Database Systems

Session 3 – Main Theme

Enterprise Data Modeling
Using The Entity/Relationship (ER) Model

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Presentation material partially based on textbook slides
by Ramez Elmasri and Shamkant Navathe
Slides copyright © 2015 and on slides produced by Zvi Kedem copyright © 2014
Agenda

1. Session Overview
2. Enterprise Data Modeling Using the ER Model
3. Enterprise Data Modeling Using the EER Model
4. Case Study
5. Summary and Conclusion
Session Agenda

- Session Overview
- Data Modeling Using the Entity-Relationship (ER) Model
- Data Modeling Using the Enhanced Entity-Relationship (EER) Model
- Case Study
- Summary & Conclusion
What is the class about?

- Course description and syllabus:
  - [http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001](http://www.nyu.edu/classes/jcf/CSCI-GA.2433-001)
  - [http://cs.nyu.edu/courses/spring16/CSCI-GA.2433-001/](http://cs.nyu.edu/courses/spring16/CSCI-GA.2433-001/)

- Textbooks:
    - Ramez Elmasri and Shamkant Navathe
    - Addition Wesley
Agenda

- Overview of Database Design Process
- Example Database Application (COMPANY)
- ER Model Concepts
  - Entities and Attributes
  - Entity Types, Value Sets, and Key Attributes
  - Relationships and Relationship Types
  - Weak Entity Types
  - Roles and Attributes in Relationship Types
- ER Diagrams - Notation
- ER Diagram for COMPANY Schema
- Alternative Notations – UML class diagrams, others
- Relationships of Higher Degree
Overview of Database Design Process

- Two main activities:
  - Database design
  - Applications design
- Focus in this chapter on conceptual database design
  - To design the conceptual schema for a database application
- Applications design focuses on the programs and interfaces that access the database
  - Generally considered part of software engineering
Overview of Database Design Process

Figure 3.1
A simplified diagram to illustrate the main phases of database design.
Methodologies for Conceptual Design

- Entity-Relationship (ER) modeling
  - Popular high-level conceptual data model
- ER diagrams
  - Diagrammatic notation associated with the ER model
- Enhanced Entity Relationship (EER) diagrams
- Use of Design Tools in industry for designing and documenting large scale designs
- Unified Modeling Language (UML)
  - Class diagrams are popular in industry to document conceptual database designs
Requirements collection and analysis
- Database designers interview prospective database users to understand and document data requirements
- Result: data requirements
- Functional requirements of the application

Conceptual schema
- Conceptual design
- Description of data requirements
- Includes detailed descriptions of the entity types, relationships, and constraints
- Transformed from high-level data model into implementation data model
- Logical design or data model mapping
  - Result is a database schema in implementation data model of DBMS

- Physical design phase
  - Internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files specified
- **Entity/relationship (ER) model** provides a common, informal, and convenient method for communication between application end users (customers) and the Database Administrator to model the information’s structure.

- This is a preliminary stage towards defining the database using a formal model, such as the relational model, to be described later.

- The ER model, frequently employs **ER diagrams**, which are pictorial descriptions to visualize information’s structure.

- ER models are both surprisingly both **simple** and **powerful**.
There are three basic concepts appearing in the original ER model, which has since been extended.
» We will present the model from more simple to more complex concepts, with examples on the way.

We will go beyond the original ER model, and cover most of Enhanced ER model.

While the ER model’s concepts are standard, there are several varieties of pictorial representations of ER diagrams.
» We will focus on one of them: Chen’s notation.
» We will also cover Crow’s foot notation in the context of the Visio tool.
» Others are simple variations, so if we understand the above, we can easily understand all of them.

You can look at some examples at:
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.
A Sample ER Diagram for the COMPANY Database

Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.
Basic Concepts

- The three basic concepts are (elaborated on very soon):
  - **Entity.** This is an “object.” Cannot be defined even close to a formal way. Examples:
    » Bob
    » Boston
    » The country whose capital is Paris
    There is only one such country so it is completely specified
  - **Relationship.** Entities participate in relationships with each other. Examples:
    » Alice and Boston are in relationship Likes (Alice likes Boston)
    » Bob and Atlanta are not in this relationship
  - **Attribute.** Examples:
    » Age is a property of persons
    » Size is a property of cities
Entities and Attributes

Entity is a basic concept for the ER model. Entities are specific things or objects 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 – e.g. integer, string, date, enumerated type, …
Entities and Attributes

Figure 7.3
Two entities, EMPLOYEE $e_1$, and COMPANY $c_1$, and their attributes.
- **Entity type**
  - Collection (or set) of entities that have the same attributes

---

**Figure 7.6**
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.
- 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.
- 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 underlined (Note: this is different from the relational schema where only one “primary key is underlined).
- Key or uniqueness constraint
  - Attributes whose values are distinct for each individual entity in entity set
  - Key attribute
    • Uniqueness property must hold for every entity set of the entity type

- Value sets (or domain of values)
  - Specifies set of values that may be assigned to that attribute for each individual entity
- **Entity** is a “thing” that is distinguished from others in our application
  - Example: Alice
- All entities of the same “type” form an **entity set**; we use the term “type” informally
  - Example: Person (actually a set of persons). Alice is an entity in this entity set
- What type is a little tricky sometimes
- Example. Do we partition people by sex or not?
  - Sometimes makes sense (gave birth)
    - This allows better enforcement of constraints. You could “automatically” make sure that only entities in the set of women, but not in the set of men can give birth
  - Sometimes not (employment)
Example. When we say “the set of all Boeing airplanes,” is this

» The set of all models appearing in Boeing’s catalog (abstract objects), or

» The set of airplanes that Boeing manufactured (concrete objects)

We may be interested in both and have two entity sets that are somehow related

We will frequently use the term “entity” while actually referring to entity sets, unless this causes confusion
Pictorially, an entity set is denoted by a rectangle with its type written inside.

By convention, singular noun, though we may not adhere to this convention if not adhering to it makes things clearer.

By convention, capitalized, or all capitals, if acronym.

Person
Each entity type will have a collection of entities stored in the database

- Called the **entity set** or sometimes **entity collection**

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

- However, entity type and entity set may be given different names

- Entity set is the current **state** of the entities of that type that are stored in the database
An entity may have (and in general has) a set of zero or more **attributes**, which are some properties.

Each attribute is drawn from some domain (such as integers) possibly augmented by **NULL** (more about NULLs later).

All entities in an entity set have the same set of properties, though not generally with the same values.

Attributes of an entity are written in ellipses (for now solid lines) connected to the entity.

Attributes can be

- **Base** (such as DOB); or **derived** denoted by dashed ellipses (such as Age, derived from DOB and the current date)
- **Simple** (such as DOB); or **composite** having their component attributes attached to them (such as Address, when we think of it explicitly as consisting of street and number and restricting ourselves to one city only)
- **Singlevalued** (such as DOB); or **multivalued** with unspecified in advance number of values denoted by thick-lined ellipses (such as Child; a person may have any number of children—we do not consider children as persons at this point, this means that they are not entities, just attributes of persons)
To have a simple example of a person with attributes:

- Child: Bob
- Child: Carol
- FN: Alice
- LN: Xie
- DOB: 1980-01-01
- Address.Number: 100
- Address.Street: Mercer
- Age: Current Date minus DOB specified in years (rounded down)
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.

- **Multi-valued**
  - An entity may have multiple values for that attribute. For example, Color of a CAR or 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.

For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by \{PreviousDegrees (College, Year, Degree, Field)\}.

Multiple PreviousDegrees values can exist.

