Unit 3
Data Modeling using Entity Relationship Model

- Using High-Level Conceptual Data Models for Database Design
- Entity Types, Entity Sets, Attributes, and Keys
- Relationship Types, Relationship Sets, Roles, and Structural Constraints
- Weak Entity Types
- ER Diagrams, Naming Conventions, and Design Issues
- Relationship Types of Degree Higher Than Two
- Subclasses, Superclasses, and Inheritance
- Specialization and Generalization
- Constraints and Characteristics of Specialization and Generalization

Introduction

- **Conceptual modeling** is a very important phase in designing a successful database application.
- **Entity-relationship (ER) model** is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts.
- The diagrammatic notation associated with the ER model is known as **ER diagram**.
- Extensions to the ER model that lead to the **enhanced-ER (EER) model** includes concepts such as specialization, generalization, inheritance, and union types (categories).

Using High-Level Conceptual Data Model for Database Design

- The first step during database design is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their **data requirements**. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the user defined operations (or transactions) that will be applied to the database.
- The next step is called **conceptual design**. During this step, designers create a **conceptual schema** for the database, using a **high-level conceptual data model**, such as **ER model** that includes detailed descriptions of the **entities**, **relationships**, and **constraints**.

Using High-Level Conceptual Data Model for Database Design

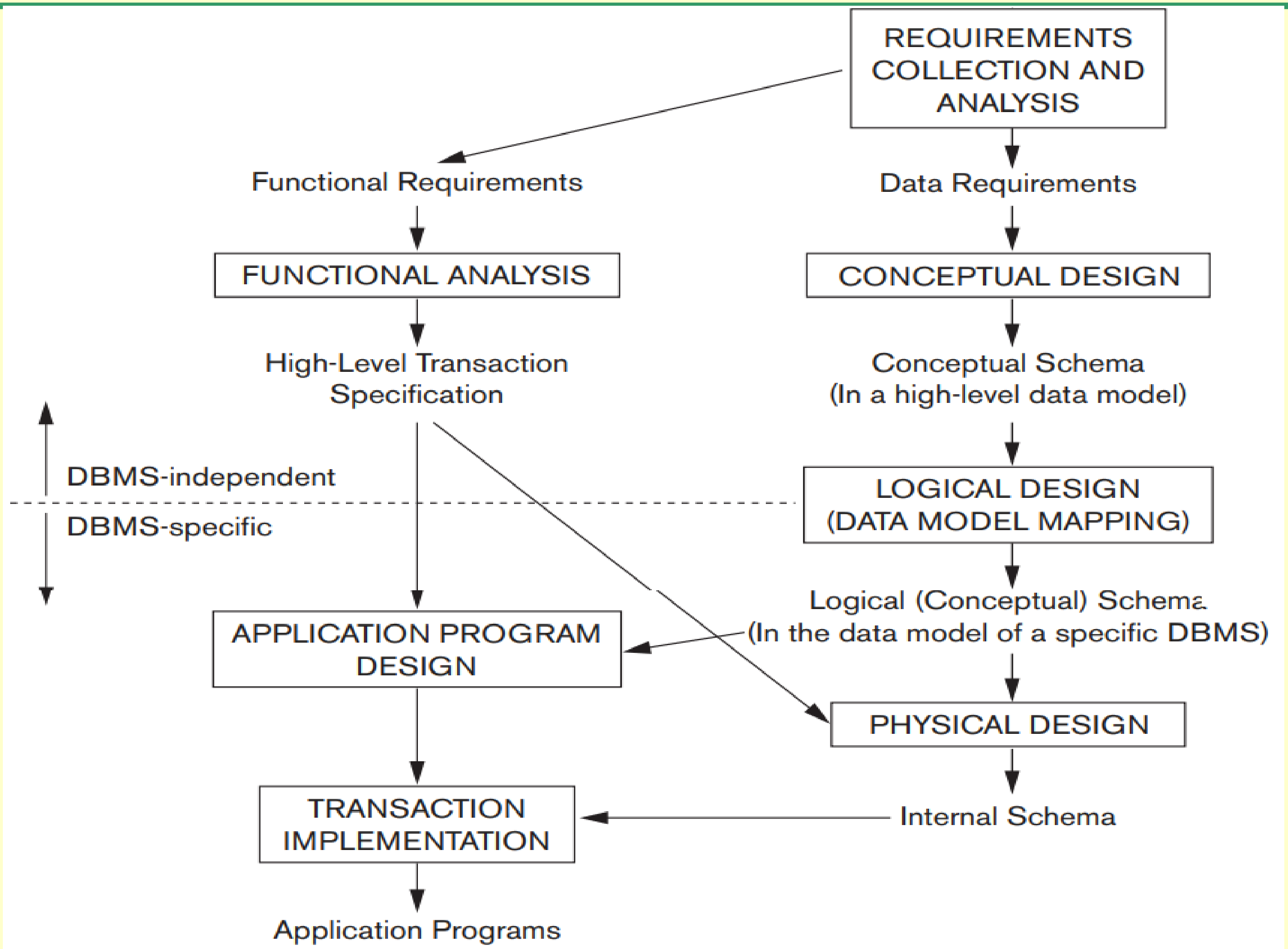

REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the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

Fig: Database Design Process

Using High-Level Conceptual Data Model for Database Design

- The next step is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use relational (SQL) model. The conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping**.
- The last step is the **physical design phase**, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications.

Entity Types, Entity Sets, Attributes, and Keys

- The ER model describes data as entities, relationships, and attributes.
- **Entities and Attributes:**
  - The basic concept that the ER model represents is an **entity**. An entity is a thing or object in the real world with an independent existence. For example, a particular person, a car, a student, a company, a job, a university course etc. are entities.
  - Each entity has **attributes**. Attributes are the particular properties that describe the entity. For example, an EMPLOYEE entity may be described by the employee’s name, age, address, salary, and job. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

Entity Types, Entity Sets, Attributes, and Keys

- Several types of attributes occur in the ER model: **simple** versus **composite**, **single-valued** versus **multivalued**, and **stored** versus **derived**.
- **Composite** versus **Simple (Atomic) Attributes. Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the *Address* attribute of the *EMPLOYEE* entity can be subdivided into *Street_address*, *City*, *State*, and *Zip*. Attributes that are not divisible are called **simple** or **atomic attributes**. The value of a composite attribute is the concatenation of the values of its component simple attributes. If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes. For example, if there is no need to refer to the individual components of an address, then the whole address can be designated as a simple attribute.

