Chapter 2. Conceptual Modeling - Introduction to Database Systems

CHAPTER 2

CONCEPTUAL MODELING

After reading this chapter, the reader will understand:

- *Basic concept of entity-relationship model*
- *Entities and their attributes*
- *Entity types and entity set*
- *Different types of attributes*
- *Key attribute, and the concept of superkey, candidate key and primary key*
- *Concept of weak and strong entity types*
- *Relationships among various entity types*
- *Constraints on relationship types*
- *Binary versus ternary relationships*
- *E-R diagram notations*
- *Features of enhanced E-R model that include specialization, generalization, and aggregation*
- *Constraints on specialization and generalization*
- *Alternative notations of E-R diagram*
- *Unified modeling language (UML), a standard language used to implement object data modeling*
- *Categories of UML diagrams that include structural diagrams and behavioural diagrams*

As discussed in the earlier chapter, the overall database design and implementation process starts with requirement collection and analysis and specifications. Once the user's requirements have been specified, the next step is to construct an abstract or conceptual model of a database based on those requirements. This phase is known as

conceptual modeling or **semantic modeling**. Conceptual modeling is an important phase in designing a successful database. It represents various pieces of data and their relationships at a very high-level of abstraction. It mainly focuses on what data is required and how it should be organized rather than what operations are to be performed on the data.

The conceptual model is a concise and permanent description of the database requirements that facilitates an easy communication between the users and the developers. It is simple enough to be understood by the end users; however, detailed enough to be used by a database designer to create the database. The model is independent of any hardware (physical storage) and software (DBMS) constraints and hence can be modified easily according to the changing needs of the user. The conceptual model can be represented using two major approaches, namely, *entity-relationship modeling* and the *object modeling*.

This chapter discusses the entity-relationship (E-R) model and its extended features, which are used for designing a conceptual model of any database application. It also discusses the entity-relationship diagram, a basic component of the E-R model used to graphically represent the data objects. An E-R diagram helps in modeling the data representation component of a software system. The other components of the software system such as user interactions with the system, functional modules and their interactions, etc. are specified with the help of Unified Modeling Language (UML), which is a standard methodology used in object modeling.

2.1 CONCEPTS OF ENTITY-RELATIONSHIP MODEL

The entity-relationship (E-R) model is the most popular conceptual model used for designing a database. It was originally proposed by Dr. Peter Chen in 1976 as a way to unify the network and relational database views. The E-R model views the real world as a set of basic objects (known as **entities**), their characteristics (known as **attributes**), and associations among these objects (known as **relationships**). The *entities, attributes*, and *relationships* are the basic constructs of an E-R model.

The information gathered from the user forms the basis for designing the E-R model. The *nouns* in the requirements specification document are represented as the entities, the *additional nouns* that describe the nouns corresponding to entities form the attributes, and the *verbs* become the relationships among various entities. The E-R model has several advantages listed as follows.

Things to Remember

The E-R model does not actually gives us a database description; rather it gives us an intermediate step from which we can easily create a database.

- It is simple and easy to understand and thus, can be used as an effective communication tool between database designers and end users.
- It captures the real-world data requirements in a simple, meaningful, and logical way.
- It can be easily mapped onto the relational model. The basic constructs, that is, the entities and attributes of the E-R model can be easily transformed into relations (or tables) and columns (or fields) in a relational model.
- It can be used as a design plan and can be implemented in any database management software.

2.1.1 Entity and its Attributes

An **entity** is any distinguishable object that has an independent existence in the real world. It includes all those 'things' of an organization about which the data is collected. For example, each book, publisher, and author in the *Online Book* database (discussed in Chapter 01) is an entity. An entity can exist either physically or conceptually. If an entity has a physical existence, it is termed as **tangible** or **concrete entity** (for example, a book, an employee, a place or a part) and if it has a conceptual existence, it is termed as **non-tangible** or **abstract entity** (for example, an event, a job title or a customer account).

