Chapter 3: Data modeling using the Entity-Relationship Model

Outline:

3.1 Design Phases.

- Phase 1: Requirements collection and analysis.
- Phase 2: Conceptual Design.
- Phase 3: Logical design (data model mapping).
- Phase 4: Physical design.

3.2 The Entity-Relationship (ER) Model.

- Entity Types, Entity Sets, Attributes, and Keys.
- Null Values.
- Relationship.
- The degree of a relationship type.

3.3 Structural Constraints on relationship.

- Cardinality ratio.
- Participation Constraints.
- Alternative (min, max) notation for relationship structural constraints.

3.4 Weak Entity Types.

3.5 An Example of Database Application (COMPANY Database).

3.1 Design Phases

Phase 1: Requirements collection and analysis

- Interview prospective database users to understand and document their data requirements
- The result of this phase is concisely written set of users' requirements and functional requirements of the application; consist of the user defined operations (or transactions) that will be applied to the database, and they include both retrievals and updates).

Phase 2: Conceptual Design

- Conceptual *schema* for the database, using High-Level Conceptual Data Model (entity types, relationships, and constraints).Because these concepts do not include implementation details, they are usually easier to understand and can used to communicate with no technical users. The high level conceptual schema can also be used as a reference to ensure that all users' data requirements are met and that the requirements do not include conflicts.
- The result is Conceptual Schema in high level model

Phase 3: Logical design (data model mapping)

- The actual implementation of a database, using a commercial DBMS; transformation of conceptual schema from the high level data model into implementation data model.
- The result is Database schema in the implementation data model of the DBMS.

Phase 4: Physical design

- The internal storage structures, access paths, and file organizations for the database files are specified.

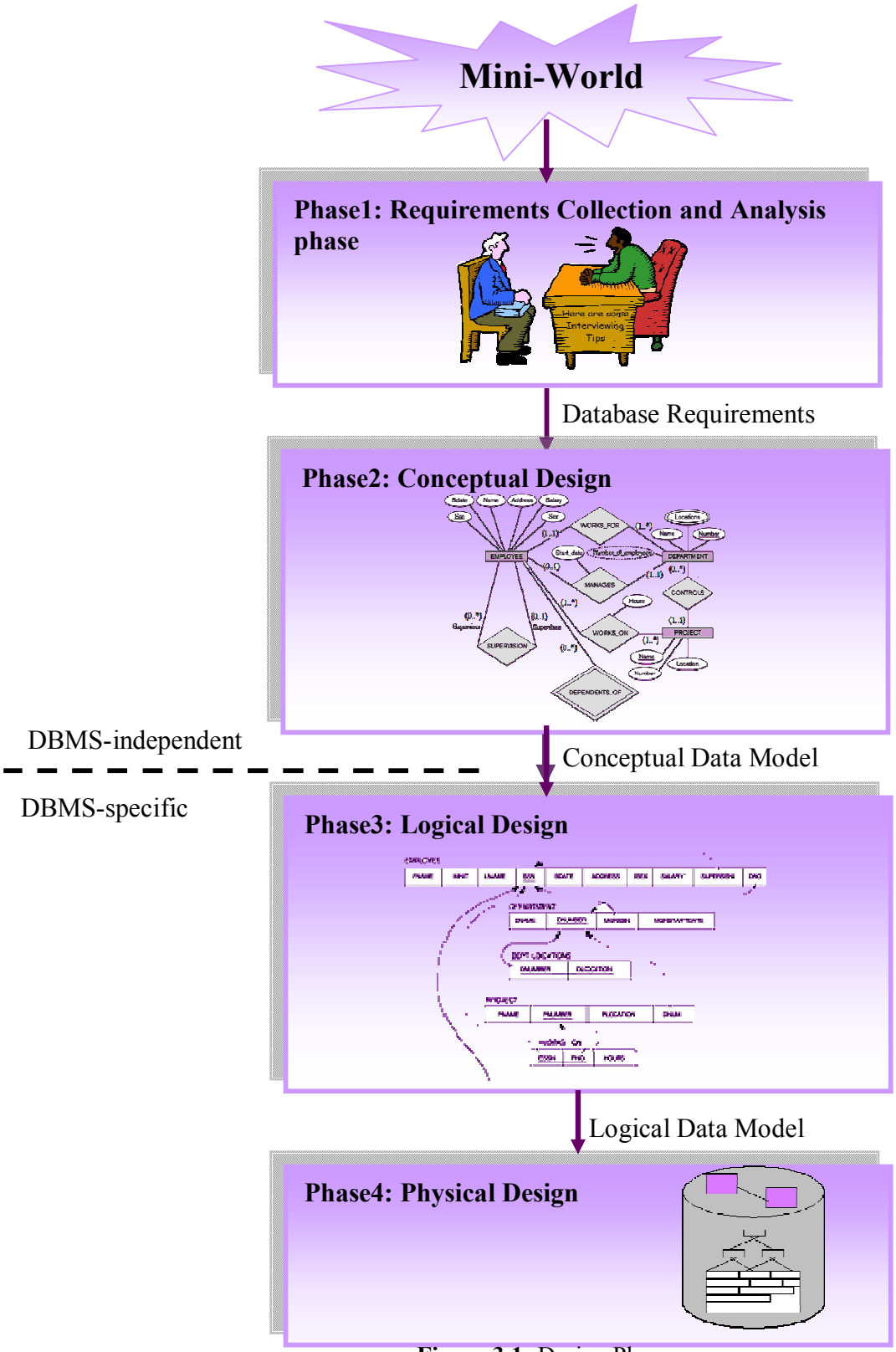


Figure 3.1: Design Phases

3.2 The Entity-Relationship (ER) Model

- Used for conceptual database design
- Relies on concepts of entities, attributes, and relationships

Entity Types, Entity Sets, Attributes, and Keys

- **Entity**: An entity is a "thing" (a being, an object, an event) in the real world and is distinguishable from other objects (things).

Figure 3.2: Entities Examples

- Entity types represent sets of objects and are pictured by rectangular nodes

- ***Characteristic of Entity:***

  - **Existence:** It may have a physical existence (also called concrete) (for example, car, house, or an employee) in the real world or it may be an object with a conceptual existence (abstract). (for example, a job, or a university course).

  - **Described by its attributes** (set of properties); for example an EMPLOYEE entity may have a Name, SSN, Address, Sex, BirthDate.

  - **Determined by particular value** of its attributes; a specific entity will have a value for each of its attributes(domain);
    - ***Example:*** a specific **book entity** will have *title*= "Database System Concepts", *pages*=821, *ISBN*= 0-07-044756-X
    - ***Example:*** A **person** *name*= "Mohammed", *age*=21, *add=salt*

  - **Domain**: the set of permitted values for each attribute

  - **Entity sets do not need to be disjoint**. For example, it is possible to define the entity set of all employees of a bank (employee) and the entity set of all customers of the bank

(customer). A person entity may be an employee entity, a customer entity, both, or neither.

- An **entity type** defines a collection (or set) of entities that have the same attribute (for example, employee entity type and company entity type). An entity is an instance of an entity type. All entities having the same set of properties are grouped into an entity type.

- An **entity set** is the collection of all entities of a particular entity type in the database at any point in time

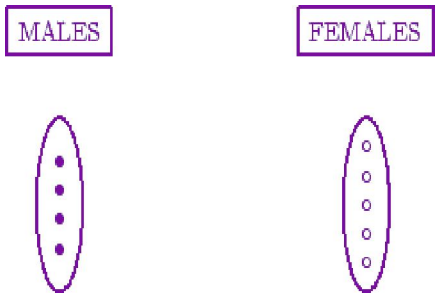


Figure 3.3: Entity sets Example

- An entity set is also called an **extension** of the entity type.

Types of Attributes

- Entity types and relationship types might have attributes.

- **Several types of attributes** occur in the ER model (simple Vs. composite, single-valued Vs. multivalued, Stored Vs. Derived Attribute, complex Attribute, and key Attribute)

1. ***Atomic attribute types (Simple)***

   - Each entity has a single atomic value for the attribute; for example SSN or Gender.