Entity Types, Entity Sets, Attributes, and Keys

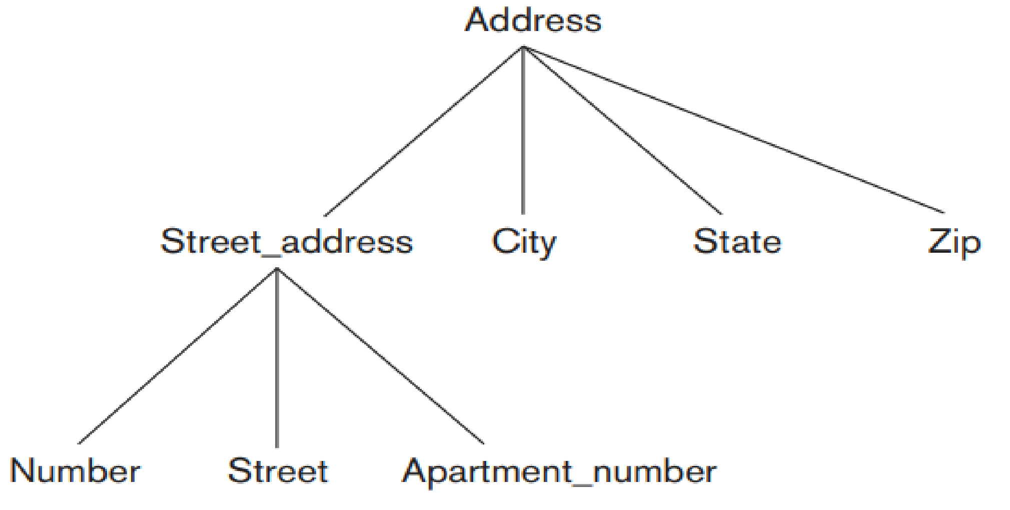


Fig: Hierarchy of composite attributes

- **Single-valued** versus **Multivalued Attributes. Single-valued attributes** have a single value for a particular entity. For example, ***Age*** is a single-valued attribute of a person. **Multivalued attributes** may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, ***College_degrees*** attribute of a person is multivalued attribute because one person may not have any college degrees, another person may have one, and a third person may have two or more.

Entity Types, Entity Sets, Attributes, and Keys

- **Stored** versus **Derived Attributes.** The value of **derived attributes** can be computed from other attribute values. For example, the ***Age*** attribute is derived attribute and is said to be derivable from the ***Birth_date*** attribute, which is **stored attribute**.
- **Complex Attributes.** In general, composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called complex attributes. For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address_phone for a person can be specified as follows.

{Address_phone({Phone(Area_code,Phone_number)},Address(Street_address (Number,Street,Apartment_number),City,State,Zip))}

Entity Types, Entity Sets, Attributes, and Keys

- **NULL Values.** In some cases, a particular entity may not have an applicable value for an attribute. For example, the *Middle_name* attribute of a *Person* may not have an applicable value for some entities. For such situations, a special value called NULL is created. NULL can also be used if we do not know the value of an attribute for a particular entity. The meaning of the former type of NULL is not applicable, whereas the meaning of the latter is unknown. The unknown category of NULL can be further classified into two cases. The first case arises when it is known that the attribute value exists but is missing. The second case arises when it is not known whether the attribute value exists.

Entity Types, Entity Sets, Attributes, and Keys

- **Entity Types, Entity Sets, Keys, and Value Sets:**
  - **Entity Types and Entity Sets.** Entities with same attributes are grouped (or typed) into an **entity type**. Each entity type in the database is described by its name and attributes. The collection of all entities of a particular entity type in the database at any point in time is called an **entity set** or **entity collection**. The entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current collection of all employee entities in the database. An entity type describes the schema or **intension** for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

Entity Types, Entity Sets, Attributes, and Keys

- **Key Attributes of an Entity Type.** An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely. For example, Citizenship_number attribute of the PERSON entity type is a key. Sometimes several attributes together form a key. Such a composite key must be minimal, that is, all component attributes must be included in the composite attribute to have the uniqueness property. Superfluous attributes must not be included in a key. Some entity types have more than one key attribute. An entity type may also have no key, in which case it is called a weak entity type.
- **Value Sets (Domains) of Attributes.** Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity. Value sets are similar to the basic data types available in most programming languages.

Relationship Types, Relationship Sets, and Structural Constraints

- **Relationship Types, Sets, and Instances:**
  - A **relationship** relates two or more distinct entities. Relationships of the same type are grouped (or typed) into a **relationship type**. Relationship type is the schema description and identifies the relationship name and the participating entity types. The current set of **relationship instances** represented in the database is called **relationship set.** It is the current state of the relationship type. A relationship type and its corresponding relationship set are referred to by the same name, R.
  - Mathematically, the relationship set R is a set of relationship instances r_i, where each r_i associates n individual entities (e_1, e_2, …, e_n), and each entity e_j in r_i is a member of entity set E_j , $1 \leq j \leq n$. Hence, a relationship set is a mathematical relation on E_1, E_2, ..., E_n. It can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times \ldots \times E_n$. Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R.

Relationship Types, Relationship Sets, and Structural Constraints

- **Relationship Degree, Role Names, and Recursive Relationships:**
  - **Degree of a Relationship Type.** The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called **binary**, and one of degree three is called **ternary**. Relationships can generally be of any degree, but the ones most common are binary relationships. Higher-degree relationships are generally more complex than binary relationships.
  - **Relationships as Attributes.** It is sometimes convenient to think of a binary relationship type in terms of attributes. For example, foreign keys are a type of reference attribute used to represent relationships in relational databases,

Relationship Types, Relationship Sets, and Structural Constraints

- **Role Names and Recursive Relationships.** Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and it helps to explain what the relationship means. Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships** or **self-referencing relationships**. For example, SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set.

Relationship Types, Relationship Sets, and Structural Constraints

- **Constraints on Binary Relationship Types:**
  - Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. We can distinguish two main types of binary relationship constraints: **cardinality ratio** and **participation**.
  - **Cardinality Ratios for Binary Relationships.** The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. The possible cardinality ratios for binary relationship types are 1:1 (one-to-one), 1:N (one-to-many), N:1 (many-to-one), and M:N (many-to-many).

    In **1:1 cardinality ratio**, an entity in entity set E_1 is associated with at most one entity in entity set E_2, and an entity in entity set E_2 is associated with at most one entity in entity set E_1. For example, cardinality ratio between DEPARTMENT and CHAIRPERSON.

Relationship Types, Relationship Sets, and Structural Constraints

In **1:N cardinality ratio**, an entity in entity set E_1 is associated with any number (zero or more) of entities in entity set E_2, and an entity in entity set E_2 is associated with at most one entity in entity set E_1. For example, cardinality ratio between MOTHER and CHILDREN.

In **N:1 cardinality ratio**, an entity in entity set E_1 is associated with at most one entity in entity set E_2, and an entity in entity set E_2 is associated with any number (zero or more) of entities in entity set E_1. For example, cardinality ratio between CHILDREN and MOTHER.

In **M:N cardinality ratio**, an entity in entity set E_1 is associated with any number (zero or more) one entities in entity set E_2, and an entity in entity set E_2 is associated with any number (zero or more) of entities in entity set E_1. For example, cardinality ratio between STUDENR and COURSE.

Relationship Types, Relationship Sets, and Structural Constraints

- **Participation Constraints and Existence Dependencies.** The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the **minimum cardinality constraint**. There are two types of participation constraints, **total** and **partial**.