Each has four subcomponent attributes:
- College, Year, Degree, Field
Example of a composite attribute

**Figure 3.4**
A hierarchy of composite attributes.
Each simple attribute is associated with a value set

- E.g., Lastname has a value which is a character string of up to 15 characters, say
- Date has a value consisting of MM-DD-YYYY where each letter is an integer

A value set specifies the set of values associated with an attribute
Attributes and Value Sets

- Value sets are similar to data types in most programming languages – e.g., integer, character (n), real, bit
- Mathematically, an attribute $A$ for an entity type $E$ whose value set is $V$ is defined as a function

$$A : E \rightarrow P(V)$$

Where $P(V)$ indicates a power set (which means all possible subsets) of $V$. The above definition covers simple and multivalued attributes.
- We refer to the value of attribute $A$ for entity $e$ as $A(e)$. 
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
- See the full ER notation in advance on the next slide
Figure 3.14
Summary of the notation for ER diagrams.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td></td>
<td>Weak Entity</td>
</tr>
<tr>
<td></td>
<td>Relationship</td>
</tr>
<tr>
<td></td>
<td>Identifying Relationship</td>
</tr>
<tr>
<td></td>
<td>Attribute</td>
</tr>
<tr>
<td></td>
<td>Key Attribute</td>
</tr>
<tr>
<td></td>
<td>Multivalued Attribute</td>
</tr>
<tr>
<td></td>
<td>Composite Attribute</td>
</tr>
<tr>
<td></td>
<td>Derived Attribute</td>
</tr>
<tr>
<td>$E_1$- $R$- $E_2$</td>
<td>Total Participation of $E_2$ in $R$</td>
</tr>
<tr>
<td>$E_1$- $R$- $E_2$</td>
<td>Cardinality Ratio 1: N for $E_1$,$E_2$ in $R$</td>
</tr>
<tr>
<td>$R$- $E$</td>
<td>Structural Constraint (min, max) on Participation of $E$ in $R$</td>
</tr>
</tbody>
</table>
Entity Type CAR with two keys and a corresponding Entity Set

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.

(a)

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

(b)

CAR₁
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR₂
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR₃
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})
Most of the times, some subset (proper or not) of the attributes of an entity has the property that two different entities must differ on the values of these attributes.

This must hold for all conceivable entities in our database.

Such a set of attributes is called a **superkey** (“weak” superset of a key: either superset or equal).

A minimal superkey is called a **key** (sometimes called a **candidate key**).

This means that no proper subset of it is itself a superkey.
In our example:

- Longitude and Latitude (their values) identify (at most) one City, but only Longitude or only Latitude do not
- (Longitude, Latitude) form a superkey, which is also a key
- (Longitude, Latitude, Size, Name) form a superkey, which is not a key, because Size and Name are superflous
- (Country, State, Name) form another key (and also a superkey, as every key is a superkey)

For simplicity, we assume that every country is divided into states and within a state the city name is unique.
- If an entity set has one or more keys, one of them (no formal rule which one) is chosen as the **primary key**
- In SQL the other keys, loosely speaking, are referred to using the keyword `UNIQUE`
- In the ER diagram, the attributes of the primary key are underlined
- So in our example, **one** of the two below:
Based on the requirements, we can identify four initial entity types in the COMPANY database:

- DEPARTMENT
- PROJECT
- EMPLOYEE
- DEPENDENT

Their initial conceptual 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.
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)

We introduce relationship concepts next.
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.
Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT.
Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

Figure 3.13
An M:N relationship, WORKS_ON.
- **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
  - When an attribute of one entity type refers to another entity type
  - Represent references as relationships not attributes
- Relationship type $R$ among $n$ entity types $E_1, E_2, ..., E_n$
  - Defines a set of associations among entities from these entity types
- Relationship instances $r_i$
  - Each $r_i$ associates $n$ individual entities ($e_1, e_2, ..., e_n$)
  - Each entity $e_j$ in $r_i$ is a member of entity set $E_j$
Relationship Degree

- Degree of a relationship type
  - Number of participating entity types
  - Binary, ternary

- Relationships as attributes
  - Think of a binary relationship type in terms of attributes
Sample Ternary Relationship

Figure 7.10
Some relationship instances in the SUPPLY ternary relationship set.
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

» Note that the relationship type is not shown with an arrow. The name should be typically be readable from left to right and top to bottom.
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)
Recursive Relationship Type is: SUPERVISION
(participation role names are shown)

Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.
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.
A relationship type between the same participating entity type in **distinct roles**

Also called a **self-referencing** relationship type.

Example: the SUPERVISION relationship

EMPLOYEE participates twice in two distinct roles:
  » supervisor (or boss) role
  » supervisee (or subordinate) role

Each relationship instance relates two distinct EMPLOYEE entities:
  » One employee in *supervisor* role
  » One employee in *supervisee* role
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)
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.
Displaying a recursive relationship

- In a recursive relationship type.
  - Both participations are same entity type in different roles.
  - For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).

- In following figure, first role participation labeled with 1 and second role participation labeled with 2.

- In ER diagram, need to display role names to distinguish participations.
A Recursive Relationship Supervision

Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).
Role names and recursive relationships

- Role name signifies role that a participating entity plays in each relationship instance

Recursive relationships

- Same entity type participates more than once in a relationship type in different roles
- Must specify role name
Sample Recursive Relationships

Figure 7.11
A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).
Constraints on Binary Relationship Types

- **Cardinality ratio for a binary relationship**
  - Specifies maximum number of relationship instances that entity can participate in

- **Participation constraint**
  - Specifies whether existence of entity depends on its being related to another entity
  - Types: total and partial
We can specify how many times each entity from some entity set can participate in some relationship, in every instance of the database.

In general we can say that:

- This number is in the interval \([i,j]\), \(0 \leq i \leq j\), with \(i\) and \(j\) integers, denoted by \(i..j\); or
- This number is at the interval \([i, \infty)\), denoted by \(i..*\)

- \(0..*\) means no constraint
- No constraint can also be indicated by not writing out anything

\[i..j\]
• Every person likes exactly 1 country
• Every country is liked by 2 or 3 persons
Returning to an old example without specifying which entities actually exist

We have a relationship: Likes

A typical “participation” in a relationship would be that Joe, IBM, Computer participate in it
We want to specify cardinality constraints that every instance of the database needs to satisfy

» Each person participates in between 1 and 5 relationships
» Each vendor participates in between 3 and 3 (that is exactly 3) relationships
» Each product participates in between 2 and 4 relationships

- This is indicated as follows:
A specific instance of the database

<table>
<thead>
<tr>
<th>Person</th>
<th>Vendor</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>IBM</td>
<td>computer</td>
</tr>
<tr>
<td>Tom</td>
<td>Apple</td>
<td>monitor</td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we have the following tuples in the relationship

- Joe IBM computer
- Tom Apple monitor
- Max Apple computer
- Max IBM monitor
- Max IBM computer
- Tom Apple computer

Then, it is true that:
Let us confirm that our instance of Likes satisfies the required cardinality constraints

**Person:** required between 1 and 5
- Joe in 1
- Tom in 2
- Max in 3

**Product:** required between 2 and 4
- Monitor in 2
- Computer in 4

**Vendor** between 3 and 3
- Apple in 3
- IBM in 3

Note that we do not have to have an entity for every possible permitted cardinality value
- For example, there is no person participating in 4 or 5 tuples
So we can also have, expressing exactly what we had before
Compare to previous notation

- Person → Born → Country
- Person ← Heads → Country
- Person ← Likes → Country
- Person 0..1 → Born → Country
- Person 0..1 ← Heads 0..1 → Country
- Person ← Likes → Country
Attributes of Relationship Types

- Attributes of 1:1 or 1:N relationship types can be migrated to one entity type

- For a 1:N relationship type
  - Relationship attribute can be migrated only to entity type on N-side of relationship

- For M:N relationship types
  - Some attributes may be determined by combination of participating entities
  - Must be specified as relationship attributes
Weak Entity Types

- An entity that does not have a key attribute and that is identification-dependent on another entity type.
- 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 relationship 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
Weak Entity Types

- Do not have key attributes of their own
  - Identified by being related to specific entities from another entity type
- Identifying relationship
  - Relates a weak entity type to its owner
- Always has a total participation constraint
A relationship type 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 can be transferred to the entity type on the N-side of the relationship
Example Attribute of a Relationship Type:
Hours of WORKS_ON

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

- **Cardinality ratio** (of a binary relationship): 1:1, 1:N, N:1, or M:N
  - Shown by placing appropriate numbers on the relationship edges.