   - **Pictured by** oval nodes

2. ***Composite attribute types***

   - Achieved by concatenating simpler attribute types. The attribute may be composed of several components; for example Address (Street, City, State, and Postal Code) or Name (First Name, Middle Name, and Last Name).

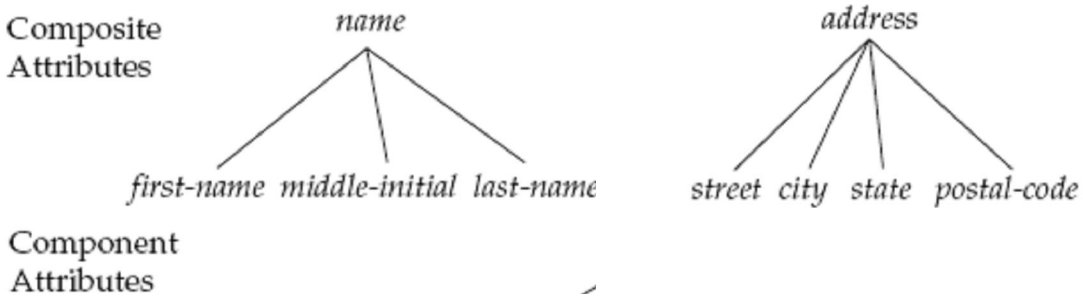
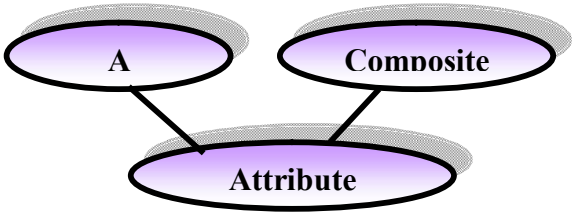


Figure 3.4: Composite Attribute Examples

- Composite Attribute **Pictured by** trees of atomic attributes



- Composition may form a hierarchy where some components are themselves composite; for example StreetAddress can be subdivided into three simple attributes, Number, Street ,and ApartmentNumber, as shown in the following figure :

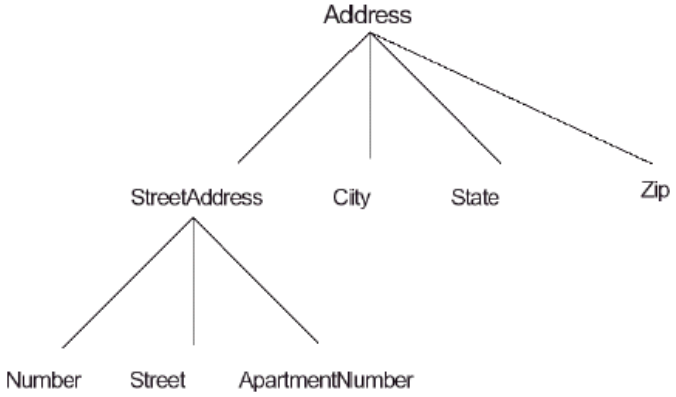


Figure 3.5: Hierarchy Composite Attribute Example

3. ***Multivalued attribute types***
   - 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}.

   - Displayed in two nested ovals:

4. ***Derived attribute types***

   - Some attribute values can be derived from related attributes or entities; for example an attribute ***Age*** of employee entity can be determined from the current date and the value of stored attribute (BirthDate).

   - It displayed in dashed ovals

5. ***Complex Attribute***

   - 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)}.
   - ***example***: {AddressPhone ({ Phone (AreaCode, PhoneNumber) }, Address (StreetAddress (Number, Name), City, State, Zip)) }

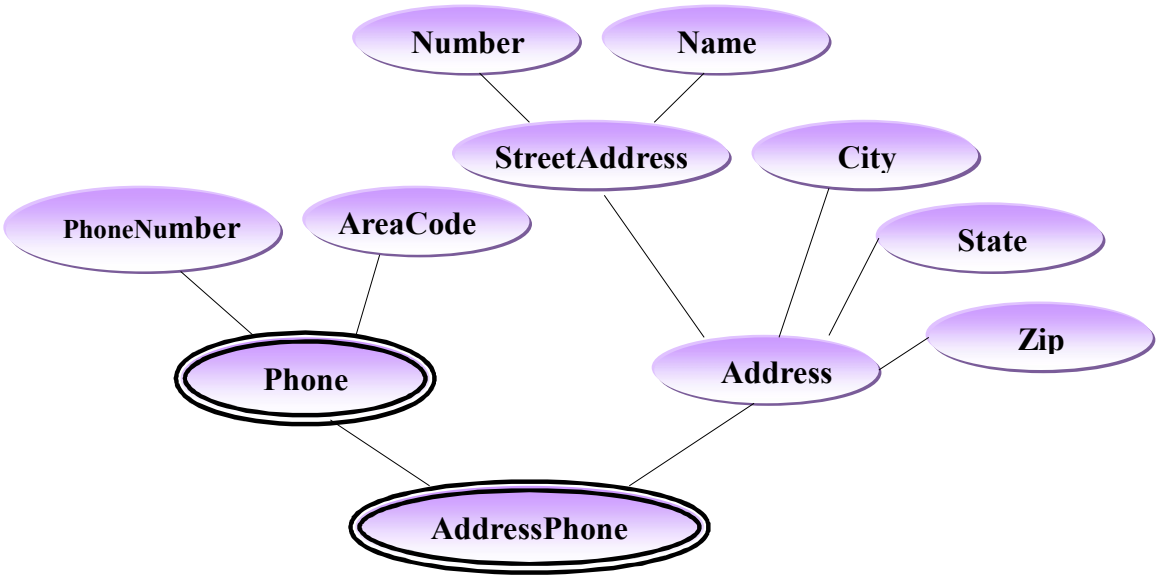


Figure 3.6: AddressPhone Complex Attribute

6. ***Key Attribute (identifier)***

   - An attribute whose values are distinct for each individual entity in the collection. In other words, 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.

   - **Displayed with underlined** attribute type names

- Key attribute may be composite. For example, VehicleRegistrationNumber is a key of the CAR entity type with components (Number, State).

- An entity type may have more than one key. For example, the CAR entity type may have two keys:
  A. VehicleIdentificationNumber (popularly called VIN) and
  B. VehicleTagNumber (Number, State), also known as license_plate number.
- The following ER diagram for car:

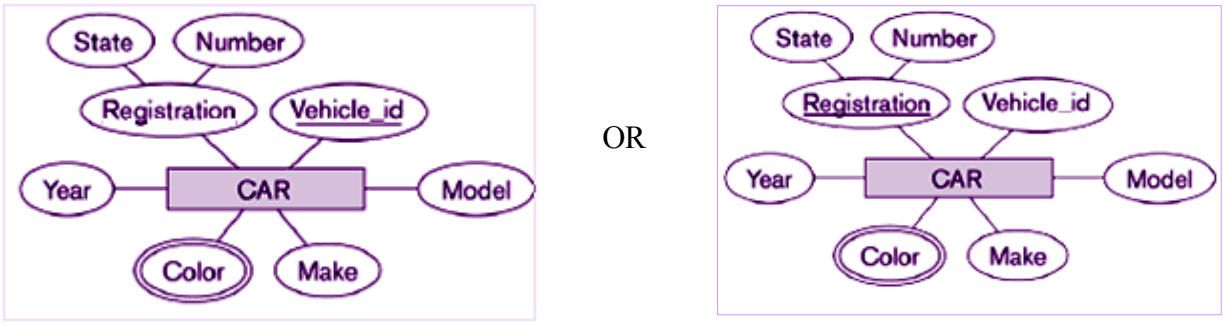


Figure 3.7: Car Entity Type.