The participation of an entity set E_1 in a relationship set R is said to be **total** if every entity in E_1 participates in at least one relationship in R. For example, consider CUSTOMER and ACCOUNT entity sets in a banking system, and a relationship set OWNS between them indicating that each customer must have an account. Then there is total participation of entity set CUSTOMER in the relationship set DEPOSITOR. Total participation is also called **existence dependency**.

Relationship Types, Relationship Sets, and Structural Constraints

If only some entities in E_1 participate in relationships in R, the participation of entity set E_1 in relationship set R is said to be **partial**. For example, consider CUSTOMER and LOAN entity sets in a banking system, and a relationship set BORROWER between them indicating that some customers have loans. Then there is partial participation of entity set CUSTOMER in the relationship set BORROWER.

We will refer to the cardinality ratio and participation constraints, taken together, as the **structural constraints** of a relationship type.

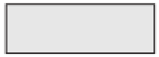
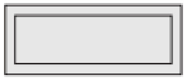
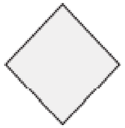
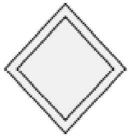
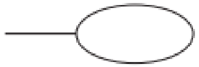
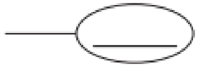
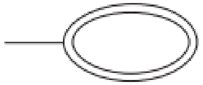
- **Attributes of Relationship Types:**
  - Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that a particular employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type.

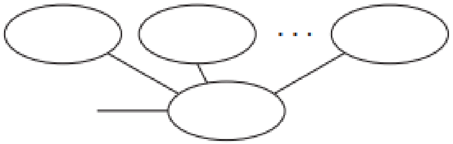
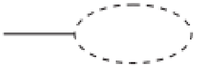
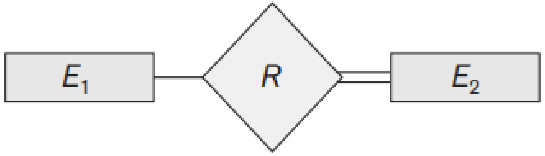
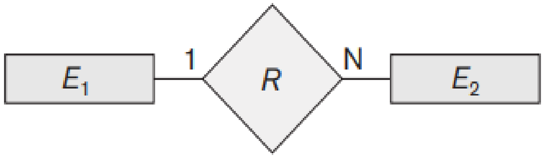
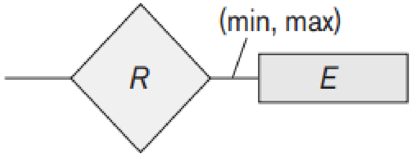
Weak Entity Types

- **Weak entity types** do not have key attributes of their own. **Regular entity types** have a key attribute. Entities in a weak entity type are identified by entities from another entity type. We call this other entity type the **identifying** or **owner entity type**, and we call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type. For example PAYMENT entity type related to LOAN entity type is a weak entity type.
- A weak entity type always has a total participation with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
- A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.

ER Diagrams, Naming Conventions, and Design Issues

- **Summary of Notation for ER Diagrams:**

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute

Symbol	Meaning
	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, Naming Conventions, and Design Issues

- **Example:** ER schema design for COMPANY database.
  - The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
  - A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
  - The database will store each employee’s name, Social Security number, address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
  - The database will keep track of the dependents of each employee for insurance purposes, including each dependent’s first name, sex, birth date, and relationship to the employee.

ER Diagrams, Naming Conventions, and Design Issues

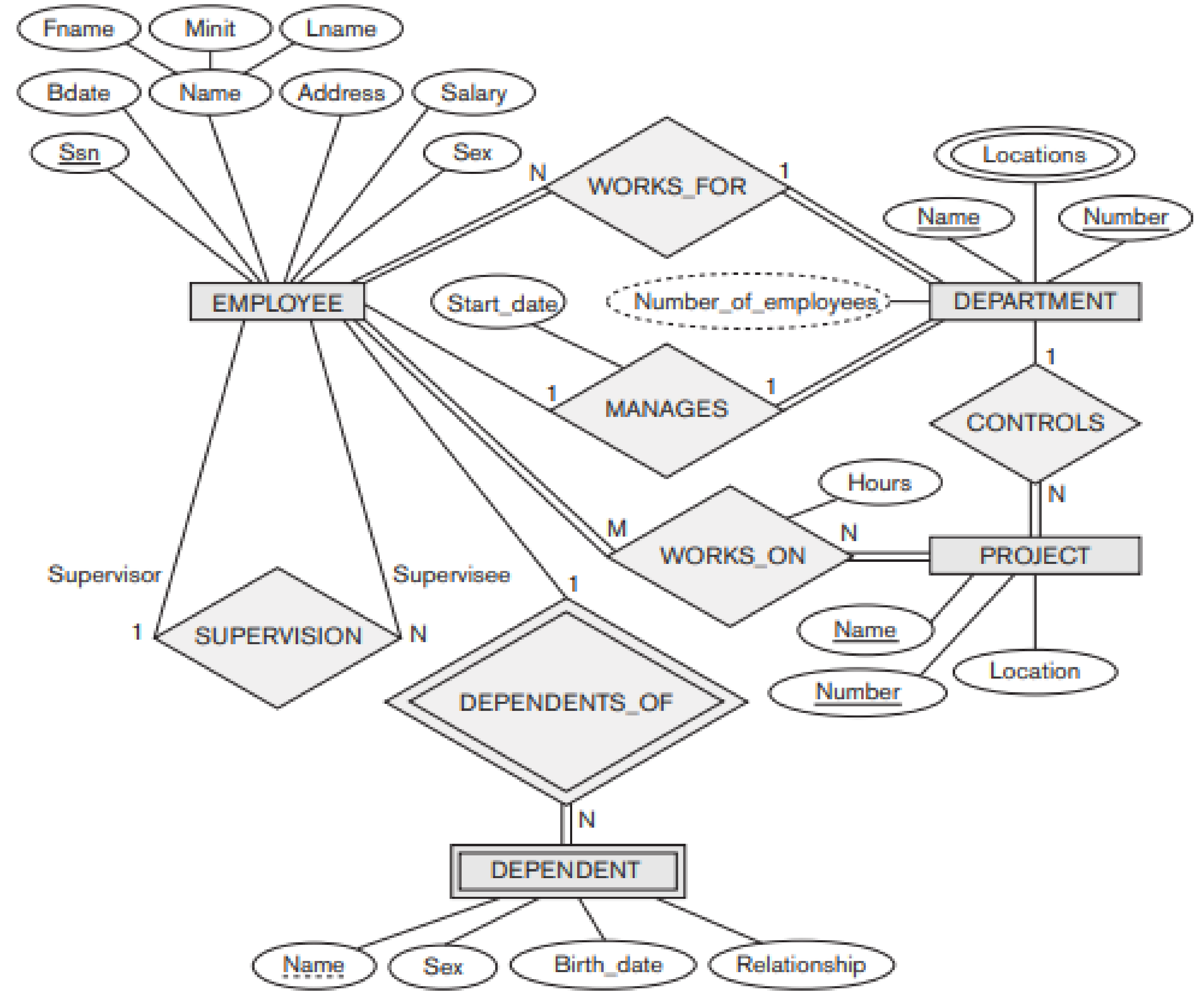


Fig: ER schema diagram for COMPANY database

ER Diagrams, Naming Conventions, and Design Issues

- **Proper Naming of Schema Constructs:**
  - The choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward.
  - One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema.
  - We use **singular nouns** (such as EMPLOYEE) to represent entity types and **verbs** (such as MNAGES) to represent relationship types. Attributes are represented using **nouns** (such as Name, Locations etc.).
  - One common convention is that entity type and relationship type names are in uppercase letters, attribute names have their initial letter capitalized, and role names are in lowercase letters.
  - Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.

