A model is a simplified abstraction of a real-world phenomenon that is used to promote informed decision-making. Modeling is the activity of abstracting information about a real-world phenomenon and developing it into a model that contains the essential features of that phenomenon with respect to requirements given to the modelers.

• It is assumed here that the requirements have been established before the modeling begins.

When the primary focus of a model is on the structural aspects of a phenomenon rather than its behavioral aspects we refer to the model as a data model in recognition of the role of data in representing the structures. For complex phenomena the volume of data involved will usually require a large, persistent data storage capability, which in turn will necessitate the use of a database management system. For this reason data modeling has become synonymous with database design. Furthermore, instead of using the term “phenomenon” to describe what is being modeled in a data model we shall use the term “enterprise.”

In this course we shall examine data models at three levels:

1. Conceptual data models

   These models capture the user’s perceptions of the data in a software system without any regard for how it might eventually be incorporated in a data storage system.

   In this course we shall focus on semantic data models, examining in particular the entity relationship model (ER) and the object-oriented model.

2. Logical (or implementation) data models

   These models represent the organization of data in a way that will yield a direct implementation in a database management system, but avoid any details of how the data should actually be stored in the data storage system.

   The primary implementation models we shall examine in this course are the relational model, the object-relational model, and the object-oriented model.

3. Physical data models

   These models describe the choices used for the storage of data on secondary storage devices and for accessing the data in terms of clustering, partitioning, indexing, etc.

   In this course we shall be primarily focus on indexing and hashing structures.
Section 2. Fundamental Concepts of Conceptual Data Models

In this section we present the fundamental concepts underlying most conceptual data models. We shall then elaborate and expand on these ideas in later subsections and sections.

1. Entities

A conceptual data model attempts to represent an enterprise as a collection of entities (also known as instances or objects). In any formal system there will always be terms that must be accepted as undefinable, and for us and conceptual data models entity will be such a term. Intuitively an entity is simply one of the “things” about an enterprise that one wants to represent. Some entities have tangible counterparts such as a person, a place, or a thing, but others may be conceptual such as a loan or a job assignment.

In conceptual data modeling it is more common to work with collections of similar entities rather than the entities themselves. Such a collection of similar entities is known as an entity set (or class when we use the term object instead of entity).

2. Relationships and Roles

A relationship is an association among two or more entities. For example in a data model for a university we may have the relationship that faculty member “George Pothering” is a member of the “Computer Science Department”

As with entities, we usually collect similar relationship into a relationship set and use the relationship set to describe an association between entity sets. Each relationship is then an element (or instance of) a relationship set. This the relationship that faculty member “Pothering” is a member of the “Computer Science Department” given above may be regarded as an instance of the relationship set “MemberOf” between the entity sets FACULTY and DEPARTMENT.

It is common practice to use “entity” in place of “entity set” and “entity instance” instead of “entity.” Likewise “relationship” is used instead of “relationship set” and “instance of relationship” in place of “relationship.” We shall use this convention as well.

It is also common to sometimes further clarify the purpose of an entity in a relationship by assigning it a role in the relationship. This is especially true in relationships involving an entity and itself. For example given a relationship IsMarriedTo between the entity PERSON and itself, we may assign one entity the role HusbandOf and the other entity the role WifeOf.

3. Attributes, Values and Domains

Earlier we described an entity (set) as a collection of “similar” entity instances. What characterizes these similarities are a set of properties that are common to each entity instance, though in data modeling we use the term attribute rather than property. In a conceptual model for a university, for example each instance of the FACULTY entity may share the attributes Name, Degree, Degree Area, and Rank with every other instance of FACULTY. What may be different about each entity instance is the value associated with each of these attributes. For example, one instance may have the values George Pothering, Ph.D., Mathematics, and Professor for these attributes respectively, while another instance may be one with the attribute values Anthony Leclerc, Ph.D., Computer Science, and Associate Professor.

The values assignable to attributes are drawn from a pool of candidate values known as the attribute’s domain. Unfortunately in data modeling there can a lot of confusion surrounding the notions of attributes, domains, and the additional concept of a data type. We will use a few examples to illustrate some of the sources of confusion

- Consider two entities PERSON and DEPARTMENT. In associating attributes with these entities one would most likely associate a “name” attribute with each entity, though what each name attribute is actually called is usually up to the discretion of the modeler — use “Name” for both, use Name for PERSON and DeptName for DEPARTMENT, use PersonName for PERSON and DeptName for DEPARTMENT, etc.
• Suppose the data modeler decides to give both PERSON and DEPARTMENT an attribute called Name. How do we designate the domains of each attribute?

a. We could explicitly designate all the elements for each domain. This could work for domain of the Name attribute for DEPARTMENT since this set would probably be small, but would be impractical for the domain of the Name attribute of PERSON.

b. We could classify the structure of the values of each domain by its data type, but not enumerate each value. For example we might describe the domain of the Name attribute of PERSON as the set of strings of characters and we could also use this same description for the domain of the Name attribute of DEPARTMENT. This would mean that we are willing to allow a person to have “Computer Science Department” for a name, or for a department to be called “George Pothering.” More significantly, suppose we have an attribute E-Mail Address for PERSON and also describe its domain as the set of strings of alphabetic characters. With this method of domain description we would be permitting more questionable equivalences.

c. We could specify domains with the names PersonNameDomain, DepartmentNameDomain, and EmailDomain and specify the data type of each as the set of strings of characters. We could then make the following associations:

\[
\begin{align*}
\text{Name attribute of PERSON} & \rightarrow \text{PersonNameDomain} \\
\text{Name attribute of DEPARTMENT} & \rightarrow \text{DepartmentNameDomain} \\
\text{Email Address attribute of PERSON} & \rightarrow \text{EmailDomain}
\end{align*}
\]

At this point we are not going to say anything more about this except to note that we are going eventually adopt an approach similar to that shown in c. above.