- **Participation constraint** (on each participating entity type): total (called existence dependency) or partial.
  - Total shown by double line, partial by single line.

- **NOTE**: These are easy to specify for Binary Relationship Types.
Several entity sets (one or more) can participate in a *relationship*

Relationships are denoted by diamonds, to which the participating entities are “attached”

A relationship could be binary, ternary, ….

By convention, a capitalized verb in third person singular (e.g., Likes)

» We may not follow this convention, when inconvenient
- We will have some examples of relations
- We will use three entity sets, with entities in these sets listed below

<table>
<thead>
<tr>
<th>Person</th>
<th>Vendor</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>IBM</td>
<td>computer</td>
</tr>
<tr>
<td>Tom</td>
<td>Apple</td>
<td>monitor</td>
</tr>
<tr>
<td>Max</td>
<td>Dell</td>
<td>printer</td>
</tr>
<tr>
<td>Mike</td>
<td>HP</td>
<td></td>
</tr>
<tr>
<td>Kim</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 \( \text{min} \) and at most \( \text{max} \) relationship instances in R
- Default (no constraint): \( \text{min}=0 \), \( \text{max}=n \) (signifying no limit)
- Must have \( \text{min} \leq \text{max} \), \( \text{min} \geq 0 \), \( \text{max} \geq 1 \)
- 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

Read the min,max numbers next to the entity type and looking **away from** the entity type
Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.
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
Let’s look at Likes, listing all pairs of \((x,y)\) where person \(x\) Likes product \(y\), and the associated ER diagram.

First listing the relationship informally (we omit article “a”):

- Joe likes computer
- Joe likes monitor
- Tom likes computer
- Max likes computer

Note

- Not every person has to Like a product
- Not every product has to be Liked
- A person can Like many products
- A product can be Liked by many persons
More on Relationships (1/2)

- Formally we say that $R$ is a relationship among (not necessarily distinct) entity sets $E_1$, $E_2$, …, $E_n$ if and only if $R$ is a subset of $E_1 \times E_2 \times \ldots \times E_n$ (Cartesian product)

- In our example above:
  - $n = 2$
  - $E_1 = \{\text{Joe, Tom, Max, Mike, Kim}\}$
  - $E_2 = \{\text{computer, monitor, printer}\}$
  - $E_1 \times E_2 = \{(\text{Joe,computer}), (\text{Joe,monitor}), (\text{Joe,printer}), (\text{Tom,computer}), (\text{Tom,monitor}), (\text{Tom,printer}), (\text{Max,computer}), (\text{Max,monitor}), (\text{Max,printer}), (\text{Mike,computer}), (\text{Mike,monitor}), (\text{Mike,printer}), (\text{Kim,computer}), (\text{Kim,monitor}), (\text{Kim,printer})\}$
  - $R = \{(\text{Joe,computer}), (\text{Joe,monitor}), (\text{Tom,computer}), (\text{Max,monitor})\}$

- $R$ is a set (unordered, as every set) of ordered tuples, or sequences (here of length two, that is pairs)
Let us elaborate

$E_1 \times E_2$ was the "universe"
  » It listed all possible pairs of a person liking a product

At every instance of time, in general only some of this pairs corresponded to the "actual state of the universe", $R$ was the set of such pairs
Relationship types of degree 2 are called binary

Relationship types of degree 3 are called ternary and of degree n are called 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
In general, 3 binary relationships can represent different information than a single ternary relationship (see Figure 3.17a and b on next slide)

If needed, the binary and $n$-ary relationships can all be included in the schema design (see Figure 3.17a and b, where all relationships convey different meanings)

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) (see Figure 3.17c)
Ternary Relationship

- Let’s look at Buys listing all tuples of \((x,y,z)\) where person \(x\) Buys product \(y\) from vendor \(z\)
- Let us just state it informally:
  » Joe buys computer from IBM
  » Joe buys computer from Dell
  » Tom buys computer from Dell
  » Tom buys monitor from Apple
  » Joe buys monitor from IBM
  » Max buys computer from IBM
  » Max buys monitor from Dell
Example of a ternary relationship

Figure 3.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.
Discussion of n-ary relationships (n > 2)

- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant
- For example, the TAUGHT_DURING binary relationship in Figure 3.18 (see next slide) can be derived from the ternary relationship OFFERS (based on the meaning of the relationships)
Another example of a ternary relationship

Figure 3.18
Another example of ternary versus binary relationship types.
Let’s look at Likes, listing all pairs of \((x,y)\) where person \(x\) Likes person \(y\)

Let us just state it informally

- Joe likes Tom
- Joe likes Max
- Tom likes Max
- Tom likes Mike
- Tom likes Tom
- Max likes Tom

Note that pairs must be ordered to properly specify the relationship, Joe likes Tom, but Tom does not like Joe
Again:
- Joe likes Tom
- Joe likes Max
- Tom likes Max
- Tom likes Mike
- Tom likes Tom
- Max likes Tom

Formally Likes is a subset of the Cartesian product Person × Person, which is the set of all ordered pairs of the form (person,person)

Likes is the set \{ (Joe,Tom), (Joe,Max), (Tom,Max), (Tom,Mike), (Tom,Tom), (Max,Tom) \}

Likes is an arbitrary directed graph in which persons serve as vertices and arcs specify who likes whom.
Frequently it is useful to give *roles* to the participating entities, when, as here, they are drawn from the same entity set.

So, we may say that if Joe likes Tom, then Joe is the “Liker” and Tom is the “Liked”

Roles are explicitly listed in the diagram, but the semantics of they mean cannot be deduced from looking at the diagram only.
Consider Buys, listing all triples of the form \((x,y,z)\) where vendor \(x\) Buys product \(y\) from vendor \(z\)

A typical tuple might be \((\text{Dell}, \text{printer}, \text{HP})\), meaning that Dell buys a printer from HP
Refining the ER Design for the COMPANY Database

- Change attributes that represent relationships into relationship types
- Determine cardinality ratio and participation constraint of each relationship type
ER Diagrams, Naming Conventions, and Design Issues

**Figure 7.14**
Summary of the notation for ER diagrams.

- **Symbol**
  - **Meaning**
    - Entity
    - Weak Entity
    - Relationship
    - Identifying Relationship
    - Attribute
    - Key Attribute
    - 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 $R$
ER Diagrams

- To show which entities participate in which relationships, and which attributes participate in which entities, we draw line segments between:
  - Entities and relationships they participate in
  - Attributes and entities they belong to
- We also underline the attributes of the primary key for each entity that has a primary key
- Below is a simple ER diagram (with a simpler Person than we had before):
Proper Naming of Schema Constructs

- Choose names that convey meanings attached to different constructs in schema
- Nouns give rise to entity type names
- Verbs indicate names of relationship types
- Choose binary relationship names to make ER diagram readable from left to right and from top to bottom
Design Choices for ER Conceptual Design

- Model concept first as an attribute
  - Refined into a relationship if attribute is a reference to another entity type
- Attribute that exists in several entity types may be elevated to an independent entity type
  - Can also be applied in the inverse
Specify structural constraints on relationships

- Replaces cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints
- Associate a pair of integer numbers (min, max) with each participation of an entity type $E$ in a relationship type $R$, where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$
Sample Use of Alternative Notations for the COMPANY Schema

Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.
Example of Other Notation: UML Class Diagrams

- UML methodology
  - Used extensively in software design
  - Many types of diagrams for various software design purposes
- UML class diagrams
  - Entity in ER corresponds to an object in UML
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 the COMPANY Conceptual Schema

**Figure 7.16**
The COMPANY conceptual schema in UML class diagram notation.
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.
Class includes three sections:
- Top section gives the class name
- Middle section includes the attributes;
- Last section includes operations that can be applied to individual objects

Associations: relationship types

Relationship instances: links

Binary association
- Represented as a line connecting participating classes
- May optionally have a name

Link attribute
- Placed in a box connected to the association’s line by a dashed line
- Multiplicities: min..max, asterisk (*) indicates no maximum limit on participation
- Types of relationships: association and aggregation
- Distinguish between unidirectional and bidirectional associations
- Model weak entities using qualified association
Relationship Types of Degree Higher than Two

- Degree of a relationship type
  - Number of participating entity types
- Binary
  - Relationship type of degree two
- Ternary
  - Relationship type of degree three
Choosing between Binary and Ternary (or Higher-Degree) Relationships

- Some database design tools permit only binary relationships
  - Ternary relationship must be represented as a weak entity type
  - No partial key and three identifying relationships
- Represent ternary relationship as a regular entity type
  - By introducing an artificial or surrogate key
Figure 7.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.
Notations for specifying structural constraints on $n$-ary relationships

- Should both be used if it is important to fully specify structural constraints
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
- In general, both (min, max) and 1, M, or N are needed to describe fully the constraints
- Overall, the constraint specification is difficult and possibly ambiguous when we consider relationships of a degree higher than two.
There are further refinements to the model and associated diagrams that will be presented piecemeal in the following slides.