ER Diagrams, Naming Conventions, and Design Issues

- **Design Choices for ER Conceptual Design:**
  - It is occasionally difficult to decide whether a particular concept should be modeled as an entity type, an attribute, or a relationship type.
  - In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached.
  - A concept may be first modeled as an attribute and then refined into a relationship. Once an attribute is replaced by a relationship, the attribute itself should be removed from the entity type to avoid duplication and redundancy.
  - An attribute that exists in several entity types may be elevated or promoted to an independent entity type.
  - An entity type may be reduced to an attribute.

ER Diagrams, Naming Conventions, and Design Issues

- **Alternative Notations for ER Diagrams:**
  - There are many alternative diagrammatic notations for displaying ER diagrams.
  - One approach replaces the cardinality ratio (1:1, 1:N, N:1, M:N) and single/double-line notation for participation constraints with 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 $\text{max} \geq 1$.
  - For each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time. In this method, min = 0 implies partial participation, whereas min > 0 implies total participation.
  - Usually, one uses either the cardinality ratio/single-line/double-line notation or the (min, max) notation. The (min, max) notation is more precise.

ER Diagrams, Naming Conventions, and Design Issues

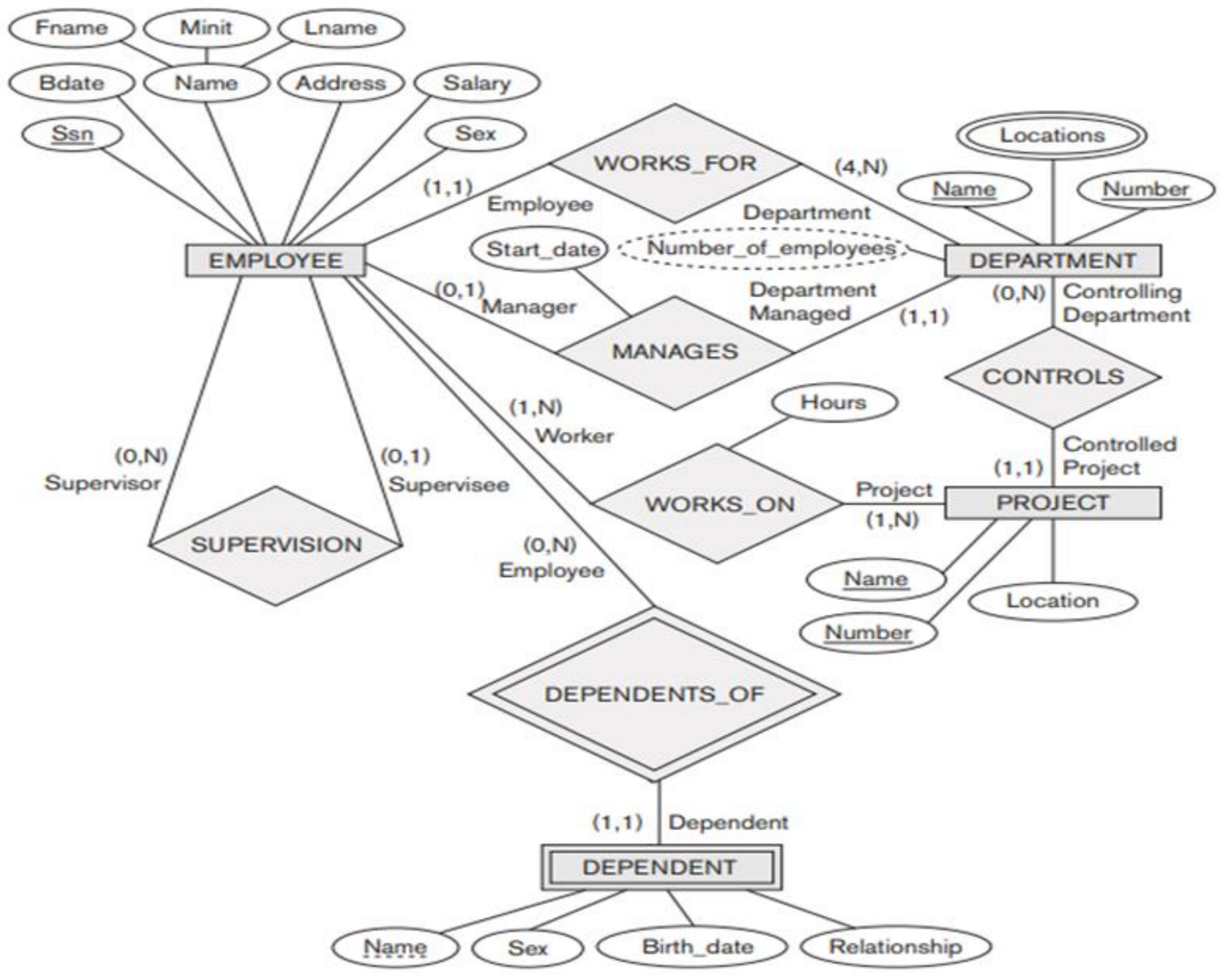


Fig: ER schema diagram with (min,max) notation and role names

Relationship Types of Degree Higher than Two

- Degree of a relationship type is the number of participating entity types.
- A relationship type of degree two is called **binary**, degree three is called **ternary** and degree n is called **n-ary**.
- Relationship types of **higher-degree** (three or more) also frequently exist during database design.
- **Choosing between Binary and Ternary (or Higher-Degree) Relationships:**
  - In general, an n-ary relationship is not equivalent to n binary relationships. Hence, a ternary relationship type can represent different information than do three binary relationship types.
  - If needed, ternary relationship plus one or more of the binary relationships will be included if they represent different meanings and if all are needed by the application.

Relationship Types of Degree Higher than Two

- Some database design tools are based on variations of the ER model that permit only binary relationships. In this case, a ternary relationship must be represented as a weak entity type, with no partial key and with three identifying relationships. The three participating entity types are the owner entity types. Hence, an entity in the weak entity type is identified by the combination of its three owner entities from three entity types.
- It is also possible to represent the ternary relationship as a regular entity type by introducing an artificial or surrogate key.
- Notice that it is possible to have a weak entity type with a ternary (or n-ary) identifying relationship type. In this case, the weak entity type can have several owner entity types.
- Constraints are harder to specify for higher-degree relationships (n > 2) than for binary relationships.