We conclude our current discussion of attributes by noting that while we have been concentrating on attributes of entities, it is also possible for relationships to have attributes. For example consider a data model for a university with entities STUDENT and COURSE and a relationship EnrolledIn between these entities. If a student can take a course more than once then we might use the term in which a given student enrolled in a course to distinguish between repeated enrollments. An attribute Term however is neither an attribute of STUDENT nor an attribute of COURSE, since it depends on both the student and the course involved. It is thus more properly seen as an attribute of the EnrolledIn relationship.

4. Identification and Primary Keys

In the physical world it is possible for two objects to exist and yet agree on all of their fundamental properties (in the case of living things, even at the level of the structure of their DNA). Likewise, if an object changes a value of any its properties it does not become a new object. We shall leave it to the philosophers to address the nature of identity as it pertains to the physical world; when it comes to data modeling, however, it is also necessary to be able to distinguish entity instances. This is usually accomplished via an entity’s attributes by one or more of the following:

a. Search for an attribute for which it is known that no two entity instances will have the same value for this attribute.

b. Search also to see if a group of attributes can be used to identify (that is, distinguish) entity instances. This means that two entity instances may agree on some attribute values in this group, but not on all of them. This group should also be minimal in the sense that if any attribute is dropped from the group then the remaining attributes are not sufficient to distinguish entity instances.

c. If no identifying attribute or group of attributes can be found, associate an artificial identifying attribute with the entity.
Each of the identifying attributes or groups of attributes established in steps a, b, or c above is known as a candidate key for the entity. One of these candidate keys is selected to serve as the primary key for the entity.\(^1\)

**Example:** A data model for a university, might use an entity **DEPARTMENT** with attributes Name, Building, Room and Phone. Assuming no two departments in the university have the same name, then the attribute Name is sufficient to identify instances of **DEPARTMENT**, and hence is a candidate key for **DEPARTMENT**.

**Example:** A data model for a symphony orchestra is intended to capture data about the orchestra and its schedules of concerts. A possible entity for this model is **COMPOSITION**, with the attributes Composer, Title, and Length. Possible instances for this entity are:

(Beethoven, Violin Concerto in D, 38:30)
(Brahms, Violin Concerto in D, 34:16)
(Beethoven, Symphony No. 9 in D-Minor, 62:23)

These entity instances demonstrate that neither Composer alone nor Title alone suffice to identify individual compositions, and although it is not apparent from the instances given, it should be clear that Length alone is also insufficient to identify compositions. If we make the reasonable assumption, however, that no composer would have two compositions with the same title, then the set of attributes \{Composer, Title\} will be sufficient to identify instances of **COMPOSITION**.

**Example:** A data model for a car rental company might use an entity **VEHICLE** with attributes VehicleID, Year, Make, Model, and License Number. Here either of the attributes VehicleID or License Number could serve an identifying attribute for **VEHICLE**.

---

*Rigorous Definitions of Fundamental Data Modeling Constructs*

If we accept as our starting point the existence of entity (set) \(E\), then an attribute \(A\) of \(E\) is a function \(A: E \rightarrow D_A\), where \(D_A\) is the value set associated with attribute \(A\).\(^2\)

Given entities \(E_1, E_2, ..., E_n\), a relationship \(R\) among these entity sets is a subset \(R \subseteq E_1 \times E_2 \times ... \times E_n\).

Let \(E\) be an entity set with attributes \(A_1, ..., A_n\) and let \(K = \{A_{k1}, ..., A_{kp}\}\) where each \(A_{kj}\) \(\in\) \{\(A_1, ..., A_n\)\} and no two \(A_{kj}\)’s are the same. Then \(K\) is a candidate key for \(E\) if given \(e_1 \in E\) and \(e_2 \in E\) and \(e_1 \neq e_2\) then \(A_t(e_1) \neq A_t(e_2)\) for some \(A_t \in K\).

---

**Section 3. Introduction to the Entity Relationship Model**

This section will be quite brief. We shall simply introduce the most elementary symbols for the entity-relationship model. In later sections we shall introduce additional data modeling concepts and modify our notation as necessary.

The entity-relationship (ER) model, initially proposed by Peter Chen in 1976,\(^3\) still remains the leader as a conceptual data model. It has been modified and extended by others\(^4\) since then, however.

---

\(^1\) Sometimes when groups of attributes provide the only naturally occurring candidate keys, to simplify identifying entity instances an artificial identifying attribute may be defined and designated as the entity’s primary key.

\(^2\) We use the expression “value set” instead of “domain” to avoid confusion with the terms “domain” and “range” as they pertain to mathematical functions.

In the E-R model an entity is represented by a rectangle and the name of the entity is placed inside the rectangle. A relationship is represented by a diamond, with the name of the relationship appearing either inside the diamond or, more commonly, directly above or below the diamond. Emanating from the sides of the diamond are lines going to each entity participating in the relationship. Each line may be labeled with the name of the role for its associated entity in the relationship.

Attributes are represented by ovals, with the name of the attribute appearing inside the oval. Each attribute oval is connected to its associated entity rectangle, or relationship diamond by a line.

In the entity-relationship model all entities must have a primary key. The names of all attributes that participate in the primary key are underlined.

