  - shared subclass leads to multiple inheritance
- In some cases, we need to model a *single superclass/subclass relationship* with *more than one* superclass
- Superclasses can represent different entity types
- Such a subclass is called a category or UNION TYPE

Categories (UNION TYPES) (2)

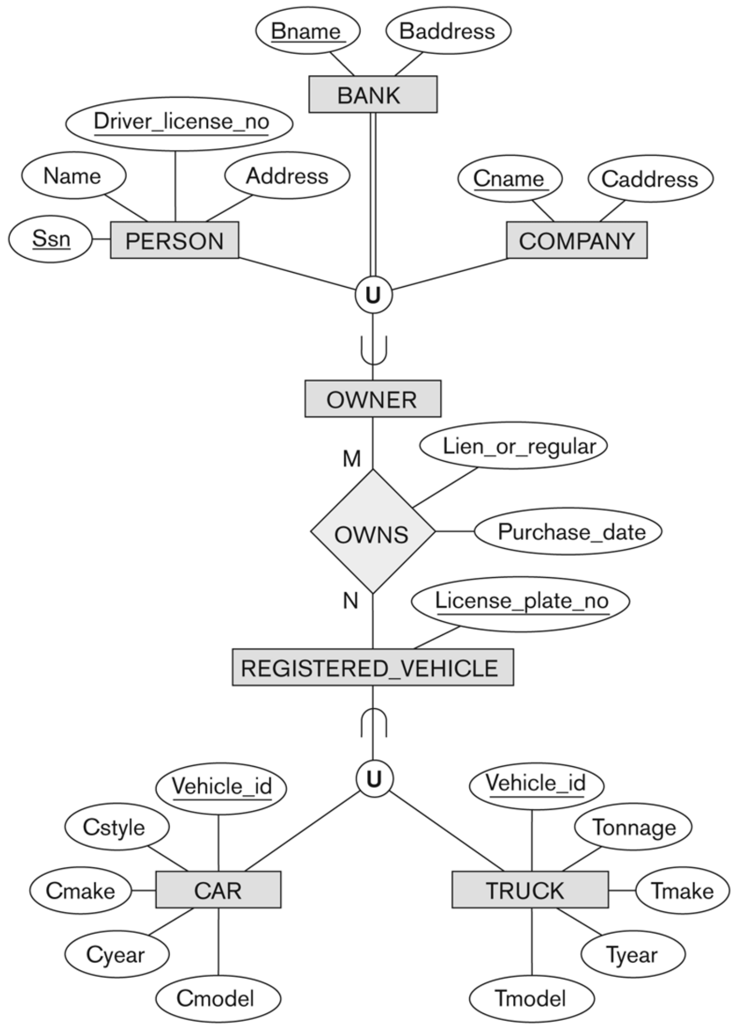


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Categories (UNION TYPES) (3)

- Example: In a database for vehicle registration, a vehicle owner can be a PERSON, a BANK (holding a lien on a vehicle) or a COMPANY.
  - A *category* (UNION type) called OWNER is created to represent a subset of the *union* of the three superclasses COMPANY, BANK, and PERSON
  - A category member must exist in ***at least one*** of its superclasses
- Difference from *shared subclass*, which is a:
  - subset of the *intersection* of its superclasses
  - shared subclass member must exist in ***all*** of its superclasses

Formal Definitions of EER Model (1)

- Class C:
  - A type of entity with a corresponding set of entities:
    - could be entity type, subclass, superclass, or category
- Note: The definition of *relationship type* in ER/EER should have 'entity type' replaced with 'class‘ to allow relationships among classes in general
- Subclass S is a class whose:
    - Type inherits all the attributes and relationship of a class C
    - Set of entities must always be a subset of the set of entities of the other class C
      - $S \subseteq C$
    - C is called the superclass of S
    - A superclass/subclass relationship exists between S and C

Formal Definitions of EER Model (2)

- Specialization Z: Z = {S1, S2,..., Sn} is a set of subclasses with same superclass G; hence, G/Si is a superclass relationship for i = 1, …., n.
  - G is called a generalization of the subclasses {S1, S2,..., Sn}
  - Z is total if we always have:
    - S1 ∪ S2 ∪ … ∪ Sn = G;
    - Otherwise, Z is partial.
  - Z is disjoint if we always have:
    - Si ∩ S2 empty-set for i ≠ j;
  - Otherwise, Z is overlapping.

Formal Definitions of EER Model (3)

- Subclass S of C is predicate defined if predicate (condition) p on attributes of C is used to specify membership in S;
  - that is, S = C[p], where C[p] is the set of entities in C that satisfy condition p
- A subclass not defined by a predicate is called user-defined
- Attribute-defined specialization: if a predicate A = ci (where A is an attribute of G and ci is a constant value from the domain of A) is used to specify membership in each subclass Si in Z
  - Note: If ci ≠ cj for i ≠ j, and A is single-valued, then the attribute-defined specialization will be disjoint.

Formal Definitions of EER Model (4)

- Category or UNION type T
  - A class that is a subset of the *union* of n defining superclasses
    D1, D2,…Dn, n>1:
    - $T \subseteq (D1 \cup D2 \cup \ldots \cup Dn)$
  - Can have a predicate pi on the attributes of Di to specify entities of Di that are members of T.
  - If a predicate is specified on every Di: $T = (D1[p1] \cup D2[p2] \cup \ldots \cup Dn[pn])$

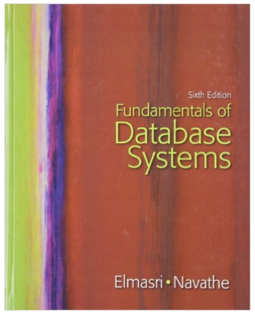
Some of the Currently Available Automated Database Design Tools

COMPANY	TOOL	FUNCTIONALITY
Embarcadero Technologies	ER Studio	Database Modeling in ER and IDEF1X
	DB Artisan	Database administration, space and security management
Oracle	Developer 2000/Designer 2000	Database modeling, application development
Popkin Software	System Architect 2001	Data modeling, object modeling, process modeling, structured analysis/design
Platinum (Computer Associates)	Enterprise Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Persistence Inc.	Pwertier	Mapping from O-O to relational model
Rational (IBM)	Rational Rose	UML Modeling & application generation in C++/JAVA
Resolution Ltd.	Xcase	Conceptual modeling up to code maintenance
Sybase	Enterprise Application Suite	Data modeling, business logic modeling
Visio	Visio Enterprise	Data modeling, design/reengineering Visual Basic/C++

Summary

- ER model concepts: Entities, attributes and relationships
- Constraints in the ER model
- Using ER in step-by-step conceptual schema design for the COMPANY database
- ER Diagrams - Notation
- Alternative Notations – UML class diagrams, others
- Relationships of Higher Degree
- Enhanced ER model (EER)

Bibliography



- Chapter 7
- Chapter 8

- Chapter 4