The previous modeling concepts and the ones that follow are needed for producing a database design that models a particular application well.

The comprehensive case study provided at the end of this presentation puts it all together.
Consider relationship Buys among Person, Vendor, and Product

We want to specify that a person Buys a product from a vendor at a specific price

Price is not

- A property of a vendor, because different products may be sold by the same vendor at different prices
- A property of a product, because different vendors may sell the same product at different prices
- A property of a person, because different products may be bought by the same person at different prices
So Price is really an attribute of the relationship Buys
For each tuple (person, product, vendor) there is a value of price
Entities can model situations that attributes cannot model naturally.

Entities can:
- Participate in relationships
- Have attributes

Attributes cannot do any of these.

Let us look at a “fleshed out example” for possible alternative modeling of Buys.
Other Choices For Modeling Buys (1/2)

- Price is just the actual amount, the number in $’s
- So there likely is no reason to make it an entity as we have below

- We should probably have (as we had earlier less fleshed out)
Other Choices For Modeling Buys (2/2)

- Or should we just have this?

- Not if we want to model something about a person, such as the date of birth of a person or whom a person likes
- These require a person to have an attribute (date of birth) and enter into a relationship (with other persons)
- And we cannot model this situation if person is an attribute of Buy
- Similarly, for product and vendor
Consider a relationship $R$ between two entity sets $A$, $B$.

We will look at examples where $A$ is the set of persons and $B$ is the set of all countries.

We will be making some simple assumptions about persons and countries, which we list when relevant.
Relationship R is called \textit{many to one} from A to B if and only if for each element of A there exists at most one element of B related to it.

Example: R is Born (in)
Each person was born in at most one country (may not in a country but in a ship).
Maybe nobody was born in some country as it has just been established.

The picture on the right describes the universe of four persons and three countries, with lines indicating which person was born in which country.

We will have similar diagrams for other examples.
The relationship R is called **one to one** between A and B if and only if for each element of A there exists at most one element of B related to it and for each element of B there exists at most one element of A related to it.

Example: R is Heads
- Each Person is a Head (President, Queen, etc.) of at most one country
- Each country has at most one head (maybe the queen died and it is not clear who will be the monarch next)

In other words, R is one to one, if and only if
- R is many to one from A to B, and
- R is many to one from B to A
The relationship is called *many to many* between A and B, if it is not many to one from A to B and it is not many to one from B to A

- Example: R is “likes”
We have in effect considered the concepts of partial functions of one variable.

- The first two examples were **partial functions**
- The last example was not a function

Pictorially, functionality for binary relationships can be shown by drawing an arc head in the direction to the “one”
- How about properties of the relationship?
- Date: when the two people in a relationship first entered into the relationship (marked also with black square)
- Can make Date in some cases the property of an entity
  - “Slide” the Date to the Person, but not the Country
  - “Slide” the Date to either the Person or the Country (but not for both, as this would be redundant)

- Cannot “slide” the Date to either “Liker” or “Liked”

- Can “slide” if no two squares end up in the same entity
This can be done if the relationship is many-to-one

» Then, the property of the relationship can be attributed to the “many” side

This can be done if the relationship is one-to-one

» Then a property of the relationship can be “attributed” to any of the two sides
- Entities “inheriting” attributes of relationships when the relationships are not many to many
Aggregation: Relationships As Entities

- It is sometimes natural to consider relationships as if they were entities.
- This will allow us to let relationships participate in other “higher order” relationships.
- Here each “contract” needs to be approved by (at most) one agency.
- Relationship is “made into” an entity by putting it into a rectangle; note that the edge between Buys and Approves touches the Buys rectangle but not the Buys diamond, to make sure we are not confused.
Strong And Weak Entities

- A **strong entity** (set): Its elements can be identified by the values of their attributes, that is, it has a (primary) key made of its attributes. Tacitly, we assumed only such entities so far.

- A **weak entity** (set): Its elements cannot be identified by the values of their attributes: there is no primary key made from its own attributes. They can be identified by a combination of their attributes and the relationship they have with another entity set.
Most entities are **strong**: a specific entity can be distinguished from other entities based on the values of its attributes.

We assume that every person has his/her own SSN.

Woman is a strong entity as we can identify a specific woman based on her attributes. She has a primary key: her own SSN.

Man is a strong entity as we can identify a specific man based on his attributes. He has a primary key: his own SSN.
We assume that women are given SSNs.

Men are not given SSNs; they have first names only, but for each we know who the mother is (that is, we know the SSN of the man’s mother).

Man is a *weak* entity as we cannot identify a specific man based on his own attributes.

Many women could have a son named Bob, so there are many men named Bob.

However, if a woman never gives the same name to more than one of her sons, a man can be identified by his name and by his mother’s SSN.
We could have the following situation of two mothers: one with two sons, and one with three sons, when we gave people also heights in inches (just to have additional attributes that are not necessary for identification)

- SSN: 070-43-1234, height: 65
  - Name: Bob, height 35
  - Name: Mike, height 35

- SSN: 056-35-4321, height 68
  - Name: Bob, height 35
  - Name: Dave, height 45
  - Name: Victor, height 74
Assuming that a woman does not have more than one son with the same name

Name becomes a *discriminant*

Man can be identified by the combination of:

- The Woman to whom he is related under the Son relation. This is indicated by thick lines around Son (it is weak). Thick line connecting Man to Son indicates the relationship is total on Man (every Man participates) and used for identification
- His Name. His name is now a *discriminant*; this is indicated by double underline
We need to specify for a weak entity through which relationship it is identified; this done by using thick lines.

Otherwise we do not know whether Man is identified through Son or through Works.
- Sometimes a discriminant is not needed
- We are only interested in men who happen to be first sons of women
- Every Woman has at most one First Son
- So we do not need to have Name for Man (if we do not want to store it, but if we do store it, it is not a discriminant)
In general, more than one attribute may be needed as a discriminant.

For example, let us say that man has both first name and middle name.

A mother may give two sons the same first name or the same middle name.

A mother will never give two sons the same first name and the same middle name.

The pair (first name, middle name) together form a discriminant.
There can be several levels of “weakness”

Here we can say that a horse named “Speedy” belongs to Bob, whose mother is a woman with SSN 072-45-9867

A woman can have several sons, each of whom can have several horses
Another Example: A UNIVERSITY Database

- To keep track of the enrollments in classes and student grades, another database is to be designed.

- It keeps track of the COLLEGEs, DEPARTMENTs within each college, the COURSEs offered by departments, and SECTIONs of courses, INSTRUCTORs who teach the sections etc.

- These entity types and the relationships among these entity types are shown on the next slide in Figure 3.20.
UNIVERSITY database conceptual schema
Data Modeling Tools (Additional Material)

- A number of popular tools that cover conceptual modeling and mapping into relational schema design.
  - Examples: ERWin, S-Designer (Enterprise Application Suite), ER-Studio, etc.