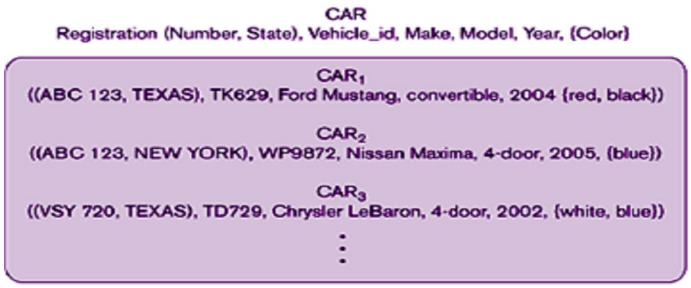


Figure 3.8: Car Entity Set with three entities.

- The following figure contains all of the above types of attributes:

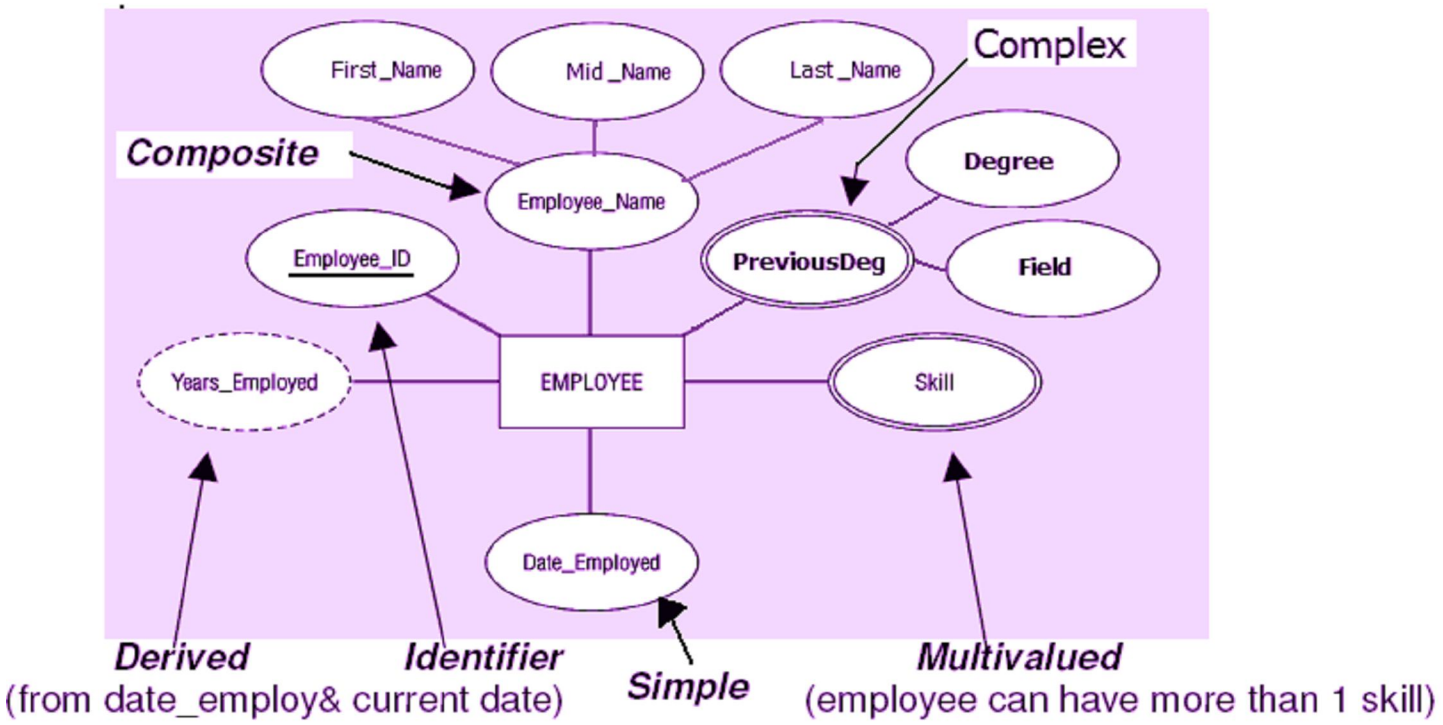


Figure 3.9: Entity Type with all attributes types.

Null Values

- There are two situations for null values:
    - In some cases a particular entity may **not have an applicable value** for an attribute. Example, the ApartmentNumber attribute of an address applies only to addresses that are in apartment buildings and not to single-family homes.
    - If we do **not know the value** of an attribute for a particular entity. The unknown category of null can be further classified into two cases:
        A. The first case arises when it is known that the attribute value is applicable, but it is **missing**—for example, if the height attribute of a person is listed as null.
        B. The second case arises when it is not known whether the attribute value applicable —for example, if the HomePhone attribute of a person is null.
- ***Example:*** Appartment-Number attribute:
    - *Not applicable*: if the address does not include a apartment-number.
    - *Missing*: an apartment-number exists but we do not know what it is.
    - *Unknown*: we do not know whether or not an apartment-number is part of the customer's address

Relationship

- A **relationship** is an association among at least two entities belonging to one or more entity sets. For example, EMPLOYEE Sami works for IT DEPARTMENT.

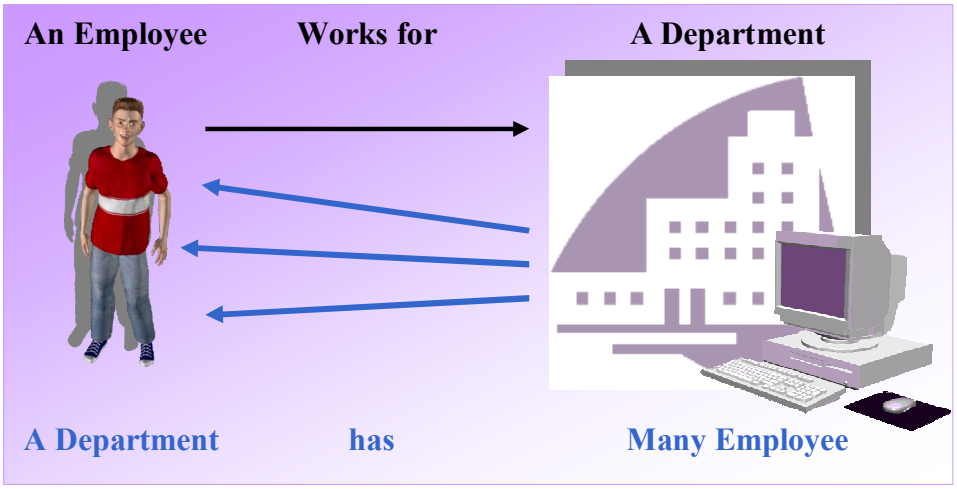
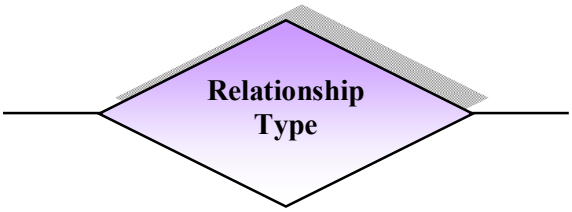


Figure 3.10: "Works for" Relationship set example.

- **Relationship types** associate entity types. For example, the WORKS_FOR relationship type in which EMPLOYEEs and DEPARTMENTs participate. They are pictured by Diamond nodes, and edges connecting to the related entity types



- ***Example:***

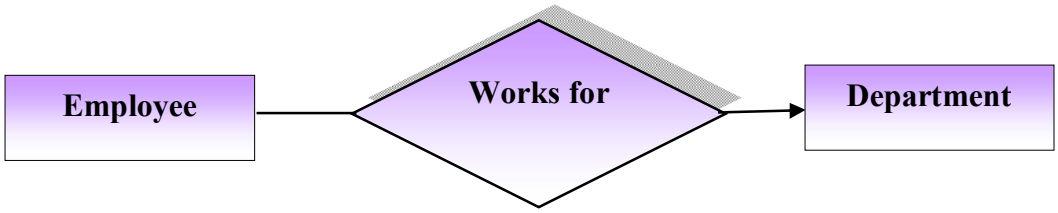


Figure 3.11: "Works for" Relationship set ER diagram.