Each entity is represented by a set of attributes. The **attributes** are the properties of an entity that characterize and describe it. Figure 2.1 shows the `BOOK` entity b_1 described by the attributes `Book_title`. `Price` (in dollars), `ISBN`, `Year`, `Page_count` and `Category`. Similarly, the `PUBLISHER` entity p_1 is described by the attributes `P_ID`, `Name`, `Address`, `Phone` and `Email_ID`. Here, b_1 and p_1 are the instances of the entity types. The entity types and instances are discussed later in this section.

Each attribute has a particular **value**. For example, the attributes `Book_title`, `Price`, `ISBN`, `P_ID`, `Year`, `Page_count` and `Category` can have the values *Ransack, 22, 001-354-921-1, 2006, 200* and *Novel*, respectively, (as shown in Figure 2.1). Similarly, the attributes `P_ID`, `Name`, `Address`, `Phone` and `Email_ID` can have the values *P001, Hills Publications, 12 Park Street Atlanta Georgia 30001, 7134019, h_pub@hills.com*, respectively. Each attribute can accept a value from a set of permitted values, which is called the **domain** or the **value set** of the attribute. For example, the domain of the attribute `Page_count` is a set of positive integers. On the other hand, the category of a book can be textbook, language book or novel. Thus, the domain of the attribute `Category` is {*Textbook, Language book, Novel*}.

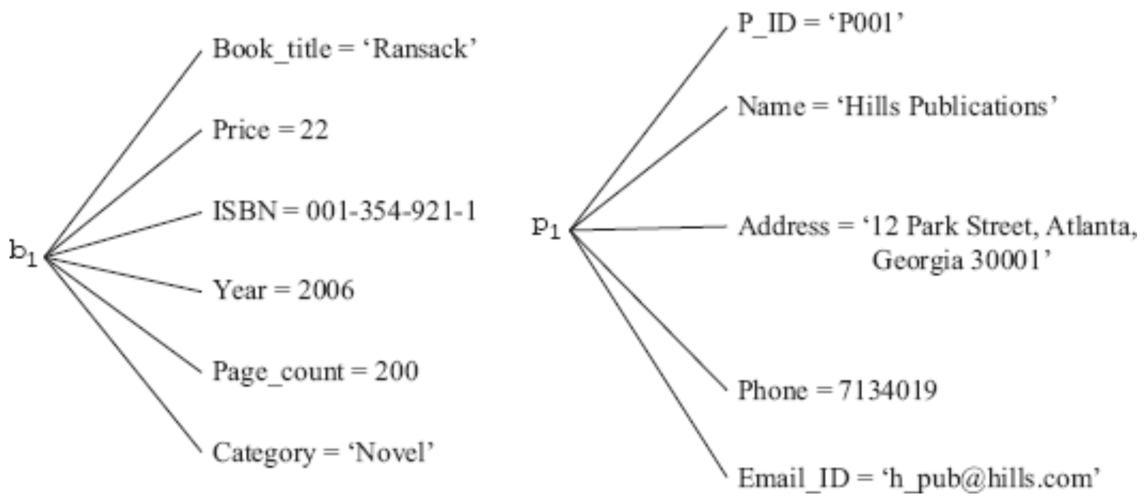
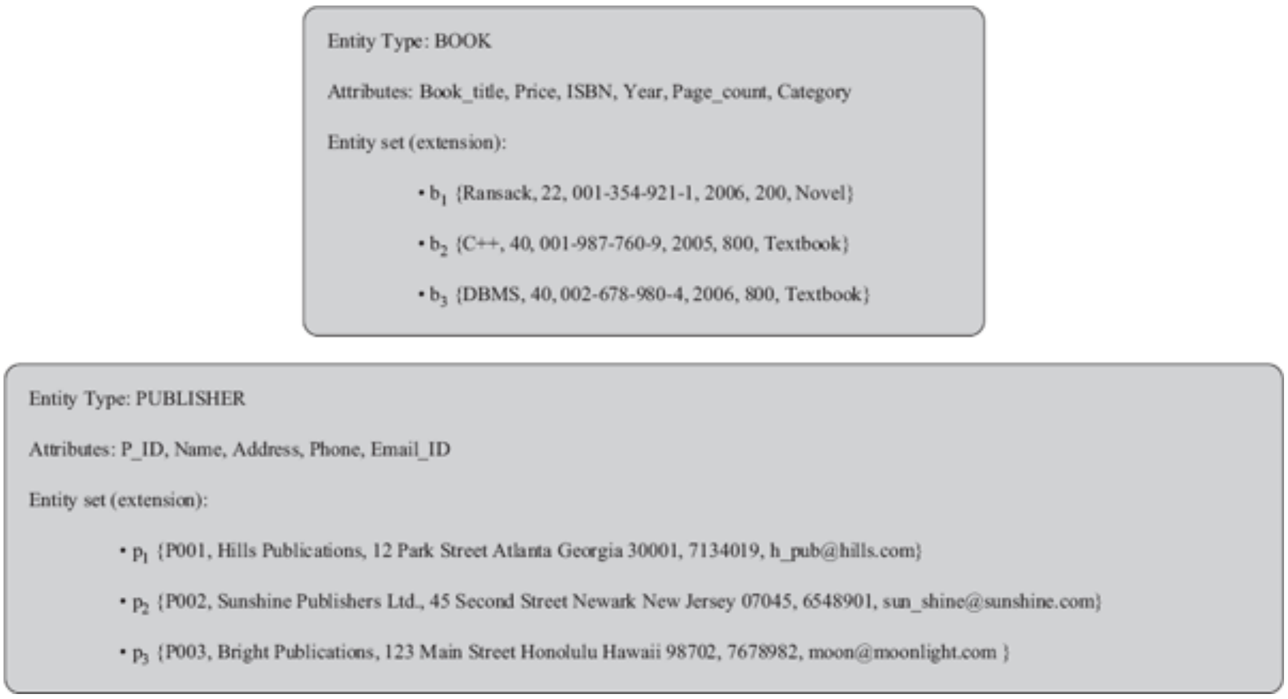