**Examples:**
1. The **FACULTY** entity on page 2.
2. The **DEPARTMENT** entity on page 4.
3. The **COMPOSITION** entity on page 4.
4. The **VEHICLE** entity on page 4.
5. The **EnrolledIn** relationship on page 3.
6. The **IsMarriedTo** relationship on page 2.

**Compound Attributes, Multi-valued Attributes, and Derived Attributes**

Given an attribute \( A \) with domain \( D_A \), then depending on the structure of \( D_A \) the attribute \( A \) could be *compound*, meaning \( D_A \subseteq D_{A1} \times \ldots \times D_{Ap} \) for some domains \( D_{A1}, \ldots, D_{Ap} \). We call \( A_1, \ldots, A_n \) the *components* of \( A \).

**Example:** Suppose we are given an entity set PERSON which among its other attributes has an attribute Address. In some data models it may be desirable to structure Address as a compound attribute with components Street, City, State, and Zip. Likewise, if PERSON has an attribute Name, it may be desirable to structure Name as a compound attribute with components FirstName, MiddleName, and LastName.

An attribute \( A \) may be *multi-valued*, meaning for a given entity instance \( e \), \( A(e) \in \mathcal{P}(D_A) \), where \( D_A \) is the domain of \( A \) and \( \mathcal{P}(D_A) \) is the power set of \( D_A \).

**Example:** Suppose in a data model for a university we have an entity STUDENT and give it an attribute Major. If the university allows a student to pursue more than one major at the same time, then Major will be a multi-valued attribute.

In the entity-relationship model we represent the compound attribute and each of its components with labeled ovals. Each of the component ovals is connected to the oval with the name of the compound attribute with a line.

---

4 See, for example Teorey, T.J., Yang, D., and Fry, J.P. “A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model,” *ACM Computing Surveys* 18, 2 (June 1986), pp. 197-222.
A multi-valued attribute is represented by using a double line to connect its oval to the associated entity or relationship.

![Diagram: STUDENT connected to Phone with a double line]

Finally, we note that some entities may have an attribute such that, for each instance of the entity, the values for the attribute may be calculated from other attribute values for the instance. Such an attribute is known as a derived attribute.

**Example:** Consider an entity PERSON with attributes BirthDate and Age. The attribute Age can be a derived attribute since in a given instance of PERSON its value can be determined from that of BirthDate.

A derived attribute is represented by using a dashed line to connect its oval to the associated entity or relationship.

![Diagram: PERSON connected to BirthDate and Age with dashed lines]

**First ER Model**

We now develop an ER model for a simple data model of a company. This model will only use the concepts we have discussed up to this point. In developing our model we shall use the following approach:

1. **Identify potential entities and attributes.**
   
   Where there’s a narrative describing the enterprise and the requirements for the data model, one can often find potential attributes and entities from the nouns and noun phrases in the narrative.

2. **Identify potential relationships**
   
   Where there’s a narrative describing the enterprise and the requirements for the data model, one can often find relationships from the verb phrases in the narrative.

3. **Determine entities, assign attributes, and define relationships.**
   
   The following suggestions may be helpful:

   - Find any synonyms and use one term or expression in place of all the synonyms.
   - Possessive forms and “has a” relationships often indicate where attributes for an entity may be prescribed. If the item possessed (or the object of the “has a”) is another entity, however, then a relationship is probably indicated.
   - Where you have several verb phrases suggesting the same relationship, determine if some of these verb phrases are identifying roles, especially where an entity is being related to itself.
   - Since the ER model requires that primary keys be specified, be alert for qualifiers such as “unique,” “exactly one,” etc. which, when associated with attributes, suggest potential candidate keys. If there are no obvious candidate keys for an entity, an artificial attribute may have to be assigned to serve as a primary key.
   - An attribute A that appears in several entities, $E_1, \ldots, E_n$ may be reclassified as an entity $E_A$. This would also entail establishing relationships between each of $E_1, \ldots, E_n$ and the new entity $E_A$. 
• If an entity has \( E_1 \) has a single attribute \( A \) and is related to just one other entity \( E_2 \), then \( E_1 \) could be dropped and \( A \) made an attribute of \( E_2 \).

4. Specify domains, including their data type and where feasible its range of possible values

Tables are often used for this.

Example: The COMPANY data model keeps track of a company’s employees, departments, and projects. Suppose that as a result of a requirements analysis the following description of the aspects of the company to be represented in the data model results:

1. The company is organized into departments and each department has a unique name and main office location. The department has a particular employee who manages the department. We should keep track of the date when that person started managing the department. We should also keep track of all of the offices that are assigned to the department. Each office has a building name, room number and a telephone.

2. A department controls a number of projects, each of which has a unique number within the company. Each project also has a description, employee who serves as its director, and budget. The company also wants to keep track of all employees who are assigned to each project.

3. We want to keep track of each employee’s name, social security number, address, salary, sex, birthdate, and starting date of employment with the company. Each employee belongs to a department but may work on projects that are controlled by departments other than the one to which he or she is assigned. We also want to keep track of the number of hours each employee has worked on each project.

Develop an ER model for this enterprise.

(a solution, as well as a sketch of its development, is available in the file CompanyER1.ppt)

Strong and Weak Entities

There are cases where we shall want to modify our requirement that all entities must have among their attributes an attribute or set of attributes that can serve as a primary key for the entity.