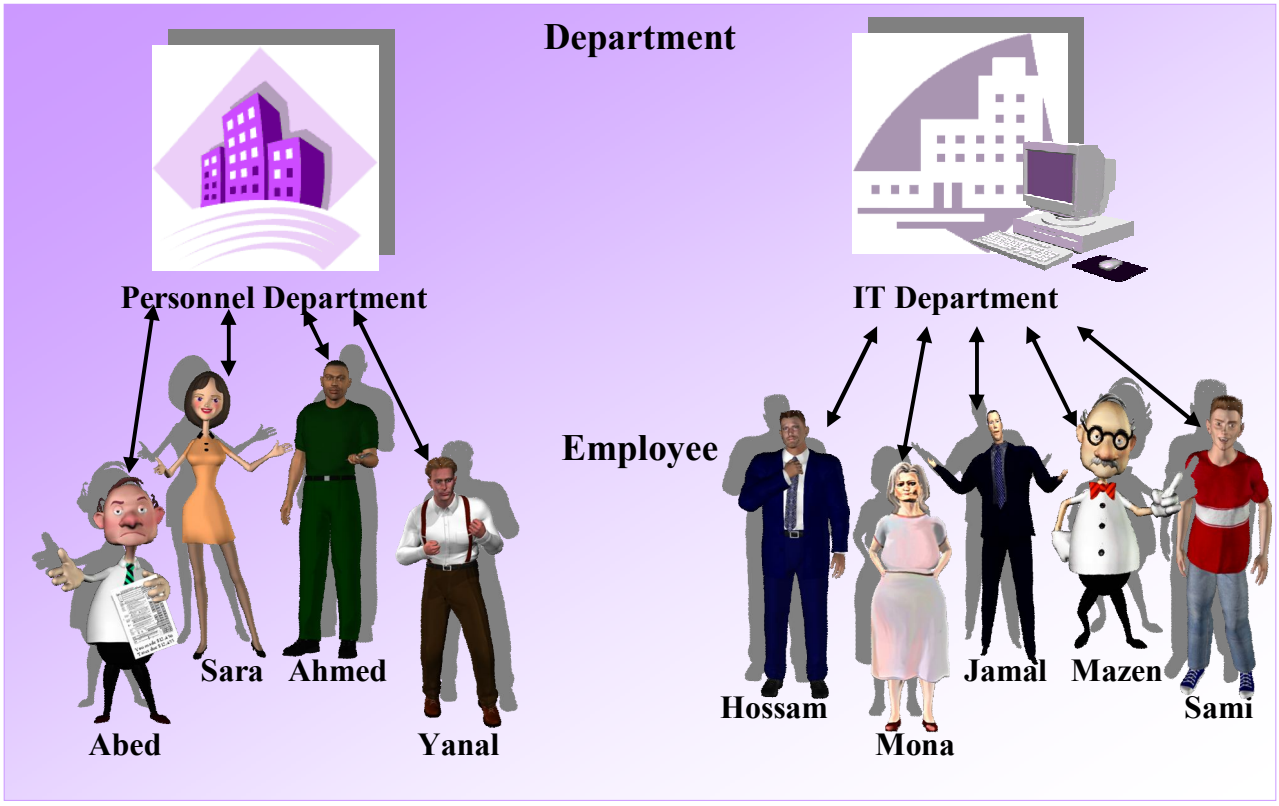
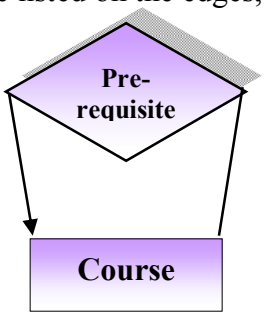


Figure 3.12: Relationship set.

- A **relationship set** is the collection of instances (i.e., relationships between objects) represented by a relationship type.

- **Relationship type vs. relationship set**
  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 types may associate an entity type with itself. In such a case, the **roles** of the entity types in the relationship type are listed on the edges, and the relationship is said to be **recursive**.



- More than one relationship type can exist with the same participating entity types; for example the WORKS_FOR relationship type in which EMPLOYEEs and DEPARTMENT participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.

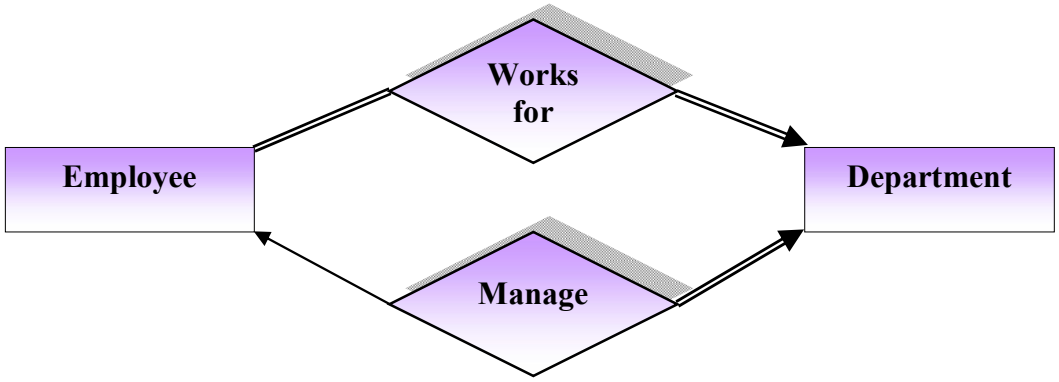


Figure 3.13:

- ***Example:*** We can have two binary relationship types associating the student and team types, TeamMemberOf and LeaderOf. In the former case, a student entity is a member of a team entity; in the later case, a student can be a leader of a team.

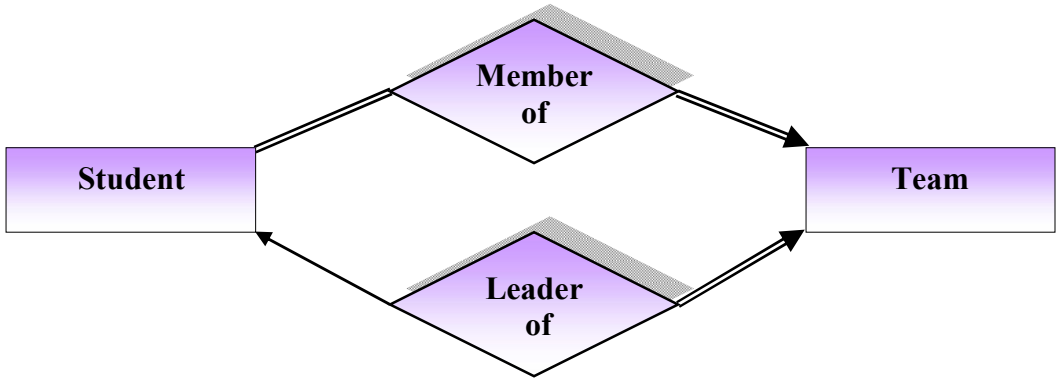


Figure 3.14:

- **A relationship type can have attributes called *descriptive* 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***

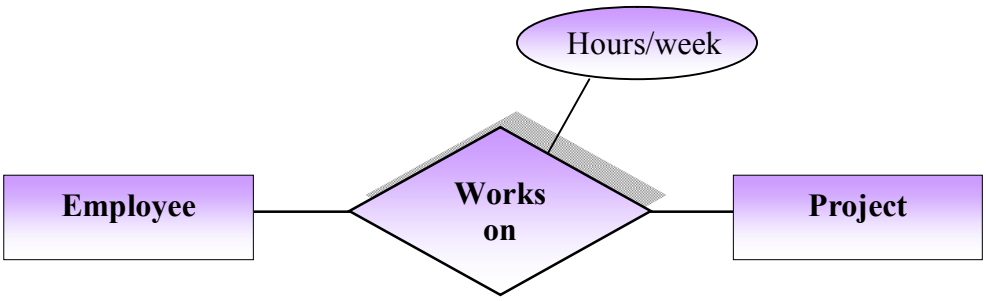


Figure 3.15:

- Example:

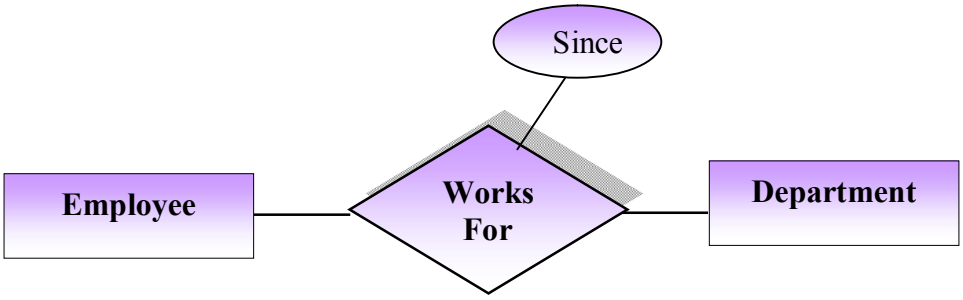


Figure 3.16:

What is the different between two the following figures?