Fig. 2.1 *Entity instances b_1 (BOOK) and p_1 (PUBLISHER), and their attributes*

Entity Types and Entity Sets

A set or a collection of entities that share the same attributes but different values, is known as an **entity type**. For example, the set of all publishers who publish books can be defined as the entity type PUBLISHER. Similarly, a set of books written by the authors can be defined as the entity type BOOK. A specific occurrence of an entity type is called its **instance**. For example, the novel *Ransack* and the publisher *Hills Publications* are the instances of the entity types BOOK and PUBLISHER, respectively.

The collection of all instances of a particular entity type in the database at any point of time is known as an **entity set**. Each entity set is referred to by its name and attribute values. Note that both the entity set and the entity type are usually referred to by the same name. For example, BOOK refers to both the type of entity and the current set of all book entities. An entity type describes the **schema** (**intension**) for the entity set that shares the same structure. The entity set, on the other hand, is called the **extension** of the entity type. Figure 2.2 shows two entity types, BOOK and PUBLISHER, and a list of attributes for each of them. Some individual instances of each entity type are also shown along with their attribute values.


Entity Type: BOOK

Attributes: Book_title, Price, ISBN, Year, Page_count, Category

Entity set (extension):

- b_1 {Ransack, 22, 001-354-921-1, 2006, 200, Novel}
- b_2 {C++, 40, 001-987-760-9, 2005, 800, Textbook}
- b_3 {DBMS, 40, 002-678-980-4, 2006, 800, Textbook}

Entity Type: PUBLISHER

Attributes: P_ID, Name, Address, Phone, Email_ID

Entity set (extension):

- p_1 {P001, Hills Publications, 12 Park Street Atlanta Georgia 30001, 7134019, h_pub@hills.com}
- p_2 {P002, Sunshine Publishers Ltd., 45 Second Street Newark New Jersey 07045, 6548901, sun_shine@sunshine.com}
- p_3 {P003, Bright Publications, 123 Main Street Honolulu Hawaii 98702, 7678982, moon@moonlight.com }