- **POSITIVES:**
  - Serves as documentation of application requirements, easy user interface - mostly graphics editor support

- **NEGATIVES:**
  - Most tools lack a proper distinct notation for relationships with relationship attributes
  - Mostly represent a relational design in a diagrammatic form rather than a conceptual ER-based design
## Automated Database Design Tools

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>TOOL</th>
<th>FUNCTIONALITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embarcadero Technologies</td>
<td>ER Studio</td>
<td>Database Modeling in ER and IDEF1X</td>
</tr>
<tr>
<td></td>
<td>DB Artisan</td>
<td>Database administration, space and security management</td>
</tr>
<tr>
<td>CA</td>
<td>ERWin</td>
<td>Database Modeling in EER and logical/physical</td>
</tr>
<tr>
<td>Oracle</td>
<td>Developer 2000/Designer 2000</td>
<td>Database modeling, application development</td>
</tr>
<tr>
<td>Popkin Software</td>
<td>System Architect 2001</td>
<td>Data modeling, object modeling, process modeling, structured analysis/design</td>
</tr>
<tr>
<td>Platinum (Computer Associates)</td>
<td>Enterprise Modeling Suite: Erwin, BPWin, Paradigm Plus</td>
<td>Data, process, and business component modeling</td>
</tr>
<tr>
<td>Persistence Inc.</td>
<td>Pwertier</td>
<td>Mapping from O-O to relational model</td>
</tr>
<tr>
<td>Rational (IBM)</td>
<td>Rational Rose</td>
<td>UML Modeling &amp; application generation in C++/JAVA</td>
</tr>
<tr>
<td>Resolution Ltd.</td>
<td>Xcase</td>
<td>Conceptual modeling up to code maintenance</td>
</tr>
<tr>
<td>Sybase</td>
<td>Enterprise Application Suite</td>
<td>Data modeling, business logic modeling</td>
</tr>
<tr>
<td>Visio</td>
<td>Visio Enterprise</td>
<td>Data modeling, design/reengineering Visual Basic/C++</td>
</tr>
</tbody>
</table>
The entity relationship model in its original form did not support the specialization and generalization abstractions.

Next chapter illustrates how the ER model can be extended with:

- Type-subtype and set-subset relationships
- Specialization/Generalization Hierarchies
- Notation to display them in EER diagrams
Summary

- ER Model Concepts: Entities, attributes, relationships
- Constraints in the ER model
- Using ER in step-by-step mode conceptual schema design for the COMPANY database
- ER Diagrams - Notation
- Alternative Notations – UML class diagrams, others
- Binary Relationship types and those of higher degree.
1. Session Overview
2. Enterprise Data Modeling Using the ER Model
3. Enterprise Data Modeling Using the EER Model
4. Case Study
5. Summary and Conclusion
- EER stands for Enhanced ER or Extended ER
- EER Model Concepts
  - Includes all modeling concepts of basic ER
  - Additional concepts:
    - subclasses/superclasses
    - specialization/generalization
    - categories (UNION types)
    - attribute and relationship inheritance
  - Constraints on Specialization/Generalization
- The additional EER concepts are used to model applications more completely and more accurately
  - EER includes some object-oriented concepts, such as inheritance
- Knowledge Representation and Ontology Concepts
The Enhanced Entity-Relationship (EER) Model

- Enhanced ER (EER) model
  - Created to design more accurate database schemas
    - Reflect the data properties and constraints more precisely
  - More complex requirements than traditional applications
EER model includes all modeling concepts of the ER model

In addition, EER includes:
- Subclasses and superclasses
- Specialization and generalization
- Category or union type
- Attribute and relationship inheritance
Subclasses and Superclasses

- 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 (the role they play)
    - 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
- Enhanced ER or EER diagrams
  - Diagrammatic technique for displaying these concepts in an EER schema
- Subtype or subclass of an entity type
  - Subgroupings of entities that are meaningful
  - Represented explicitly because of their significance to the database application
Terms for relationship between a superclass and any one of its subclasses

- Superclass/subclass
- Supertype/subtype
- Class/subclass relationship

Type inheritance

- Subclass entity inherits all attributes and relationships of superclass
Subclasses and Superclasses

Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}
Each of these subgroupings is a subset of EMPLOYEE entities

Each is called a subclass of EMPLOYEE

EMPLOYEE is the superclass for each of these subclasses

These are called superclass/subclass relationships:
  » EMPLOYEE/SECRETARY
  » EMPLOYEE/TECHNICIAN
  » EMPLOYEE/MANAGER
  » …
Subclasses and Superclasses (2)

- 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 (3)

- **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

- It is not necessary that every entity in a superclass be a member of some subclass.
For certain purposes, we consider subsets of an entity set.

The subset relationship between the set and its subset is called \textit{ISA}, meaning “is a”.

Elements of the subset, of course, have all the attributes and relationships as the elements of the set: they are in the “original” entity set.

In addition, they may participate in relationships and have attributes that make sense for them.

- But do not make sense for every entity in the “original” entity set.

ISA is indicated by a triangle.

The elements of the subset are weak entities, as we will note next.
Example: A subset that has an attribute that the original set does not have

We look at all the persons associated with a university

Some of the persons happen to be professors and some of the persons happen to be students
The ISA Relationship (3/7)

- Professor is a weak entity because it cannot be identified by its own attributes (here: Salary)
- Student is a weak entity because it cannot be identified by its own attributes (here: GPA)
- They do not have discriminants, nothing is needed to identify them in addition to the primary key of the strong entity (Person)
- The set and the subsets are sometimes referred to as class and subclasses
A person associated with the university (and therefore in our database) can be
- Only a professor
- Only a student
- Both a professor and a student
- Neither a professor nor a student

In general, ISA could be
- **Disjoint**: no entity could be in more than one subclass
- **Overlapping**: an entity could be in more than one subclass
- **Total**: every entity has to be in at least one subclass
- **Partial**: an entity does not have to be in any subclass

This could be specified by replacing “ISA” in the diagram by an appropriate letter.
Some persons are professors
Some persons are students
Some persons are neither professors nor students
No person can be both a professor and a student
Example: subsets participating in relationships modeling the assumed semantics more clearly (every person has one woman who is the legal mother)
• ISA is really a superclass/subclass relationship
• ISA could be *specialization*: subsets are made out of the “more basic” set
• ISA could be *generalization*: a superset is made of “more basic” sets
• Again, the diagram could be annotated to indicate this
Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.
Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}
Specialization and Generalization

- **Specialization**
  - Process of defining a set of subclasses of an entity type
  - Defined on the basis of some distinguishing characteristic of the entities in the superclass
- **Subclass can define:**
  - Specific attributes
  - Specific relationship types
Instances of a Specialization

Figure 8.2
Instances of a specialization.
• Certain attributes may apply to some but not all entities of the superclass
• Some relationship types may be participated in only by members of the subclass
Generalization

- Reverse process of abstraction
- Generalize into a single superclass
  - Original entity types are special subclasses
- Generalization
  - Process of defining a generalized entity type from the given entity types
**Attribute Inheritance in Superclass / Subclass Relationships**

- An entity that is member of a subclass *inherits*
  - All attributes of the entity as a member of the superclass
  - All relationships of the entity as a member of the superclass

- Example:
  - In the previous slide, SECRETARY (as well as TECHNICIAN and ENGINEER) inherit the attributes Name, SSN, …, from EMPLOYEE
  - Every SECRETARY entity will have values for the inherited attributes
Specialization is the process of defining a set of subclasses of a superclass

The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass

» Example: \{SECRETARY, ENGINEER, TECHNICIAN\} is a specialization of EMPLOYEE based upon job type.

» Example: MANAGER is a specialization of EMPLOYEE based on the role the employee plays
  • May have several specializations of the same superclass
Example: Another specialization of EMPLOYEE based on *method of pay* is \{SALARIED_EMPLOYEE, HOURLY_EMPLOYEE\}.

» Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams

» Attributes of a subclass are called *specific* or *local* attributes.
  - For example, the attribute TypingSpeed of SECRETARY

» The subclass can also participate in specific relationship types.
  - For example, a relationship BELONGS_TO of HOURLY_EMPLOYEE
Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}
Generalization is the reverse of the specialization process

Several classes with common features are generalized into a superclass;
» original classes become its subclasses

Example: CAR, TRUCK generalized into VEHICLE;
» both CAR, TRUCK become subclasses of the superclass VEHICLE.

» We can view \{CAR, TRUCK\} as a specialization of VEHICLE

» Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK
Generalization (2)

Figure 4.3

Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.
Diagrammatic notations are sometimes used to distinguish between generalization and specialization

- Arrow pointing to the generalized superclass represents a generalization
- Arrows pointing to the specialized subclasses represent a specialization
- *We do not use* this notation because it is often subjective as to which process is more appropriate for a particular situation
- *We advocate not drawing any arrows*
Data Modeling with Specialization and Generalization

- A superclass or subclass represents a collection (or set or grouping) of entities
- It also represents a particular *type of entity*
- Shown in rectangles in EER diagrams (as are entity types)
- We can call all entity types (and their corresponding collections) *classes*, whether they are entity types, superclasses, or subclasses
Types of Specialization

- Predicate-defined (or condition-defined): based on some predicate. E.g., based on value of an attribute, say, Job-type, or Age.
- Attribute-defined: shows the name of the attribute next to the line drawn from the superclass toward the subclasses (see Fig. 4.1)
- User-defined: membership is defined by the user on an entity by entity basis
• Constraints that apply to a single specialization or a single generalization
• Differences between specialization/generalization lattices and hierarchies
May be several or one subclass

Determine entity subtype:
- Predicate-defined (or condition-defined) subclasses
- Attribute-defined specialization
- User-defined
- Disjointness constraint
  - Specifies that the subclasses of the specialization must be disjoint
- Completeness (or totalness) constraint
  - May be total or partial
- Disjointness and completeness constraints are independent
If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses.

- Condition is a constraint that determines subclass members.
- Display a predicate-defined subclass by writing the predicate condition next to the line attaching the subclass to its superclass.
If all subclasses in a specialization have membership condition on same attribute of the superclass, specialization is called an attribute-defined specialization

- Attribute is called the defining attribute of the specialization
- Example: JobType is the defining attribute of the specialization \{SECRETARY, TECHNICIAN, ENGINEER\} of EMPLOYEE

If no condition determines membership, the subclass is called user-defined

- Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass
- Membership in the subclass is specified individually for each entity in the superclass by the user
Displaying an attribute-defined specialization in EER diagrams

**Figure 4.4**
EER diagram notation for an attribute-defined specialization on Job_type.
Two basic constraints can apply to a specialization/generalization:

» Disjointness Constraint:
» Completeness Constraint:
Disjointness Constraint:

» 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

» 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
Completeness (Exhaustiveness) 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
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.
Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.
Example of overlapping total Specialization

Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.
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 hierarchy
  - Every subclass participates as a subclass in only one class/subclass relationship
  - Results in a tree structure or strict hierarchy

- Specialization lattice
  - Subclass can be a subclass in more than one class/subclass relationship
Shared Subclass “Engineering_Manager”

A specialization lattice with shared subclass ENGINEERING_MANAGER.
Figure 8.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.
Multiple inheritance

- Subclass with more than one superclass
- If attribute (or relationship) originating in the same superclass inherited more than once via different paths in lattice
  - Included only once in shared subclass

Single inheritance

- Some models and languages limited to single inheritance
Specialization process
- Start with entity type then define subclasses by successive specialization
- Top-down conceptual refinement process

Bottom-up conceptual synthesis
- Involves generalization rather than specialization
In a lattice or hierarchy, a subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclasses.

A subclass with more than one superclass is called a shared subclass (multiple inheritance).

Can have:
- specialization hierarchies or lattices, or
- generalization hierarchies or lattices,
- depending on how they were derived

We just use specialization (to stand for the end result of either specialization or generalization)
In *specialization*, start with an entity type and then define subclasses of the entity type by successive specialization
   » called a *top down* conceptual refinement process

In *generalization*, start with many entity types and generalize those that have common properties
   » Called a *bottom up* conceptual synthesis process

In practice, a *combination of both processes* is usually employed
Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.
- Union type or a category
- Represents a single superclass/subclass relationship with more than one superclass
- Subclass represents a collection of objects that is a subset of the UNION of distinct entity types
- Attribute inheritance works more selectively
- Category can be total or partial
- Some modeling methodologies do not have union types
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.
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 (typically just 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
Two categories (UNION types): OWNER, REGISTERED_VEHICLE

Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.
The UNIVERSITY Database Example

- UNIVERSITY database
  - Students and their majors
  - Transcripts, and registration
  - University’s course offerings
Figure 8.9
An EER conceptual schema for a UNIVERSITY database.
Many specializations and subclasses can be defined to make the conceptual model accurate.

If subclass has few specific attributes and no specific relationships:
  - Can be merged into the superclass.
If all the subclasses of a specialization/generalization have few specific attributes and no specific relationships

- Can be merged into the superclass
- Replace with one or more type attributes that specify the subclass or subclasses that each entity belongs to
- Union types and categories should generally be avoided
- Choice of disjoint/overlapping and total/partial constraints on specialization/generalization
  - Driven by rules in miniworld being modeled
Formal Definitions for the EER Model Concepts (1/2)

- **Class**
  - Set or collection of entities
  - Includes any of the EER schema constructs of group entities

- **Subclass**
  - Class whose entities must always be a subset of the entities in another class

- **Specialization**
  - Set of subclasses that have same superclass
- **Generalization**
  - Generalized entity type or superclass
- **Predicate-defined**
  - Predicate on the attributes of is used to specify which entities in $C$ are members of $S$
- **User-defined**
  - Subclass that is not defined by a predicate
- **Category**
  - Class that is a subset of the union of $n$ defining superclasses
- **Relationship type**
  - Any class can participate in a relationship
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
**Specialization Z:** \( Z = \{S_1, S_2, \ldots, S_n\} \) is a set of subclasses with same superclass \( G \); hence, \( G/S_i \) is a superclass relationship for \( i = 1, \ldots, n \).

- \( G \) is called a generalization of the subclasses \( \{S_1, S_2,\ldots, S_n\} \)
- \( Z \) is total if we always have:
  - \( S_1 \cup S_2 \cup \ldots \cup S_n = G \);
  - Otherwise, \( Z \) is partial.
- \( Z \) is disjoint if we always have:
  - \( S_i \cap S_2 \) empty-set for \( i \neq j \);
  - 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 $S_i$ in $Z$
  - Note: If $ci \neq cj$ for $i \neq j$, and $A$ is single-valued, then the attribute-defined specialization will be disjoint.
Category or UNION type T

» A class that is a subset of the union of n defining superclasses D1, D2,…Dn, n>1:
  • T ⊆ (D1 ∪ D2 ∪ … ∪ 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] ∪ D2[p2] ∪…∪ Dn[pn])
Alternative diagrammatic notations

- ER/EER diagrams are a specific notation for displaying the concepts of the model diagrammatically.

- DB design tools use many alternative notations for the same or similar concepts.

- One popular alternative notation uses UML class diagrams.