• Consider a data model for a company that has entity sets EMPLOYEE and DEPENDENT. While the EMPLOYEE entity may use an attribute SSN as a key attribute, it may not be important for the model to give the entity DEPENDENT a similar attribute. Instead we may use an attribute Name as a partial identifier for an instance of DEPENDENT and rely on the fact that a dependent will only exist in the model if it is associated with an instance of EMPLOYEE. We can then use the SSN value of the associated employee and the name of the dependent to uniquely identify the employee\(^5\). Thus, there may be several DEPENDENT entities with the name Joe, but it’s through each of their associations with a specific EMPLOYEE entity that they acquire their full identity – the Joe who is the dependent of employee 123456789 versus the Joe who is the dependent of employee 987654321.

An entity that requires a relationship to other entities in order to identify its instances is known as a weak entity, while an entity that does not require such a relationship for full instance identification is known as a strong entity (or regular entity). The relationship that serves to identify instances of weak entities is known as an identifying relationship.

• Weak entity sets are represented in an ER model with a double rectangle and the identifying relationship is represented using a double diamond\(^6\).

\(^5\) We are assuming here that an employee will not have two dependents with the same name.

\(^6\) Some authors do not represent the identifying relationship explicitly and instead simply connect the weak entity with its identifying entities by single lines.
Section 4. Properties of Relationships

In this section we further develop the concept of a relationship by describing several properties of a relationship.

1. **Degree**: The *degree* of a relationship is the number of entities participating in the relationship. A relationship of degree 2 is known as a *binary* relationship and a relationship of degree 3 is known as a *ternary* relationship. Relationships of degree $n > 3$ are generally referred to as *n-ary* relationships. If an entity participates more than once in a relationship the relationship is said to be a *recursive relationship*. The most common recursive relationships are binary, meaning that an entity is being associated with itself.

2. **Connectivity Constraints**: The *connectivity constraint* of a relationship (or simply the *connectivity* of the relationship) specifies generally the number of relationship instances that instances of the related entities can participate in.

   **Connectivity Constraints for Binary Relationships**: For binary relationships, connectivity constraints are one of three types:

   a. *One-to-one* (1:1): A relationship $R$ between $E_1$ and $E_2$ is one-to-one if an entity instance $e_1 \in E_1$ can be related to at most one instance $e_2 \in E_2$ and an entity instance $e_2 \in E_2$ can be related to at most one instance $e_1 \in E_1$

   b. *One-to-many* (1:N): A relationship $R$ between $E_1$ and $E_2$ is one-to-many if an entity instance $e_1 \in E_1$ can be related to several instances $e_i \in E_2$, but an entity instance $e_2 \in E_2$ can be related to at most one instance $e_1 \in E_1$.

      - Here the positioning of the entities $E_1$ and $E_2$ in the description of $R$ is important. We can also say that $R$ is *many-to-one* between $E_2$ and $E_1$.

   c. *Many-to-many* (N:M): A relationship $R$ between $E_1$ and $E_2$ is many-to-many if an entity instance $e_1 \in E_1$ can be related to several instances $e_i \in E_2$, and if an entity instance $e_2 \in E_2$ can be related to several instances $e_k \in E_2$.

   **Connectivity Constraints for n-ary Relationships**: Interpreting connectivity constraints for n-ary relationships is more complicated than in the binary case. Given a relationship $R$ between the entities $E_1, E_2, \ldots, E_n$ an entity $E_k$ is considered to be “one” in the relationship if only one instance of $E_k$ can be associated with one instance of each of the $k-1$ associated entities $E_1, \ldots, E_{k-1}, E_{k+1}, \ldots, E_n$. The entity $E_k$ is considered to be “many” in the relationship if several instances of $E_k$ can be associated with one instance of each of the $k-1$ associated entities $E_1, \ldots, E_{k-1}, E_{k+1}, \ldots, E_n$.

**Examples of Connectivity Constraints**

**Example**: Consider a data model for a health club. Among the entities one might identify for this model is one for **PATRON** and one for **LOCKER**. The assignment of lockers to patrons can be represented by a relationship **LockerAssignment**. If we assume that no two
patrons share a locker and that no patron will be assigned more than one locker, then
**LockerAssignment** is a one-to-one relationship between **PATRON** and **LOCKER**.

**Example2**: Consider a data model for a college that has entities **DEPARTMENT** and **SCHOOL** and a relationship **BelongsTo** between **DEPARTMENT** and **SCHOOL** to represent the organization of the college in terms of departments within schools. If we assume that no department can belong to more than one school, then **BelongsTo** is a one-to-many relationship between **SCHOOL** and **DEPARTMENT**, or a many-to-one relationship between **DEPARTMENT** and **SCHOOL**.

**Example3**: Continuing with our data model for a college, consider the relationship **Enrollment** between the entities **STUDENT** and **COURSE** that represents the enrollment of students in courses. This is a many-to-many relationship since a student can take more than one course and a given course enrolls more than one student.

**Example4**: In a data model an airline, consider a relationship **FlightAssignment** among the entities **FLIGHT**, **PLANE** and **PILOT** to represent plane and the individuals that were assigned to a given flight (identified by flight number and date). This relationship is many-to-one-to-many since:

- Given a plane and a pilot, there may be many flights on which the pilot flew that plane.
- Given a flight and a plane, there may be several pilots assigned to that flight (on that plane).
- Given a flight and a pilot, there is only one plane that is assigned to that flight (with that pilot).

3. **Cardinalities**: Cardinalities are traditionally defined only for binary relationships and refine connectivity constraints by indicating the minimum and maximum number of entity occurrences that can be associated with one occurrence of a related entity. Thus, given a relationship R between E₁ and E₂, if an instance e₁ of E₁ can be related to at most q entities of E₂, but must be related to at least p, then E₂ participates in the relationship R with a *minimal cardinality* of p and a *maximum cardinality* of q.