Fig. 2.2 *Entity types and entity sets*

Types of Attributes

The attributes of an entity are classified into the following categories:

- **Identifying and descriptive attributes:** The attribute that is used to uniquely identify an instance of an entity is known as **identifying attribute** or simply an **identifier**. For example, the attribute `P_ID` of the entity type `PUBLISHER` is identifying attribute, as two publishers cannot have the same publisher ID. Similarly, the attribute `ISBN` of the entity type `BOOK` is also an identifier, as two different books cannot have the same ISBN. A **descriptive attribute** or simply a **descriptor**, on the other hand, describes a non-unique characteristic of an entity instance. For example, the attributes `Price` and `Page_count` are the descriptive attributes as two books can have the same price and number of pages.
- **Simple and composite attributes:** The attributes that are indivisible are known as **simple** (or **atomic**) **attributes**. For example, the attributes `Book_title`, `Price`, `Year`, `Page_count`, and `Category` of the entity type `BOOK` are simple attributes as they cannot be further divided into smaller subparts. On the other hand, **composite attributes** can be divided into smaller subparts. For example, the attribute `Address` can be further divided into `House_number`, `Street`, `City`, `State` and `Zip_code` (as shown in Figure 2.3). Composite attributes help in grouping the related attributes together. They are useful in those situations where the user sometimes wants to refer to an entire attribute and sometimes to only a component of the attribute. In case the user does not want to refer to the individual components, there is no need to subdivide the composite attribute into its component attributes.

  Fig. 2.3 *Composite attribute*

- **Stored and derived attributes:** In some cases, two or more attribute values are related in such a way that the value of one attribute can be determined from the value of other attributes or related entities. For example, if the entity type PUBLISHER has an attribute Books_published, then it can be calculated by counting the book entities associated with that publisher. Thus, the attribute is known as **derived attribute**. Similarly, consider an entity type PERSON and its attributes Date_of_birth and Age. The value of the attribute Age can be determined from the current date and the value of Date_of_birth. Thus, the attribute Age is a derived attribute and the attribute Date_of_birth is known as **stored attribute**.

- **Single-valued and multivalued attributes:** The attributes that can have only one value for a given entity are called the **single-valued attributes.** For example, the attribute Book_title is a single-valued attribute as one book can have only one title. However, in some cases, an attribute can have multiple values for a given entity. Such attributes are called **multivalued attributes**. For example, the attributes Email_ID and Phone of the entity type PUBLISHER are multi-valued attributes, as a publisher can have zero, one or more e-mail IDs and phone numbers. The multivalued attributes can also have lower and upper bounds to restrict the number of values for each entity. For example, the lower bound of the attribute Email_ID can be specified to zero and upper bound to three. This implies that each publisher can have none or at most three e-mail IDs.
- **Complex attributes:** The attributes that are formed by arbitrarily nesting the composite and multivalued attributes are called **complex attributes**. The arbitrary nesting of attributes is represented by grouping components of a composite attribute between parentheses '()' and by displaying multivalued attributes between braces '{}'. These attributes are separated by commas. For example, a publisher can have offices at different places each with different address and each office can have multiple phones. Thus, an attribute Address_phone for a publisher can be specified as given in Figure 2.4. Here, Phone is a multivalued attribute and Address is a composite attribute.

Fig. 2.4 *Complex attribute* — *`Address_phone`*

Null Values for the Attributes

Sometimes, there may be a situation where a particular entity may not have an appropriate value for an attribute. In such situations, the attribute takes a special value called ***null*** value. A null value of an attribute may indicate 'not applicable'. For example, consider an entity type `PERSON` with three attributes `First_name`, `Middle_name` and `Last_name`. Since, not every person has a middle name, the person without any middle name will have a *null* value for the attribute `Middle_name`. It implies that the attribute `Middle_name` is not applicable to that person. A *null* value can also be used in situations where the value of a particular attribute is unknown. An unknown category can further be classified as *missing* (the value that exists, but is not available) and *not known* (it is not known that the value exists or not). For example, if the attribute `Book_title` has the value *null*, it indicates that the title of the book is missing, as every book must have a title. However, if the attribute `Phone` has the *null* value, it indicates that it is not known whether the value for `Phone` exists or not.