Relationship Types of Degree Higher than Two

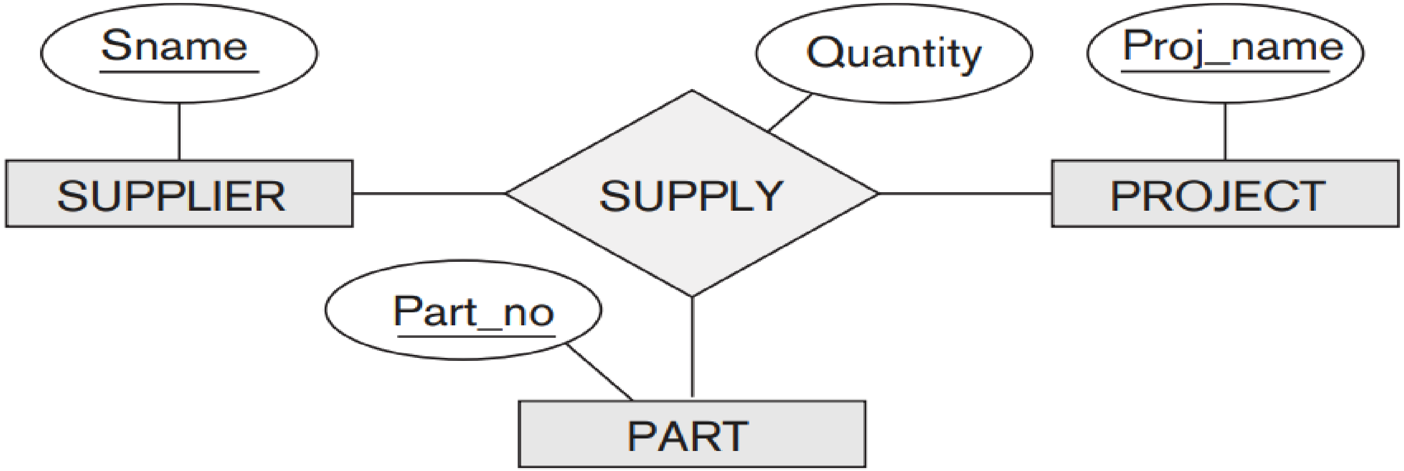


Fig: Ternary relationship

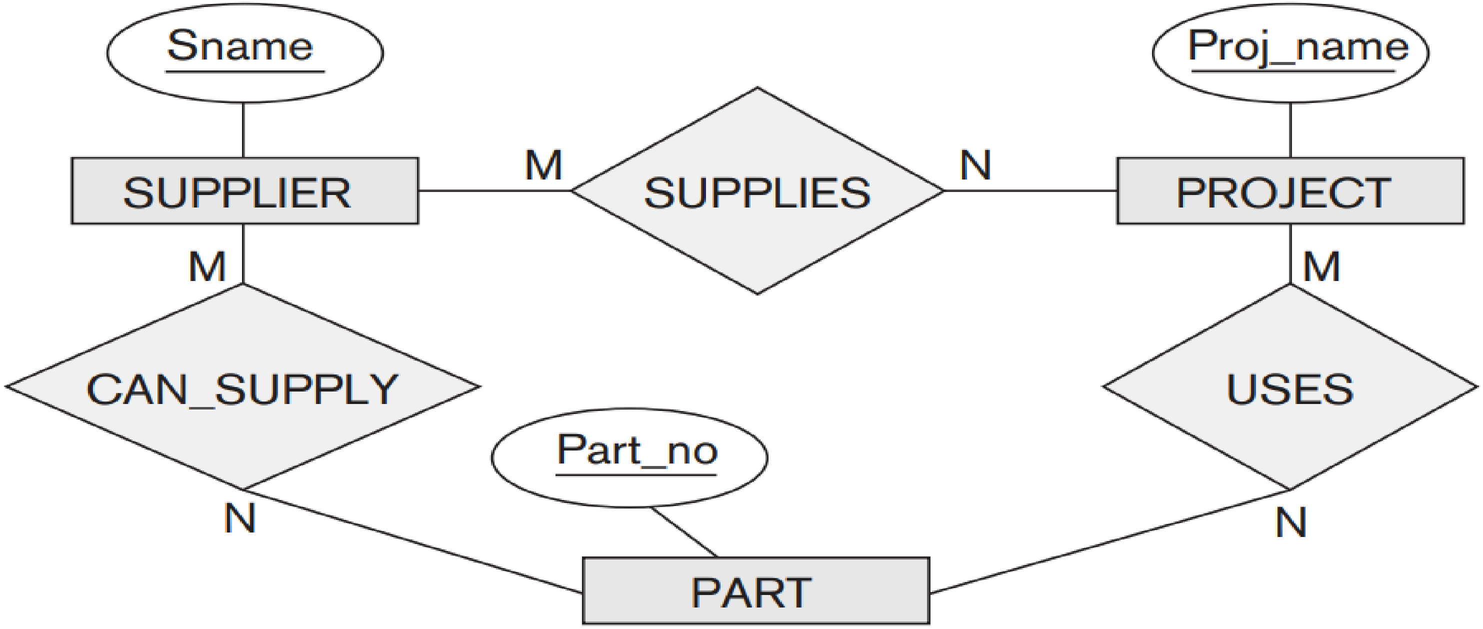


Fig: Three binary relationships not equivalent to ternary relationship

Relationship Types of Degree Higher than Two

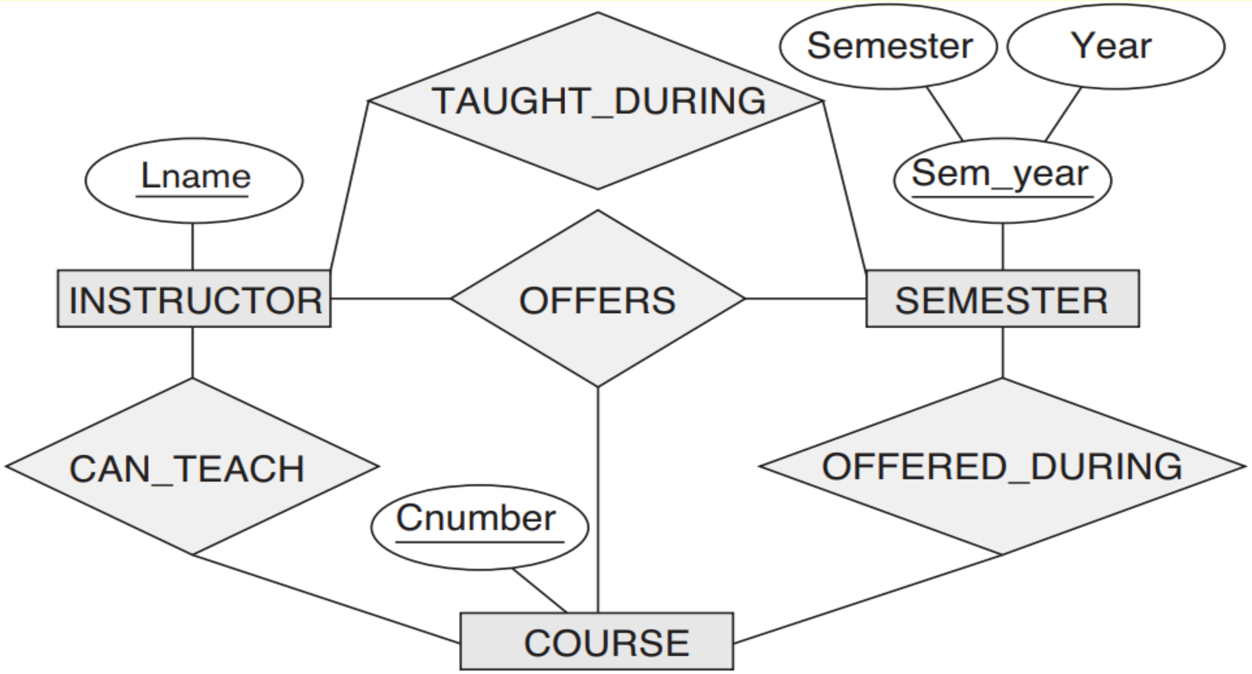


Fig: Ternary plus binary relationship

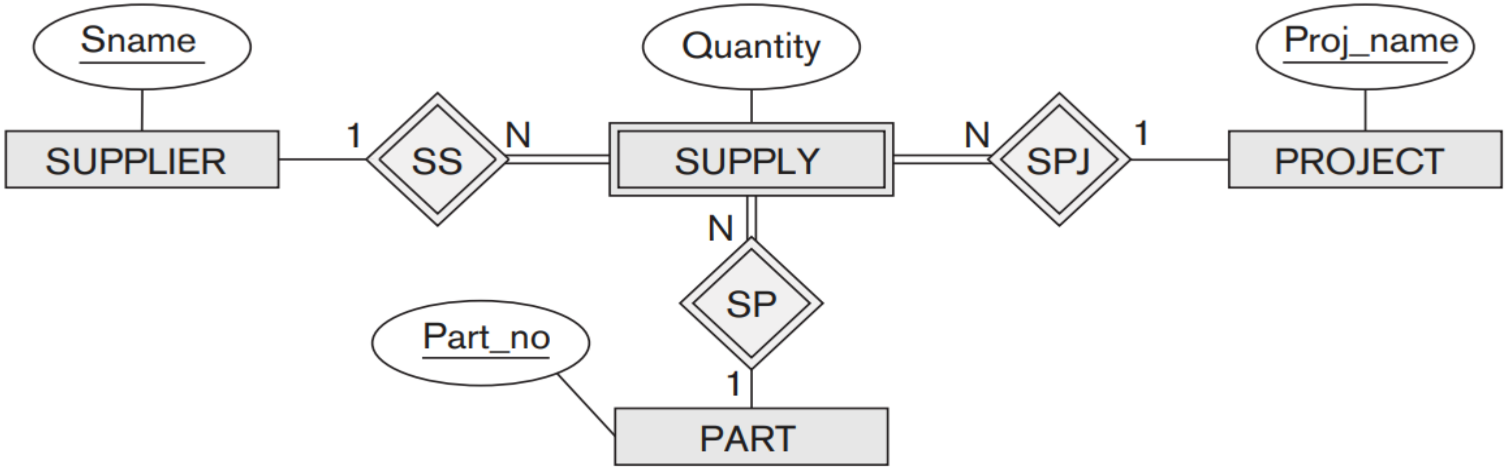


Fig: Ternary relationship represented as a weak entity type

Relationship Types of Degree Higher than Two

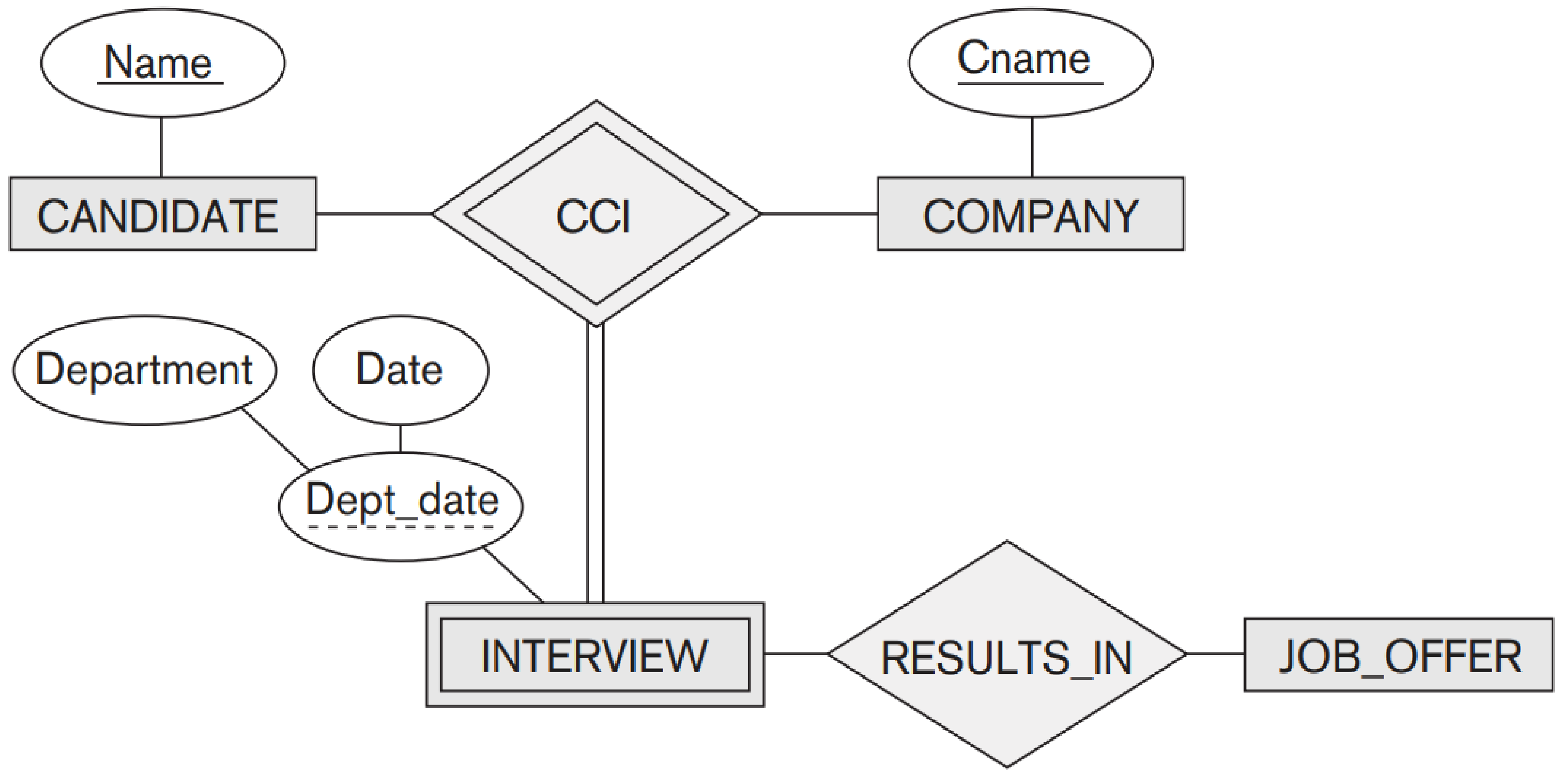


Fig: A weak entity type with a ternary identifying relationship type

Relationship Types of Degree Higher than Two

- **Constraints on Ternary (or Higher-Degree) Relationships:**
  - Both 1, M, or N and (min, max) constraints should be used to fully specify the constraints on a ternary or higher-degree relationship.
  - An M or N indicates that an entity can participate in any number of relationship instances that has a particular combination of the other participating entities. A 1 indicates that an entity can participate in at most one relationship instance that has a particular combination of the other participating entities
  - A (min, max) specifies that each entity is related to at least min and at most max relationship instances in the relationship set. These constraints specify restrictions on how many relationship instances each entity can participate in.
  - Constraint specification is difficult and possibly ambiguous when we consider relationships of a degree higher than two.