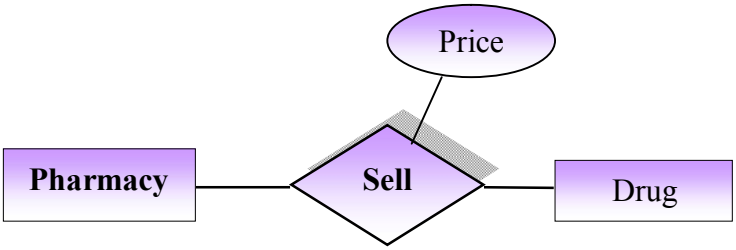


Figure (1)

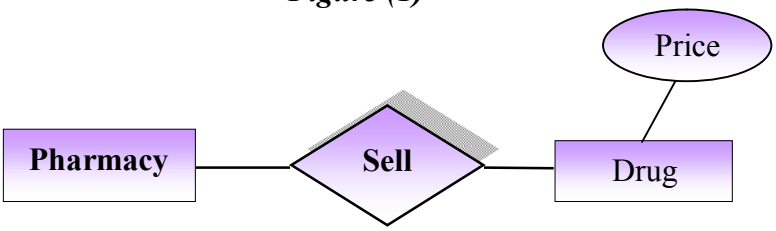


Figure2

The degree of a relationship type

- It is the number of participating entity types.

1. ***Unary:*** related to another of the same entity type. Also called recursive relationships.

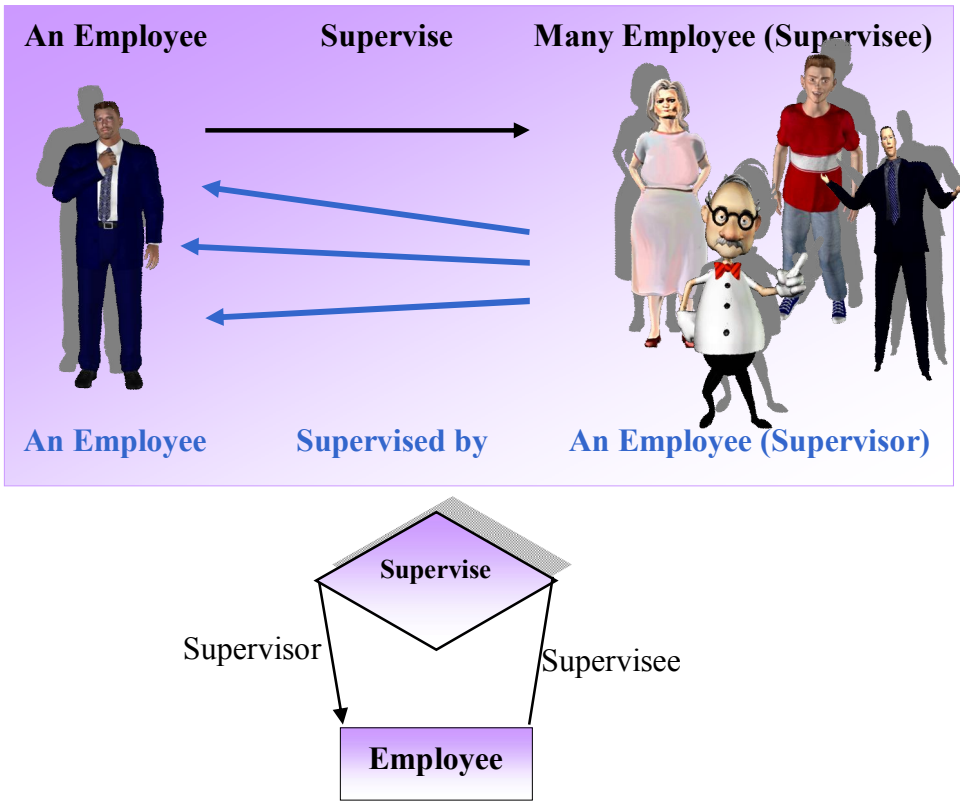


Figure 3.17: Unary Relationship type example.

2. ***Binary*:** entities of two different types related to each other. (Two participating entities).

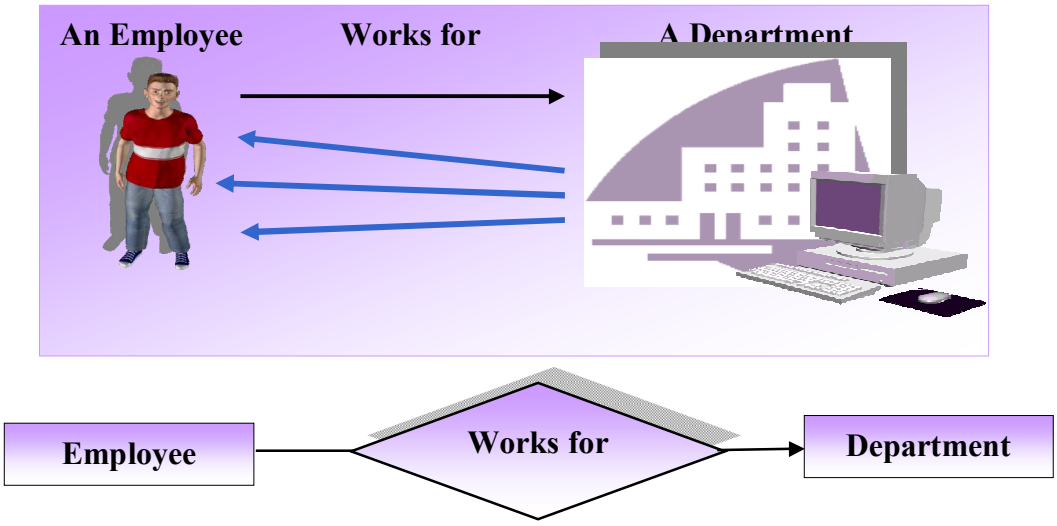


Figure 3.18: Binary Relationship type example.

3. ***Ternary:*** entities of three different types related to each other. (three participating entities).

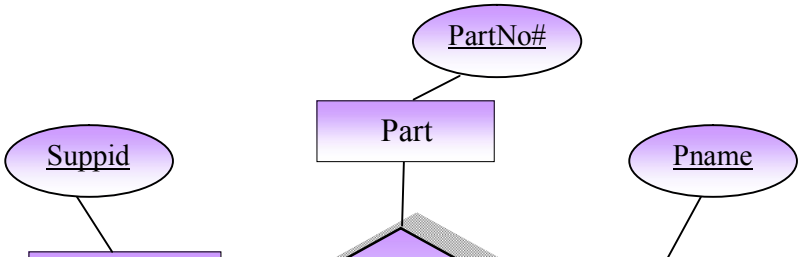


Figure 3.19: Ternary Relationship type example.