**Example1**: For the previous **Example1**, the maximum cardinality in each case is 1. It is reasonable to assume that a patron does not have to be assigned a locker in order to be a member of the health club; likewise not all lockers may be assigned to patron, hence the minimum cardinality for each is zero.

**Example2**: In **Example2** under connectivity, if we assume that each department must belong a school, then the minimum cardinality for **SCHOOL** is one. On the other hand, a given school need not have any departments belonging to it (Graduate Schools are one example) so the minimum cardinality for **DEPARTMENT** is zero. If we do not specify an upper limit for the number of departments that can belong to a given school, we simply designate the maximum cardinality for **SCHOOL** as N.
Example3: In Example3 from connectivity, new students may not yet be assigned any courses and new courses may not yet have been offered, whence the minimum cardinality for each is zero. On the other hand, if there is no specific upper limits on the number of courses a student may take or the number of students in a course, the maximum cardinality for each is N.\footnote{Using the common symbol N for each does not imply that both entities have the same maximum cardinality in the relationship.}

4. **Structural Constraints on Entities in Relationships:** Besides their degree and connectivity, relationships may have constraints that dictate how instances of the associated entities may arise vis-a-vis the relationship. These go under the general categorization of structural constraints. Typically these structural constraints are only discussed with respect to binary relationships.

   a. **Participation Constraints:** Given a (binary) relationship $R$ involving entities $E_1$ and $E_2$, a participation constraint on $E_1$ with respect to $R$ specifies whether the existence of an instance of $E_1$ depends on it being related to an instance of $E_2$. If every instance of $E_1$ must be related to an instance of $E_2$ via $R$, the participation of $E_1$ in $R$ is said to be \textit{total}; otherwise $E_1$’s participation is \textit{partial}.

      \textbf{Example:} Consider a relationship $\text{IsManagedBy}$ between entities $\text{EMPLOYEE}$ and $\text{DEPARTMENT}$. If every department must have an employee who serves as manager, then we cannot have a $\text{DEPARTMENT}$ instance in our data model unless there is a $\text{EMPLOYEE}$ instance with which that department can be associated. In this case, the participation of $\text{DEPARTMENT}$ in the $\text{IsManagedBy}$ relationship is total. On the other hand, the participation of $\text{EMPLOYEE}$ is partial since not every $\text{EMPLOYEE}$ has to manage a $\text{DEPARTMENT}$.

      An entity whose participation in a relationship $R$ is total has a minimum cardinality of at least one, while an entity whose participation is partial has a minimum cardinality of zero.

   b. **Existence Dependency:** Given an entity $E$, if there is a (binary) relationship $R$ between $E$ and another entity $E^*$ such that the participation of $E$ in $R$ is total, then $E$ is said to be existence-dependent \textit{on} $E^*$ \textit{(because of} $R$) since every instance of $E$ requires an instance of $E^*$ to which it is related by $R$. If $E$ is existence-dependent on $E^*$ because of relationship $R$, then $E^*$’s existence in $R$ is \textit{mandatory}; otherwise its existence is \textit{optional}.

      \textbf{Example:} Consider the relationship $\text{IsManagedBy}$ between entities $\text{EMPLOYEE}$ and $\text{DEPARTMENT}$ from above. Here $\text{DEPARTMENT}$ is existence-dependent on $\text{EMPLOYEE}$ because of $\text{IsManagedBy}$. Furthermore $\text{EMPLOYEE}$’s existence in $\text{IsManagedBy}$ is mandatory, while $\text{DEPARTMENT}$’s existence is optional.

   \textit{Existence Dependency and Weak Entities:} Earlier we introduced the concept of a weak entity. In this case the weak entity $W$ is an existence-dependent since it depends on
another entity for a portion of its primary key. Not every entity with an existence dependency is a weak entity, however. Consider a data model for a bank and suppose it has entities CUSTOMER and LOAN. Since loans typically use an institute-wide loan number that can serve as a primary key, the entity LOAN is not a weak entity. On the other hand, one would not create an instance of LOAN unless it is for a CUSTOMER, so LOAN has an existence dependency even though it is not a weak entity.

Section 5. Relationship Constraints and the Entity-Relationship Model

It is unfortunate that there is no standard for representing connectivity and existence constraints in the entity-relationship model. Making things even worse, two different authors may use similar notations but give them different meanings. In this course we shall generally use the notation illustrated on pages 81, 82, 84, and 86-89 of your textbook.

The following diagram shows another common notation (the “crow’s feet” notation) for incorporating structural constraints (including min-max pairs) into the ER model.

**Example1:** University ER diagram redrawn with structural constraints (to be completed in class).
Example 2: Company ER diagram redrawn with structural constraints (to be completed in class).

Section 6. Higher Order Abstraction in Data Modeling: Generalization and Specialization; Aggregation

Generalization and Specialization

Generalization is a process through which one takes a collection of entities and forms from them a new entity, known as their supertype (or parent) entity, embodying properties common to the entities being generalized. Thus, differences among instances of the entities in the collection being generalized are ignored and their similarities emphasized. Conversely, specialization starts with a given entity and derives from it other entities, known as subtype (or child) entities that, in addition to all of the properties of the given entity, have additional characteristics which distinguish each one’s instances from other instances of the given entity.

Example: A data model for a publishing company may have initially identified entities AUTHOR, EDITOR, and REVIEWER. These may be generalized, however, to a supertype PERSON.

Example: A data model for a college or university may have originally identified an entity FACULTY. This class may be specialized into the subtype entities ROSTER and ADJUNCT. The subtype ROSTER may itself be specialized into the subtypes TENURE-TRACK and NON-TENURE-TRACK.