- See next slides for UML class diagrams and other alternative notations.
Representing specialization and generalization in UML class diagrams

- Basic notation
  - See Figure 8.10

- Base class
  - Root superclass

- Leaf classes
  - Subclasses (leaf nodes)
Sample UML Class Diagram

Figure 8.10
A UML class diagram corresponding to the EER diagram in Figure 8.7, illustrating UML notation for specialization/generalization.
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.
Goal of knowledge representation (KR) techniques

- Accurately model some domain of knowledge
- Create an ontology that describes the concepts of the domain and how these concepts are interrelated

Goals of KR are similar to those of semantic data models

- Important similarities and differences
Classification and Instantiation (1/2)

- **Classification**
  - Systematically assigning similar objects/entities to object classes/entity types

- **Instantiation**
  - Inverse of classification
  - Generation and specific examination of distinct objects of a class
Exception objects
  - Differ in some respects from other objects of class
  - KR schemes allow such class properties

One class can be an instance of another class (called a meta-class)
  - Cannot be represented directly in EER model
Abstraction process

Classes and objects are made uniquely identifiable by means of some identifier

Needed at two levels

- To distinguish among database objects and classes
- To identify database objects and to relate them to their real-world counterparts
Specialization and Generalization

- **Specialization**
  - Classify a class of objects into more specialized subclasses

- **Generalization**
  - Generalize several classes into a higher-level abstract class
  - Includes the objects in all these classes
Aggregation and Association

- **Aggregation**
  - Abstraction concept for building composite objects from their component objects

- **Association**
  - Associate objects from several independent classes

- **Main structural distinction**
  - When an association instance is deleted
    - Participating objects may continue to exist
**Figure 8.11**

Aggregation. (a) The relationship type INTERVIEW. Including JOB_OFFER in a ternary relationship type (incorrect). (c) Having the RESULTS_IN relationship participate in other relationships (not allowed in ER). (d) Using aggregation and a composite (molecular) object (generally not allowed in ER but allowed by some modeling tools). (e) Correct representation in ER.
Figure 8.11
Aggregation. (a) The relationship type INTERVIEW. (b) Including JOB_OFFER in a ternary relationship type (incorrect). (c) Having the RESULTS_IN relationship participate in other relationships (not allowed in ER). (d) Using aggregation and a composite (molecular) object (generally not allowed in ER but allowed by some modeling tools). (e) Correct representation in ER.
Deals with modeling and representing a certain domain of knowledge.

Typically done by using some formal model of representation and by creating an Ontology.

An ontology for a specific domain of interest describes a set of concepts and interrelationships among those concepts.

An Ontology serves as a “schema” which enables interpretation of the knowledge in a “knowledge-base”
COMMON FEATURES between KR and Data Models:
- Both use similar set of abstractions – classification, aggregation, generalization, and identification.
- Both provide concepts, relationships, constraints, operations and languages to represent knowledge and model data

DIFFERENCES:
- KR has broader scope: tries to deal with missing and incomplete knowledge, default and common-sense knowledge etc.
DIFFERENCES (continued):

- KR schemes typically include rules and reasoning mechanisms for inferencing
- Most KR techniques involve data and metadata. In data modeling, these are treated separately
- KR is used in conjunction with artificial intelligence systems to do decision support applications
GENERAL Basis for Conceptual Modeling

- **TYPES OF DATA ABSTRACTIONS**
  - CLASSIFICATION and INSTANTIATION
  - AGGREGATION and ASSOCIATION (relationships)
  - GENERALIZATION and SPECIALIZATION
  - IDENTIFICATION

- **CONSTRAINTS**
  - CARDINALITY (Min and Max)
  - COVERAGE (Total vs. Partial, and Exclusive (Disjoint) vs. Overlapping)
Use conceptual modeling and other tools to develop “a specification of a conceptualization”

- **Specification** refers to the language and vocabulary (data model concepts) used
- **Conceptualization** refers to the description (schema) of the concepts of a particular field of knowledge and the relationships among these concepts

Many medical, scientific, and engineering ontologies are being developed as a means of standardizing concepts and terminology
Ontologies and the Semantic Web

• Documents contain less structure than database information does

• Semantic Web
  • Allow meaningful information exchange and search among machines

• Ontology
  • Specification of a conceptualization

• Specification
  • Language and vocabulary terms used to specify conceptualization
Summary

- Introduced the EER model concepts
  - Class/subclass relationships
  - Specialization and generalization
  - Inheritance
- Constraints on EER schemas
- These augment the basic ER model concepts introduced in Chapter 3
- EER diagrams and alternative notations were presented
- Knowledge Representation and Ontologies were introduced and compared with Data Modeling
1. Session Overview
2. Enterprise Data Modeling Using the ER Model
3. Enterprise Data Modeling Using the EER Model
4. Case Study
5. Summary and Conclusion
Next, we will go through a relatively large example to make sure we know how to use ER diagrams.

We have a large application to make sure we understand all the points.

The fragment has been constructed so it exhibits interesting and important capabilities of modeling.

It will also review the concepts we have studied earlier.

It is chosen based on its suitability to practice modeling using the power of ER diagrams.

It will also exercise various points, to be discussed later on how to design good relational databases.
We are supposed to design a database for a university

We will look at a small fragment of the application and will model it as an entity relationship diagram annotated with comments, as needed to express additional features.

But it is still a reasonable “small” database.

In fact, much larger than what is commonly discussed in a course, but more realistic for modeling real applications.
Our Application (2/2)

- Our understanding of the application will be described in a narrative form.
- While we do this, we construct the ER diagram.
- For ease of exposition (technical reasons only: limitations of AV equipment) we look at the resulting ER diagram and construct it in pieces.
- We will pick some syntax for annotations, as this is not standard.
- We describe the application in stages, getting:
- **Horse**: entity set
- Attributes:
  - `Name`
- Constraints
  - `Primary Key: Name`
Our ER Diagram

Diagram showing the relationship between 'Name' and 'Horse'.
- We should specify what is the domain of each attribute, in this case, Name only.
- We will generally not do it in our example, as there is nothing interesting in it.
  » We could say that Name is an alphabetic string of at most 100 characters, for example.
- **Person**: entity set

- **Attributes**:
  - *Child*: a multivalued attribute
  - *ID#*
  - *SS#*
  - *Name*: composite attribute, consisting of
    - *FN*
    - *LN*
  - *DOB*
  - *Age*: derived attribute (we should state how it is computed)

- **Constraints**
  - Primary Key: ID#
  - Unique: SS#
Our ER Diagram
Since ID# is the primary key (consisting here of one attribute), we will consistently identify a person using the value of this attribute.

Since SS# is unique, no two persons will have the same SS#.
Automobile; entity set

Attributes:
- Model
- Year
- Weight

Constraints
- Primary Key: Model, Year

Note: Automobile is a “catalog entry”
- It is not a specific “physical car”
Our ER Diagram
Likes; relationship

Relationship among/between:
  » Person
  » Automobile

Attributes

Constraints
Our ER Diagram

[Diagram showing relationships between entities such as Name, Person, Child, Automobile, Model, Year, Weight, ID#, SS#, DOB, Age, Likes, Horse, and FN, LN.]
- This relationship has no attributes
- This relationship has no constraints
- This relationship is a general many-to-many relationship (as we have not said otherwise)
- This relationship does not have any cardinality constraints
- **Car**: entity set

- Attributes
  - **VIN**
  - **Color**

- Constraints
  - Primary Key: VIN

- **Note**: Car is a “physical entity”
  - VIN stands for “Vehicle Identification Number,” which is like a Social Security Number for cars
Our ER Diagram

- **Model**, **Year**, **Weight**, **Like** (Child)
- **Child** with attributes: **ID**, **SS#, DOB, Age**
- **Person** with attributes: **FN, LN, Name, DOB, Age**
- **Automobile** with attributes: **Model, Year, Weight**
- **Car** with attributes: **VIN, Color**
- **Horse** with attributes: **Name**
Type; relationship

Relationship among/between:

- Automobile
- Car

Attributes

Constraints

Cardinality: 1..1 between Car and Type

This tells us for each physical car what is the automobile catalog entry of which it is an instantiation

Each car is an instantiation of a exactly one catalog entry
Our ER Diagram

- **Person**
  - FN
  - LN
  - SS#
  - ID#
  - Name
  - DOB
  - Age

- **Automobile**
  - Model
  - Year
  - Weight
  - Likes
  - Type
    - Car
      - VIN
      - Color

- **Horse**
  - Name
We see that the relationship Type is:
- Many to one from Car to Automobile
- It is total not partial

In other words, it is a total function from Car to Automobile

Not every Automobile is a “target”