Subclass, Superclass, and Inheritance

- The enhanced ER (EER) model includes all the modeling concepts of the ER model and concepts of **subclass** and **superclass** and the related concepts of **specialization** and **generalization**.
- Associated with these concepts is the important mechanism of **attribute and relationship inheritance**.
- We also describe a diagrammatic technique for displaying these concepts when they arise in an EER schema. We call the resulting schema diagrams **enhanced ER** or **EER diagrams**.
- The first EER model concept we take up is that of a **subtype** or **subclass** of an entity type.

Subclass, Superclass, and Inheritance

- An entity type may have numerous subgroupings or subtypes of its entities that are meaningful and need to be represented explicitly because of their significance to the database application.
- We call the relationship between a superclass and any one of its subclasses a **superclass/subclass** or **supertype/subtype** or simply **class/subclass** relationship.
- A member entity of the subclass represents the same real-world entity as some member of the superclass. Hence, the subclass member is the same as the entity in the superclass, but in a distinct specific role.
- A class/subclass relationship is often called an **IS-A** (or **IS-AN**) relationship.

Subclass, Superclass, and Inheritance

- An important concept associated with subclasses (subtypes) is that of **type inheritance**. An entity that is a member of a subclass inherits all the attributes of the entity as a member of the superclass. The entity also inherits all the relationships in which the superclass participates. Notice that a subclass, with its own specific (or local) attributes and relationships together with all the attributes and relationships it inherits from the superclass, can be considered an entity type in its own right.

Subclass, Superclass, and Inheritance

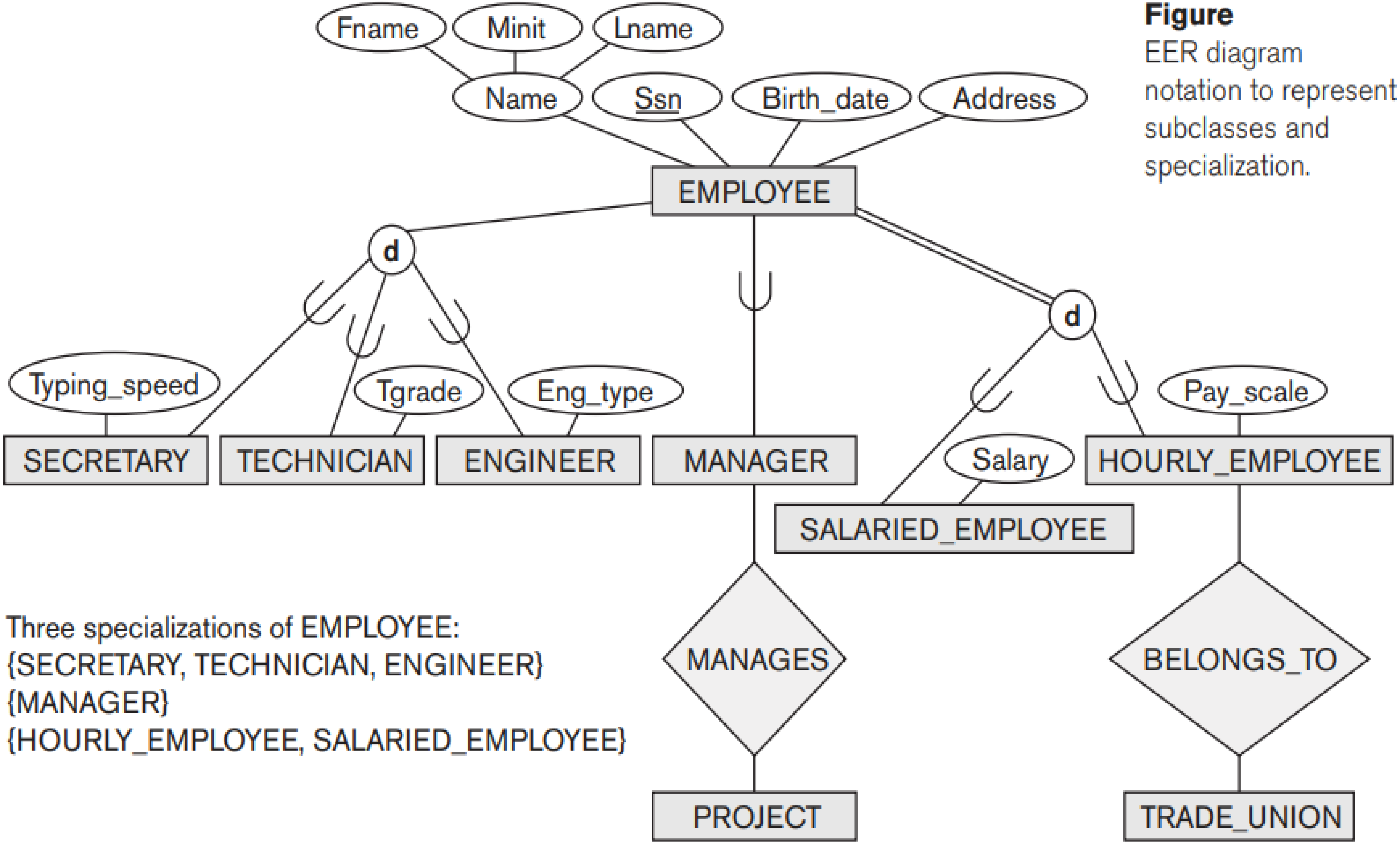


Figure
EER diagram notation to represent subclasses and specialization.

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

Specialization and Generalization

- **Specialization:**
  - Specialization is the process of defining a set of subclasses of an entity type (supertype). The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.
  - We may have several specializations of the same entity type based on different distinguishing characteristics.
  - The subclasses are connected to a circle that represents the specialization (using lines *with the subset symbol*). The circle is also connected to the superclass.
  - Attributes of a subclass are called *specific* or *local* attributes. The subclass can also participate in specific relationship types.
  - Database designers typically start with an entity type and then define subclasses of the entity type repeatedly to define more specific groupings. This process is called **top-down conceptual refinement**.