Since both processes yield a supertype entity E and a set of subtypes \{S_1, \ldots, S_n\}, it is common to call the collection E and \{S_1, \ldots, S_n\} a generalization/specialization hierarchy, or simply a hierarchy. The following constraint conditions apply to such hierarchies:
1. **Disjointness constraint**: If an instance of E can belong to at most one of the \( S_i \)'s, then the set of subtypes is said to be a **disjoint specialization**; otherwise they **overlap**.

2. **Completeness constraint**: This specifies whether an instance of E must belong to one of the subtypes. If every member of E must belong to at least one of the subtypes \( S_1, ..., S_n \), the set of subtypes is known as a **total specialization**; otherwise the set of subtypes is a **partial specialization**.

**Example**: In the data model for a publisher above where the entity EMPLOYEE has subtypes AUTHOR, REVIEWER, and EDITOR. These alone may not be a total specialization of EMPLOYEE (for example, typesetters, artists, etc. may also part of the data model) and may overlap (for instance an author may also be a reviewer).

**Example**: In the data model for a college or university, the hierarchy with FACULTY as a supertype and with the entities ROSTER and ADJUNCT as subtypes is both total and disjoint. Furthermore, if ROSTER is itself specialized to the entities TENURE-TRACK and NON-TENURE-TRACK this hierarchy also would be total and disjoint as would the hierarchy arrived at by specializing ADJUNCT into the subtypes FULL-TIME and PART-TIME.

**Aggregation**

Aggregation is an abstraction concept for forming composite entities from base entities. It is primarily to handle two types of situations:

1. where an entity \( P \) serves as an attribute for another entity \( E \). It is common to say that \( P \) has a "part-of" relationship with \( E \), but rather than model this situation as a relationship between \( P \) and \( E \), it is more natural to devise an alternative structure for this type of relationship.

   **Example**: Consider a data model for a computer company that not only manufactures computers, but which also markets local area networks. Among some of the entities that might appear in this system are those representing the network (LAN), its components (CLIENT and SERVER, both of which may be subtypes of the entity COMPUTER), and any installed software (including SYSTEM software and APPLICATION software, both of which may have subtypes of the entity SOFTWARE, and yet which may also have their own subtypes such as OS and UTILITY for SYSTEM, and OFFICE-SUITE, GAME, BROWSER for APPLICATION).

2. where one wants to combine entities \( E_1 \) and \( E_2 \) that are related by a relationship \( R \) into a higher-level entity that can be related to another entity.

   **Example**: Consider a data model that for a job placement organization that stores information about interviews by job applicants to various companies, and any job offers that may result from such interviews. Initially we may identify three entities COMPANY,
APPLICANT, and JOB-OFFER. We further interested in knowing which applicants have had interviews with various companies and those interviews which resulted in job offers.

One might initially be tempted to devise the following relationship between these entities.

It is not appropriate to relate COMPANY, APPLICANT and JOB-OFFER by such a ternary relationship Interview, however, because this would require that we can only record an instance of an interview between applicant a and company c if it resulted in a job offer.

Ideally we would want to model a relationship Interview between COMPANY and APPLICANT and then related JOB-OFFER instances to those interviews that resulted in offers. This would require relating an instance of Interview to one of JOB-OFFER, which is not allowed in the ER model. With aggregation one could treat Interview as an aggregate entity INTERVIEW having component entities COMPANY and APPLICANT and then form a relationship ResultsIn between INTERVIEW and JOB-OFFER.

The Extended Entity-Relationship Model (EER)

Originally the entity-relationship model did not support generalization/specialization or aggregation. The extended entity relationship model (EER; also known as the expanded entity-relationship) extended the representation structures of the entity-relationship model to include support for hierarchies of entities (that is the supertype and subtype entities we described above) and aggregation.

There is no standardized notation for representing generalization/specialization hierarchies or aggregation. We shall adopt the notation shown in your textbook on page 25 for generalization/specialization, and on page 26 for aggregation. Alternate notations, especially for hierarchies, abound.
Section 7  A Design Methodology for Conceptual Data Models Using EER

We now expand on the approach given earlier in this chapter for developing simple ER models and provide the following steps can serve as a guide for developing an EER model for an enterprise. As before, these steps presume that an analysis of the data requirements of the enterprise being modeled has already been undertaken.

Step 1: Classify entities and attributes

It is not always easy to determine whether a specific set of instances in a data model should be modeled as an entity, an attribute, or even as a relationship.

Example: Publishers are located in cities. Should city be an entity or an attribute? Reviewers write reviews of books. Should a review be an entity? An attribute? A relationship between reviewer and book?

The following guides may be useful for distinguishing between entities and attributes:

  a) Entities have descriptive information; attributes do not.

     If an item requires only a name to identify it then it should be modeled as an attribute, but if there is descriptive information available about the item it should be modeled as an entity.

     **Example:** Concerning the cities where publishers are located. If all that is required is the name of the city then the item CITY can be modeled as an attribute. If the city contains additional descriptive information, however, such as the state or country where it is located, its population, etc., then CITY should be modeled as an entity.

  b) Instead of a multi-valued attribute consider using an entity.

     **Example:** Suppose in our publishing example we decide that CITY has no further descriptive information, but the publisher has operations in several cities. In this case we would want to make CITY an entity and then possibly have it participate as the “many” entity in a one-to-many relationship with another entity that is representing the “operation.”

     Also, while multi-valued attributes are permissible in the ER model, later design and implementation decisions usually make it desirable to avoid them. Consequently we shall reclassify multi-valued attributes as entities even if they do not have any descriptors themselves.

  c) Re-classify as an entity any attribute that appears in several entities and identify appropriate relationships with the entities for which they were attributes.