There may be elements in Automobile for which no Car exists
\textbf{Has}; relationship

Relationship among/between
  
  » \textit{Person}
  » \textit{Car}

Attributes
  
  » \textit{Date}

Constraints
  
  » Cardinality: 2..* between Person and Has
  » Cardinality: 0..1 between Car and Has

Date tells us when the person got the car

Every person has at least two cars

Every car can be had (owned) by at most one person
  
  » Some cars may have been abandoned
Our ER Diagram

- **Person**
  - Name
  - SS#
  - ID#
  - Age
  - DOB
  - Date
  - Likes
- **Automobile**
  - Model
  - Year
  - Weight
  - VIN
  - Color
  - Type
- **Car**
  - Has 0..1
  - Date 0..1
  - Type 1..1
- **Horse**
  - Name
- We see that Has is a partial function from Car to Person

- Every Person is a “target” in this function (in fact at least twice)
- **Student**: entity set
- Subclass of Person
- Attributes
  - GPA
- Constraints
- Note that Student is a weak entity
  - It is identified through a person
  - You may think of a student as being an “alias” for some person
  - “Split personality”
Our ER Diagram
- **Professor**; entity set
- Subclass of Person
- Attributes
  - **Salary**
- Constraints
Our ER Diagram

- **Person**
  - ID#
  - SS#
  - Name
  - DOB
  - Age
  - ISAs
  - Salary
  - GPA

- **Automobile**
  - Model
  - Year
  - Weight
  - Likes
  - ID#
  - Car
    - Type
    - VIN
    - Color
    - Has
  - Horse

- **Student**
  - Student

- **Professor**
  - Professor

- **Horse**
  - Name
Course; entity set
Attributes:
- C#
- Title
- Description
Constraints
- Primary Key: C#
Our ER Diagram

- **Person**:
  - Name
  - SS#
  - ID#
  - Date
  - GPA
  - Salary
  - ISA
  - Student
  - Professor

- **Automobile**:
  - Model
  - Year
  - Weight
  - Likes
  - VIN
  - Color
  - Type

- **Car**:
  - 0..1
  - Has

- **Horse**:
  - Name

- **Course**:
  - C#
  - Title
  - Description
Prereq

- **Prereq**; relationship

- Relationship among/between:
  - Course; role: First
  - Course; role: Second

- Attributes

- Constraints

- We have a directed graph on courses, telling us prerequisites for each course, if any
  - To take “second” course every “first” course related to it must have been taken previously
  - We needed the roles first and second, to be clear
  - Note how we model well that prerequisites are not between offerings of a course but catalog entries of courses
Book; entity set

Attributes:
- Author
- Title

Constraints
- Primary Key: Author, Title
Our ER Diagram
- **Required**; relationship
- Relationship among/between:
  - Professor
  - Course
  - Book
- Attributes
- Constraints
- A professor specifies that a book is required for a course
- Note that there are no cardinality or other restrictions
- Any professor can require any book for any course and a book can be specified by different professors for the same course
- A book does not have to required for any course
**Section**; entity set

**Attributes:**
- Year
- Semester
- Sec#
- MaxSize

**Constraints**
- Discriminant: Year, Semester, Sec#
- Identified through relationship Offered to Course
- Each Course has to have at least one Section (we do not put a course in a catalog unless it has been offered at least once)
Section

- Section is a weak entity
- It is related for the purpose of identification to a strong entity Course by a new relationship Offered
- It has a discriminant, so it is in fact identified by having the following specified
  C#, Year, Semester, Sec#
- Our current section is:
  G22.2433, 2011, Fall, 001
Offered

- **Offered**: relationship
- Relationship among/between:
  - Course
  - Section
- Attributes
- Constraints
  - Course has to be related to at least one section (see above)
  - Section has to be related to exactly one course (this automatically follows from the fact that section is identified through exactly one course, so maybe we do not need to say this)
- **Took**; relationship
- Relationship among/between
  - Student
  - Section
- Attributes
  - Grade
- Constraints
  - Cardinality: 3..50 between Section and Took (this means that a section has between 3 and 50 students)
Taught

- **Taught**: relationship
- Relationship among/between
  - **Professor**
  - **Section**
- Attributes
- This tells us which professor teach which sections
  - Note there is no cardinality constraint: any number of professors, including zero professors can teach a section (no professor yet assigned, or hypothetical situation)
  - If we wanted, we could have put 1..* between Section and Taught to specify that at least one professor has to be assigned to each section
Our ER Diagram
We want to think of Taught as an entity
» We will see soon why
Monitors

- **Monitors**: relationship
- Relationship among/between
  - **Professor**
  - **Taught** (considered as an entity)

- Attributes

- Constraints
  - Cardinality: 0..1 between Taught and Professor

- This models the fact that Taught (really a teaching assignment) may need to be monitored by a professor and at most one professor is needed for such monitoring
  - We are not saying whether the professor monitoring the assignment has to be different from the teaching professor in this assignment (but we could do it in SQL DDL, as we shall see later)
Our ER Diagram

- **Person**: ID# (Unique), SS#, Name, DOB, Age, Child, ISA, GPA, Salary
- **Automobile**: Model, Year, Weight, Likes, VIN, Color
- **Car**: Type, 1..1, Likes, 0..1, Has, 2..*
- **Student**: Name, Horse, Grade, Took, MaxSize, Sec#, Year, Semester
- **Professor**: Name, ISA, Salary, GPA, ISA
- **Book**: Title, Author, Required
- **Course**: Title, C#, Prereq, First, Second, Offered, 1..*
- **Section**: Sec#, Year, Semester, 1..1, Offered, 1..*
- **Horse**: Name, Date

Relationships:
- Person ISA Professor
- Person Likes Automobile
- Person Has Car
- Person ISA Student
- Person ISA Course
- Person ISA Section
- Person ISA Book
- Person ISA Professor
- Person ISA Course
- Person ISA Section
- Person ISA Book
- Person ISA Professor
- Person ISA Course
- Person ISA Section
- Person ISA Book
What Can We Learn From The Diagram?

- Let’s look
- We will review everything we can learn just by looking at the diagram
We now observe that GPA should probably be modeled as a derived attribute, as it is computed from the student’s grade history.

So, we may want to revise the diagram.
Some Constraints Are Difficult To Specify

- Imagine that we also have relationship Qualified between Professor and Course specifying which professors are qualified to teach which courses.
- We probably use words and not diagrams to say that only a qualified professor can teach a course.
An ER diagram should be annotated with all known constraints
There is a natural hierarchy for our ER diagram

It shows us going from bottom to top how the ER diagram was constructed

Section and Offered have to be constructed together as there is a circular dependency between them

Similar issue comes up when dealing with ISA
Hierarchy For Our ER Diagram

Note: circular dependency, need to be treated together

Note: circular dependency, need to be treated together

Note: circular dependency, need to be treated together
1. Session Overview
2. Enterprise Data Modeling Using the ER Model
3. Enterprise Data Modeling Using the EER Model
4. Case Study
5. Summary and Conclusion
Summary

- Basic ER model concepts of entities and their attributes
  - Different types of attributes
  - Structural constraints on relationships
- ER diagrams represent E-R schemas
- UML class diagrams relate to ER modeling concepts
- Enhanced ER or EER model
  - Extensions to ER model that improve its representational capabilities
  - Subclass and its superclass
  - Category or union type
- Notation and terminology of UML for representing specialization and generalization
Assignments & Readings

- Readings
  - Slides and Handouts posted on the course web site
  - Textbook: Chapters 3 & 4

- Assignment #3 due on 02/25/16
  - Textbook exercises: 3.19, 3.22, 3.25, 3.30, 4.19, 4.21, 4.26

- Project Framework Setup (ongoing)
We will learn how to take an ER diagram and convert it into a relational database

We will learn how to specify such databases using CA’s ErWin or another tool

» These tools allow you to produce a mapping of an ER model to a corresponding relational logical model

» These tools are also used by Enterprise Data Architects to automatically generate SQL DDL code for various database systems
Any Questions?