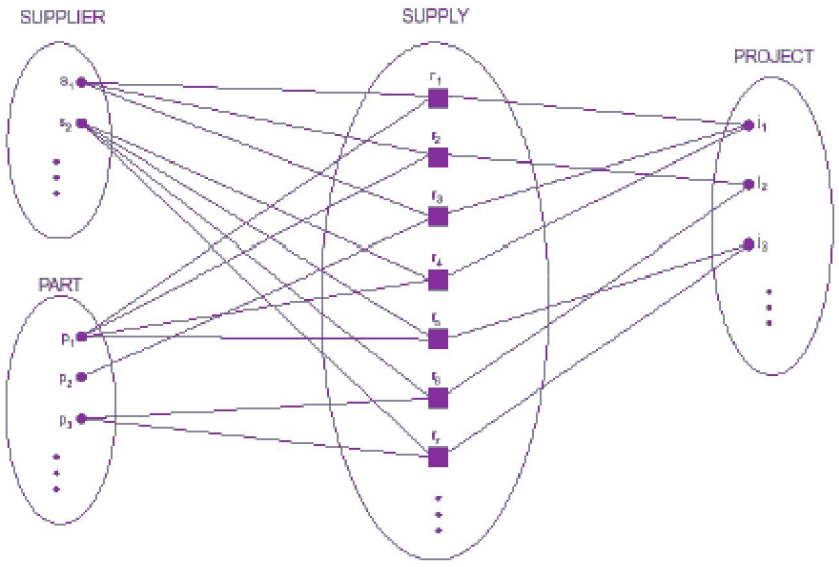


Figure 3.20: Ternary Relationship set example.

- Example:

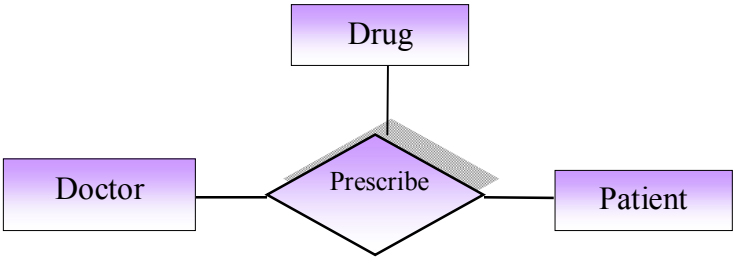


Figure 3.21: *"Prescribe"* ternary relationship type.

4. ***n-ary:*** entities of more than three different types related to each other.

3.3 Structural Constraints on relationship

Cardinality ratio (of a binary relationship)

- Mapping cardinalities, or cardinality ratio, express the number of entities to which another entity can be associated via a relationship set. (specifies maximum participation).
- There are four types of cardinality: 1:1, 1:N, N:1, or M:N.
  1. **One-to-One (1:1)** cardinality ratio, an entity in one set is associated with at most one entity in another.

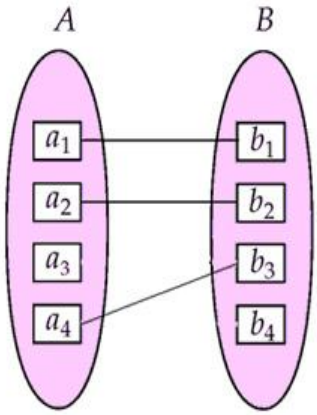
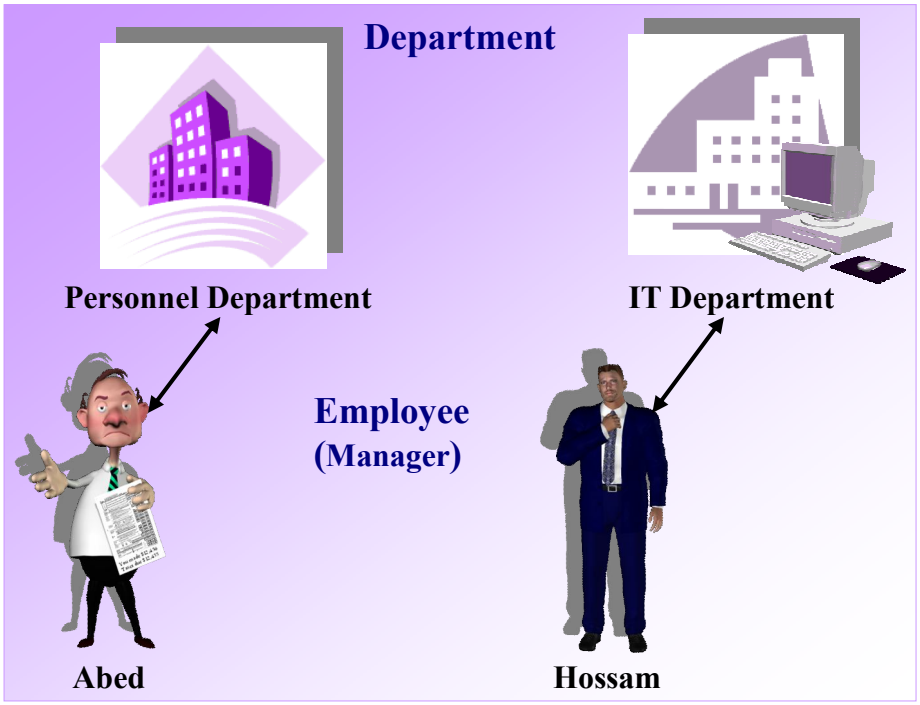


Figure 3.22: One-to-One relationship set.

- For Example:



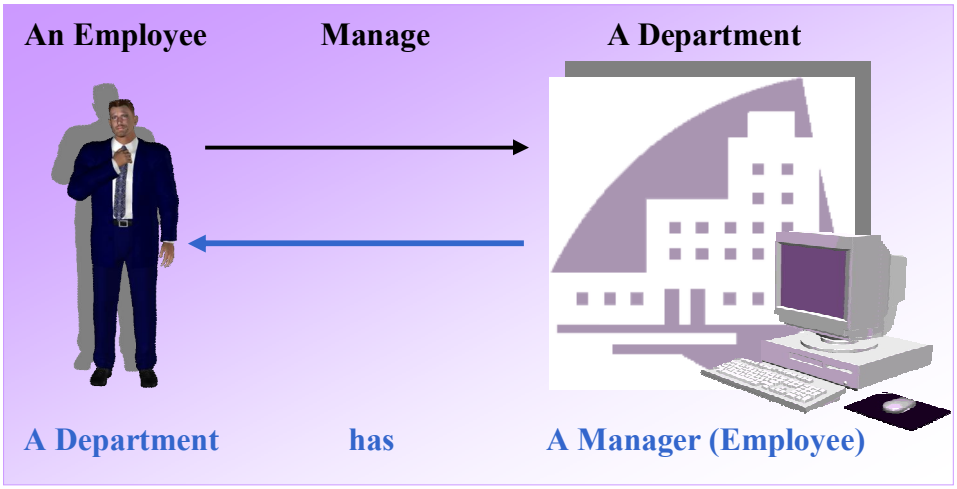


Figure 3.23: One-to-One relationship Example.

- The ER Diagram is:

Figure 3.24: One-to-One relationship ER diagram.

2. **One-to-many (1: N)** cardinality ratio, an entity in the first set is associated with 0 or more entities in the second set. However, those entities in the second set can be associated with at most one entity in the first.

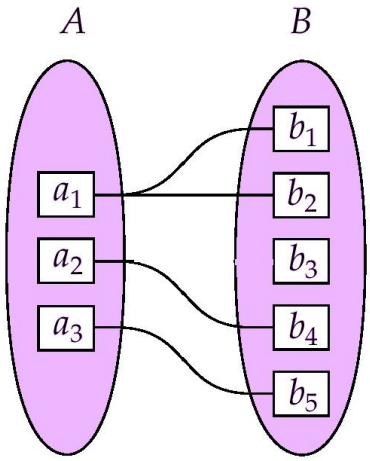


Figure 3.25: One-to-Many relationship set.