     **Example:** An entity STORE has as attributes StoreNumber and City (and possibly others). Suppose the entity STATE arises in our model also and uses an attribute City that is identical in type to that used with STORE. The attribute(s) City should be re-
classified as an entity and placed in a many-to-one relationship with STATE and in an appropriate relationship with STORE.

d) Associate attributes with the entities they most directly describe.

**Example:** The attribute *OfficeBuilding* probably should be associated with the entity DEPARTMENT instead of the entity EMPLOYEE.

e) Avoid entities with composite keys, if possible.

If an entity has been defined with a compound key consider the following:

i. If the components of the compound key also appear as identifying attributes of other entities, define the entity with the compound key as a weak entity, or possibly reclassify it as a relationship.

**Example:** An entity BOOK-DATA has been defined with *AuthorName* and *BookTitle* as its compound key, and an attribute *PlacesUsed* as a descriptive attribute. If this entity arises in a data model for a college or university and serves to depict the books written by a faculty member, one may want to make this a weak entity to the FACULTY entity. The BOOK-DATA (or perhaps simply BOOK) would retain its attributes *Title* and *PlacesUsed*.

ii. If the components of the compound key are not identifiers of other entities then there are two options:

- Eliminate the entity and define new entities each with a component of the compound key as its primary key. Later, define a relationship to represent the entity that was eliminated.

  **Example:** Consider the BOOK-DATA entity above. If this entity arose in a data model for a library it may be more feasible to eliminate the entity BOOK-DATA, create new entities AUTHOR and BOOK, and then in a later step establish a relationship between BOOK and AUTHOR to represent BOOK-DATA.

- Keep the entity with the compound key if it is reasonably natural.

  **Example:** Consider the entity COMPOSITION with compound key Composers + Work. If no other information is being kept on composer, it may be best to keep this entity as it is.

**Step 2: Identify generalization/specialization hierarchies**

If there is a specialization or generalization hierarchy among entities, then place identifying attributes and generic descriptor attributes with the supertype and place the same identifying attributes and specific descriptor attributes with the subtypes.

**Example:** Suppose the following three entities were identified in a data model:
EMPLOYEE with identifying attribute EmployeeID and descriptors Name, Address, and DateOfBirth
AUTHOR with identifying attribute AuthorID and descriptors AuthorName and Specialty
REVIEWER with identifying attribute IdNo and descriptors RevName, TopicArea, JobTitle, and TeamLeader

Further analysis determines that EMPLOYEE is a generalization of AUTHOR and REVIEWER. As a result we put the identifying attribute EmployeeID and the generic descriptors Name, Address, and DateOfBirth in the supertype EMPLOYEE. We put the identifying attribute EmployeeID and descriptor Specialty with the entity AUTHOR, and we put the identifying attribute EmployeeID and specific descriptors TopicArea, JobTitle, and TeamLeader with the entity REVIEWER.

Step 3: Define relationships

For every relationship the following should be specified: its degree (that is number of entities being related), its connectivity as well as any minimal and maximal cardinalities, and any attributes. In addition we offer the following guidelines for defining relationships:

a) Analyze two or more relationships between the same entities to be sure that they aren’t representing the same thing. The most commonly occurring source of redundant relationships are the so-called transitive dependencies

Example:

b) Define relationships of degree three or higher only if they cannot be represented by several binary relationships.

Example:

Step 4: Integrate multiple views of entities, attributes, and relationships

This will be handled in the next section.

Data Modeling Example

See handout from class.

Section 8. View Integration

An important aspect of conceptual data model development is view integration, which we introduced as step 4 in the design methodology from section 7., but whose discussion we deferred until now.

When developing a data model for a large or complex enterprise it is common for this to be done from several perspectives – several teams of developers may be working on different aspects of
the data model, different individuals or groups of individuals within the enterprise may be consulted for their views of the enterprise, or different end-users of the data model or its ensuing database may be consulted for their perspectives. What will usually result are several ER (or EER) conceptual models representing these different perspectives, and typically employing different terminology.

**Example:** The marketing division of a product’s manufacturer may view the product as a unit, while the engineering division will focus on individual parts of the whole product.

**Example:** Three views of a review for a book.

- There is an entity REVIEW as well as entities REVIEWER and BOOK. There is a one-to-many relationship **MakesReview** between REVIEWER and REVIEW and a one-to-many relationship **ReviewedAs** between BOOK and REVIEW.

- There are entities REVIEWER and BOOK and a many-to-many relationship **Reviews** between REVIEWER and BOOK.

- There are entities REVIEWER and BOOK and a many-to-many relationship **Evaluates** between REVIEWER and BOOK. The relationship has an attribute **Review**

**Example:** Consider the following two descriptions (from Batini, Lenzerini, and Navathe) related to the same enterprise:

- The data of interest is about books. Books have titles. They are published by publishers with names and addresses. Books are adopted by universities having a name and belonging to a state. Books refer to certain topics.

- The data of interest includes publications of different types. Each publication has a title, a publisher, and a list of key words. Each keyword consists of a name, a code, and a research area.

Further analysis or inquiry could lead one to conclude that “topics” in the first ‘description is the same as “keyword” in the second description. On the other hand “publication” in the second description may be a more abstract concept than “book” in the first description, including such additional items as proceeding, journals, monographs, etc.

Experience has shown that nearly every situation involving differences in perspective and terminology of views can be resolved in a meaningful way through integration techniques. We now present a methodology defined by Batini and Lenzerini for integrating various views into a single, coherent model.