Specialization and Generalization

- **Generalization:**
  - Here, we identify their common features of several entity types and generalize them into a single superclass of which the original entity types are special subclasses.
  - We use the term generalization to refer to the process of defining a generalized entity type from the given entity types. Notice that the generalization process can be viewed as being functionally the inverse of the specialization process.
  - A same diagrammatic notation can be used for both generalization and specialization.
  - Generalization is called **bottom-up conceptual synthesis** process.

Constraints and Characteristics of Specialization and Generalization

- In general, we may have several specializations defined on the same entity type (or superclass). In such a case, entities may belong to subclasses in each of the specializations.
- A specialization may also consist of a single subclass only. In such a case, we do not use the circle notation.
- **Predicate defined specialization:**
  - In some specializations we can determine exactly the entities that will become members of each subclass by placing a condition on the value of some attribute of the superclass. Such subclasses are called **predicate-defined** (or **condition-defined**) subclasses and the predicate is called the **defining predicate** of the subclass.
  - We display a predicate-defined subclass by writing the predicate condition next to the line that connects the subclass to the specialization circle.

Constraints and Characteristics of Specialization and Generalization

- **Attribute-defined specialization:**
  - If all subclasses in a specialization have their membership condition on the same attribute of the superclass, the specialization itself is called an **attribute-defined specialization**, and the attribute is called the **defining attribute** of the specialization.
  - All the entities with the same value for the attribute belong to the same subclass. We display an attribute-defined specialization by placing the defining attribute name next to the arc from the circle to the superclass.
- **User-defined specialization:**
  - When we do not have a condition for determining membership in a subclass, the subclass is called **user-defined**.
  - Membership in such a subclass is determined by the database users when they apply the operation to add an entity to the subclass; hence, membership is specified individually for each entity by the user, not by any condition that may be evaluated automatically.

Constraints and Characteristics of Specialization and Generalization

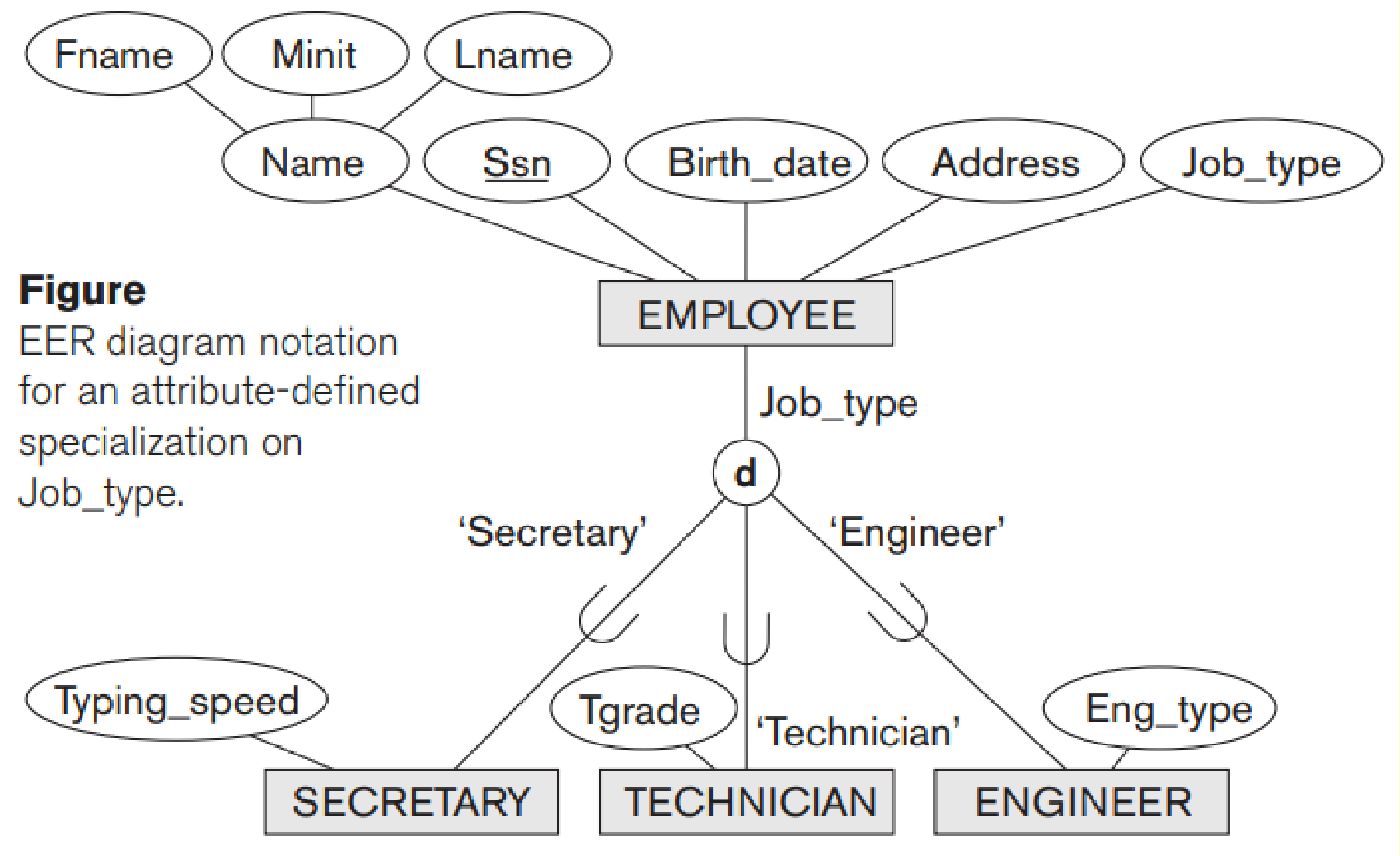


Figure
EER diagram notation for an attribute-defined specialization on Job_type.

Constraints and Characteristics of Specialization and Generalization

- **Disjointness Constraint:**
  - Disjoint constraint specifies that the subclasses of the specialization must be disjoint sets. An entity can be a member of at most one of the subclasses of the specialization. The **d** in the circle stands for disjoint.
  - If the subclasses are not constrained to be disjoint, their sets of entities may be **overlapping**; that is, the same entity may be a member of more than one subclass of the specialization. We place an **o** in the circle for overlapping constraint.
- **Completeness (or totalness) constraint:**
  - This constraint may be **total** or **partial**.
  - A **total specialization** constraint specifies that every entity in the superclass must be a member of at least one subclass in the specialization. This is shown in EER diagrams by using a *double line* to connect the superclass to the circle.
  - A *single line* is used to display a **partial specialization**, which allows an entity not to belong to any of the subclasses.

Constraints and Characteristics of Specialization and Generalization

- Disjointness and completeness constraints are independent. Hence, we have the four possible constraints on a specialization:
  - Disjoint, total
  - Disjoint, partial
  - Overlapping, total
  - Overlapping, partial