- For Example:

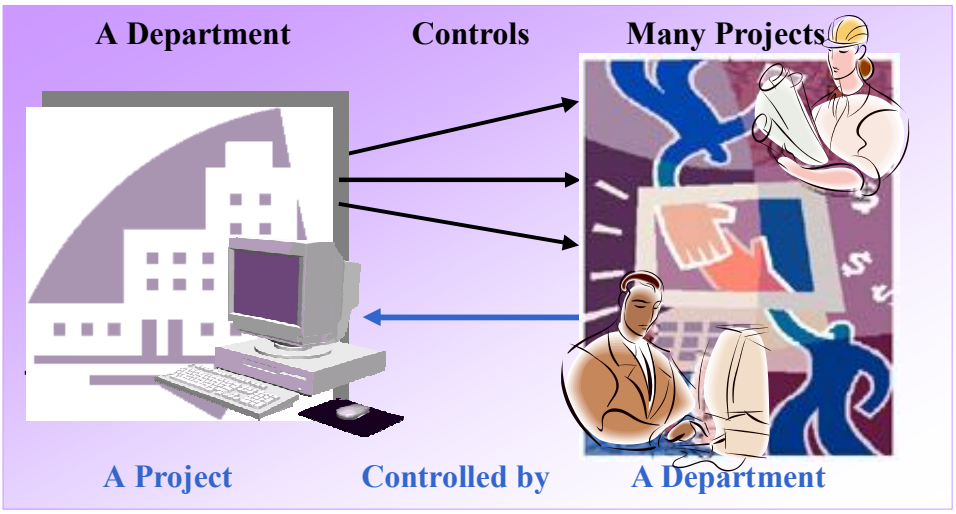


Figure 3.26: One-to-Many relationship set example.

- The ER-Diagram is:

Figure 3.27: One-to-Many relationship ER diagram.

3. **Many-to-one (N: 1)** cardinality ratio is just the reverse of the 1:N cardinality ratio. You can think of whichever entity set you like as being the first set. Just specify the direction that makes sense for your application, and then be consistent.

4. **Many-to-many (N:M)** cardinality ratio, entities of either set may be associated with any number of entities in the other.

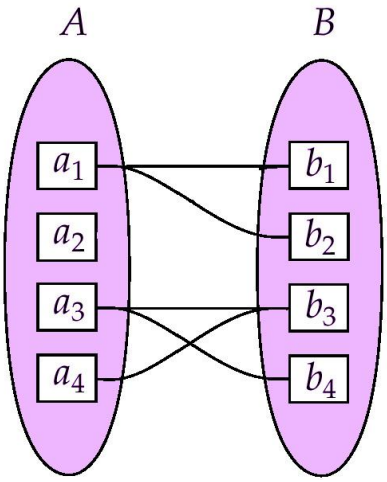
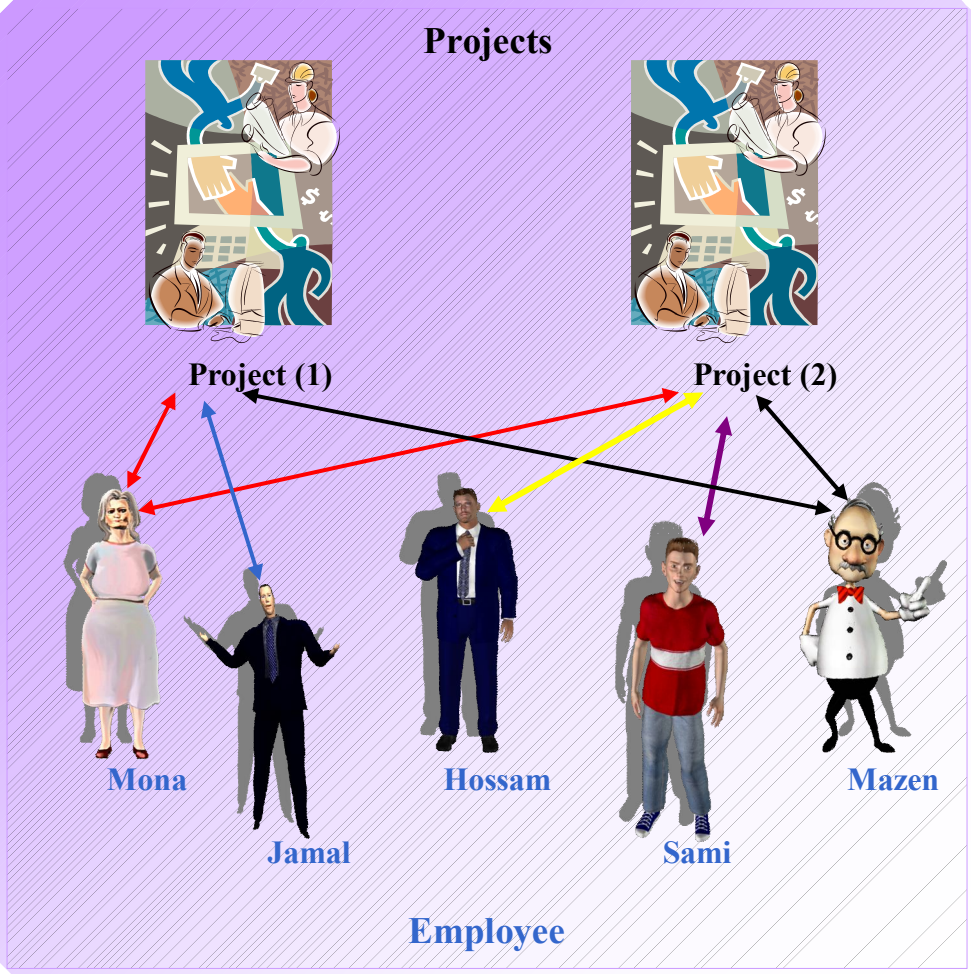


Figure 3.28: Many-to-Many relationship set.



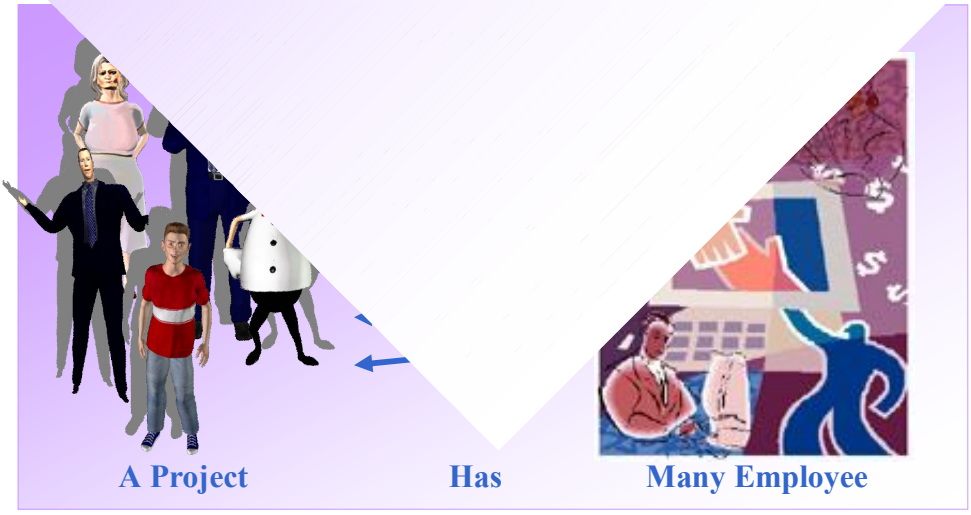


Figure 3.29: Many-to-Many relationship set example.

- The ER Diagram is:

Figure 3.30: Many-to-Many relationship ER diagram.

- Attributes of relationship types:
  - The descriptive attribute of 1:1 or 1:N relationship type can be migrated to one of the participating entity types:
    - In 1:1 relationship type, a descriptive attribute can be an attribute of either entity type 1 or entity type 2.
    - In 1:N relationship type, a descriptive attribute can be migrated only to the entity type at the N-side of the relationship.
  - For N:M relationship type, some attribute must be specified as descriptive attribute.

- The cardinality ratio depends on the real-world situation.
  - Example: there are two situation of the borrower relationship between customer and loan:
    - If a loan can belong to only one customer, and a customer can have several loans, then the relationship type from customer to loan is one-to-many.
    - If a loan can belong to several customers, the relationship type is many-to-many.