---


**Step 1: Choose an Integration Strategy**

Typically the choice is between merging two views at a time, or merging between 2 and the total number of views. Merging just two views at a time is usually more attractive because each merge generally involves a small number of constructs and is easier to conceptualize.

**Step 2: Compare the various ER models**

Compare views to see how entities correspond and to detect conflicts arising from the various views adopting different perspectives. Among some of the conflicts to be alert for are:

- **Naming conflicts:** Synonyms occur when two or more names are given to the same concept, while homonyms occur when the same name is used for different concepts.

  **Example:** Consider a relationship “Owns” between entities DEPARTMENT and EQUIPMENT in a data model for a university. Here “equipment” may refer to computers, copiers, office furniture, etc. Now consider a relationship “Contains” between the entities BUILDING and EQUIPMENT in the same university enterprise, but in a view intended to capture air conditioning equipment, cleaning equipment, classroom furniture, etc. Since two conceptually distinct types of equipment are being modeled here the name EQUIPMENT is a homonym in this context.

  **Example:** In two data models for a credit card company, consider the relationship “Holds” between the entities CLIENT and CREDIT and the relationship “Orders” between the entities CUSTOMER and MERCHANDISE. Here CLIENT and CUSTOMER could be synonyms.

- **Type conflicts:** These involve using different ER constructs to model the same concept.

  **Example:** “City” may be an entity in one view and an attribute in a second view.

- **Dependency conflicts:** These result when users specify different cardinalities for similar, or even the same concept.

  **Example:** A relationship “Marriage” between the entity PERSON and itself (a recursive relationship) may be 1-to-1 in one view, but many-to-many in another that accounts for a marriage history.

- **Domain conflicts:** An attribute may have different domains in two views.

  **Example:** SSN may be a string in one view and an integer in another view.

  **Example:** The attribute Weight may be in units of pounds in one view and kilograms in a second view.

- **Key conflicts:** These arise when different primary keys are assigned to the same entity in different views.

  **Example:** SSN# and Employee-Id, with different domains, may be the primary keys of the entity EMPLOYEE in two different views.
Step 3: Conformation of the various ER models

The basic goal here is to align or conform views to make them compatible for integration.

- Entities may need to be renamed and their primary keys changed.
- Conversion may be required so that concepts modeled as entities, attributes, and relationships appear as only one data model type.
- Relationships sets with equal numbers of participating entities (the degree of the relationship), roles, and cardinalities are easy to merge while those with differing characteristics may be more difficult, if not impossible.

Examples: Several examples worked out in class.

Step 4: Merging and Restructuring of the various ER models

The goals of this step are completeness, minimality, and understandability.

- Completeness: All component concepts appear in the merged views with their semantics intact.
- Minimality: All redundant concepts are removed from the merged model.
- Understandability: The merged model must make sense to the user.

Section 9. Object-Oriented Data Model

The object-oriented data model is a modeling methodology based on object-oriented concepts. Most of these concepts are the same as those we have already studied and differ only in the names we attach to them. There are two important differences, however:

- In the object-oriented approach the fundamental unit is the object, which corresponds with an entity instance. Unlike entity instances, however, objects may encapsulate behavior properties along with its structural properties. In developing object-oriented data models, on the other hand, the behavioral aspects of objects are usually not emphasized.

- In the object-oriented approach the notion of identity is assumed for each object and it is not necessary to seek candidate keys for identifying distinguishing objects.

In the following table we now relate data modeling structures from the EER and object-oriented models
In addition to the object-oriented concepts listed above there are other concepts which do not have EER counterparts, for example the concept of a method which the object-oriented approach uses to attach behaviors with an object.

**Data Modeling in the Object-Oriented Model**

For the most part our approach to developing data models in the object-oriented approach will parallel that used for developing EER models. Although objected-oriented analysis and object-oriented design include steps for capturing an object’s behaviors and incorporating them into class design, these steps are less relevant for data models and hence we do not concern ourselves with them here. On the other hand there are other structures that are more commonly used in object-oriented data models than in EER models and we shall discuss these in the next subsection.

**Representation of Object-Oriented Data Models**

1. **classes**: represented by a three-part box. The top part gives the class name, the second (optional) part lists attributes, and the third (also optional part) lists operations/methods.

<table>
<thead>
<tr>
<th>ClassName</th>
<th>ClassName</th>
<th>ClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Attributes</td>
<td>Methods (will not be used)</td>
</tr>
</tbody>
</table>

If desired one can also specify an attribute’s domain by appending it to the attribute name. The two are separated by a colon.

2. **associations**:
Binary associations are represented by a line between associated entities, with the name of the association centered either above or below the line. Optionally, roles may be specified on the opposite side of the line from the association name and each is located near the class whose role is being described.

Connectivity constraints are sometimes denoted by the presence or absence of a filled or empty circle at the ends of the association line. These are interpreted as follows:

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>No circle (exactly one)</td>
</tr>
<tr>
<td>—○</td>
<td>At most one (possibly none)</td>
</tr>
<tr>
<td>—●</td>
<td>Many (0 or more)</td>
</tr>
<tr>
<td>1:*</td>
<td>One or more</td>
</tr>
</tbody>
</table>

In other cases minimum and maximum cardinalities may be included near a class using the notation \textit{minimum..maximum}. A symbol \text{*} is used to represent “many.” A single value represents a common minimum and maximum cardinality, except for a solitary \text{*}, which means 0..*.

Ternary associations are represented with a diamond as in the EER model, but lack the cardinality designations found in the EER model.