Participation Constraints (Existence Dependency Constraints)

- Specifies minimum participation
- Specifies whether the existence of an entity **e** □ entity type **E** depends on its being related to another entity via the relationship type **R**.
- The participation of an entity set E in a relationship set R is said to be:

1. **Partial**: if only some entities in E participate in relationships in R.
   - Each entity e □ E ***can*** participate in a relationship set.
   - Optional participation, not existence-dependent.
   - The minimum value is zero.
   - Shown by a single line.

2. **Total**: if every entity in E participates in at least one relationship in R.
   - Each entity e □ E ***must*** participate in a relationship set.
   - mandatory participation, existence-dependent
     - Example: if a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in a works-for relationship instance.
   - The minimum value is one or more.
   - Every entity in E participates in at least one relationship in R
   - Shown by double line.

Figure 3.31: Total/Partial ER diagram.

Alternative (min, max) notation for relationship structural constraints:

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least min and at most max relationship instances in R

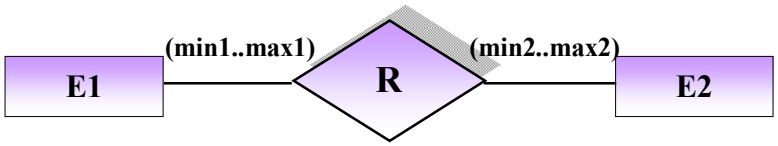


Figure 3.32: Min..Max ER diagram.

- Default (no constraint): min=0, max=n (signifying no limit).
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints
- Examples: A department has exactly one manager and an employee can manage at most one department.
    - Specify (0..1) for participation of EMPLOYEE in MANAGES
    - Specify (1..1) for participation of DEPARTMENT in MANAGES

Figure 3.33: one-to-one relationship type example using Min..Max notation.

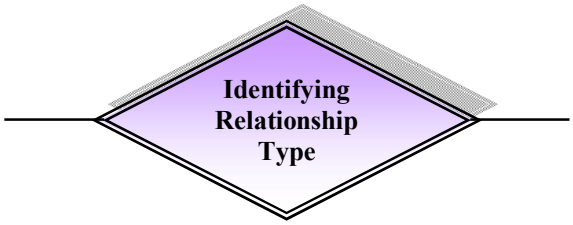
- An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1..1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0..*) for participation of DEPARTMENT in WORKS_FOR

Figure 3.34: many-to-one relationship type example using Min..Max notation.

3.4 Weak Entity Types

- An entity type that does not have a key attribute.
- It is pictured by

- A weak entity type must participate in an **identifying relationship type** with an **owner** or **identifying entity type**.
- **Identifying relationship type** is pictured by diamond with double lines.



- Entities are identified by the combination of:
  - A partial key of the weak entity type .It is underlined with a dashed or dotted line.

  - The particular entity they are related to in the identifying entity type
- A weak entity type may have more than one identifying entity type and an identifying relationship type of degree higher than two.

- **For example:**

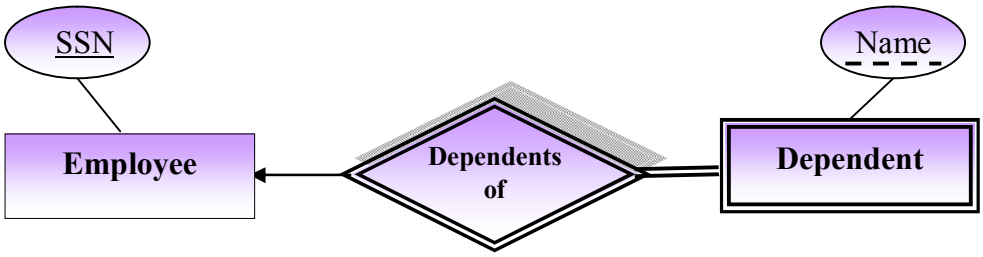


Figure 3.35: Weak entity Example.

  - 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.

3.5 An Example of Database Application (COMPANY Database)

- The database designers stated the following description of the "miniworld"—the part of the company to be represented in the database:
  1. 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.
  2. A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
  3. We store each employee's name, social security number, address, salary, sex, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
  4. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

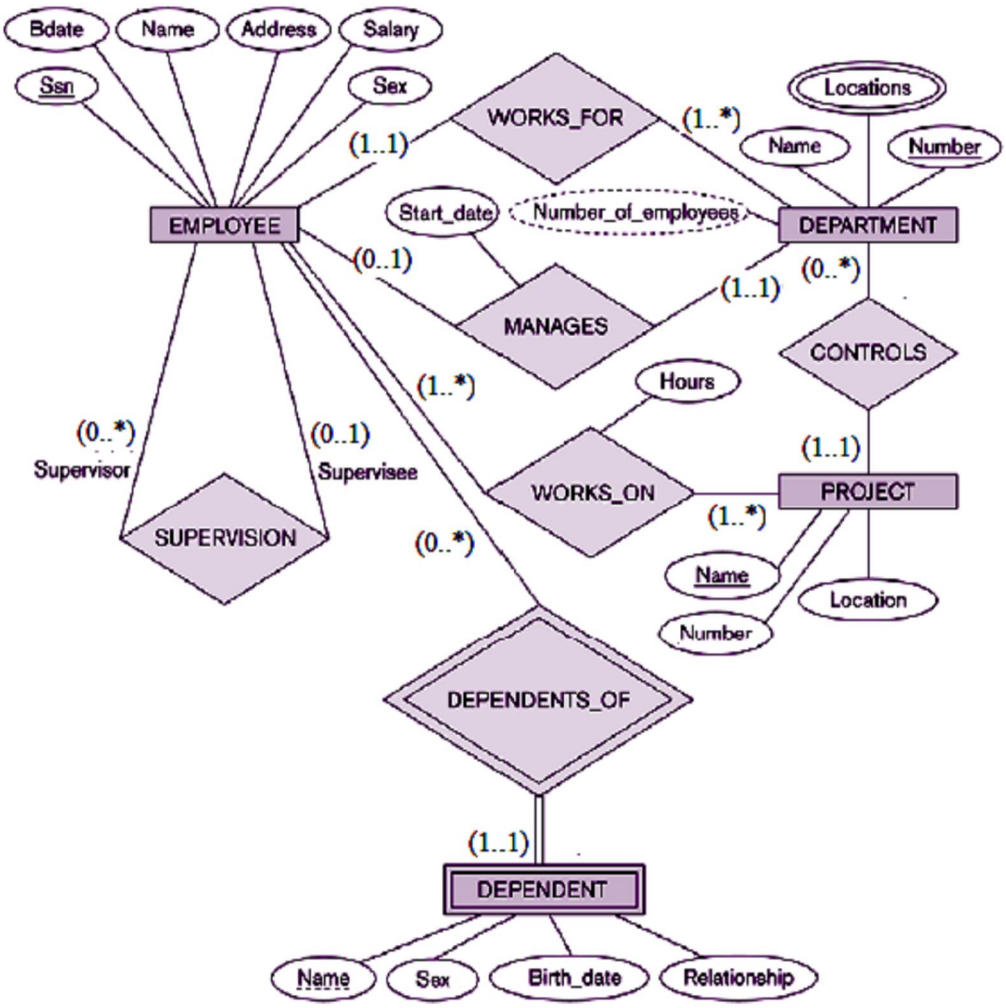


Figure 3.36: The ER Diagram for the company DB.

Consider the following information about a <u>university database</u>:

- Professors have an SSN, a name, a rank, and a research specialty.
- Projects are identified by a project number, more than one sponsor name (e.g., NSF), a starting date, an ending date, and a budget.
- Graduate students have an SSN, a name, and a current degree program (e.g., M.S.or Ph.D.).
- Each project is managed by one professor, each project is worked on by one or more professors and each project is worked on by one or more graduate students.
- When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects.
- Departments have a department number, a department name, and a main office.
- Departments have a professor (known as the chairman) who runs the department.
- Professor works in one or more departments and for each department that they work in, a time percentage is associated with their job.

Design and draw an ER diagram that captures the information about the University.