Key Attribute

The attribute (or combination of attributes) whose values are distinct for each individual instance of an entity type is known as a **key attribute**. It is used to uniquely identify each instance of an entity type. That is, no two entities can have the same value for the key attribute. For example, the attribute `ISBN` of the entity type `BOOK` is the key attribute, as two different books cannot have the same ISBN. An entity type may have more than one key attributes. For example, the attributes `P_ID` and `Name` of the entity type `PUBLISHER` are the key attributes, as no two publishers can have the same publisher ID and publisher name.

Superkey, Candidate Key, and Primary Key A set of one or more attributes, when taken together, helps in uniquely identifying each entity is called a **superkey**. For example, the attribute `P_ID` of the entity type

`PUBLISHER` can be used to uniquely identify each entity instance. Thus, `P_ID` is a superkey. Similarly, the attribute `Name` of the entity type `PUBLISHER` is also a superkey. The combination of the attributes `P_ID` and `Name` is also a superkey. One can also use the set of all attributes (`P_ID`, `Name`, `Address`, `Phone`, `Email_ID`) to uniquely identify each entity instance of type `PUBLISHER`. Thus, the combination of all attributes is also a superkey.

However, `P_ID` itself can uniquely identify each entity instance, thus, other attributes are not required. Such a minimal superkey that does not contain any superfluous (extra) attributes in it is called a **candidate key**. A candidate key contains a minimized set of attributes that can be used to uniquely identify a single entity instance. For example, the attributes `P_ID` and `Name` are the candidate keys. However, the combination of `P_ID` and `Name` is not a candidate key. The candidate key, which is chosen by the database designer to uniquely identify entities, is known as the **primary key**. For example, the attribute `P_ID` can be chosen as the primary key. If a primary key is formed by the combination of two or more attributes, it is known as **composite key**. Note that a composite key must be minimal, that is, all attributes that form the composite key must be included in the key attribute to uniquely identify the entities.

An entity type can have only one primary key; however, it can have multiple superkeys and candidate keys. $\boxed{X}$

The attributes whose values are never or rarely changed are chosen as primary key. For example, the attribute `Address` cannot be chosen as a part of primary key as it is likely to be changed. Moreover, the attributes, which can have *null* values, cannot be taken as primary key. For example, the attributes `Phone` and `Email_ID` can have *null* values, as it is not necessary for every publisher to have an email ID or phone number.

Weak and Strong Entity Types

An entity type that does not have any key attribute of its own is called a

weak entity type. On the other hand, an entity type that has a key attribute is called a **strong entity type**. The weak entity is also called a *dependent entity* as it depends on another entity for its identification. The strong entity is called an *independent entity*, as it does not rely on another entity for its identification. The weak entity has no meaning and relevance in an E-R diagram if shown without the entity on which it depends. For example, consider an entity type `EDITION` with attributes `Edition_no` and `Type` (Indian or International). It depends on another entity type `BOOK` for its existence, as an edition cannot exist without a book. Thus, `EDITION` is a weak entity type and `BOOK` is a strong entity type.

A weak entity type does not have its own identifying attribute that can uniquely identify each entity instance. It has a partial identifying attribute, which is known as **partial key** (or **discriminator**). A partial key is a set of attributes that can uniquely identify the instances of a weak entity type that are related to the same instance of a strong entity type. For example, the attribute `Edition_no` of the entity type `EDITION` can be taken as a partial key.

The primary key of a weak entity type is formed by the combination of its discriminator and the primary key of the strong entity type on which it depends for its existence. Thus, the primary key of the entity type `EDITION` is `(ISBN, Edition_no)`. Since, the entities belonging to a weak entity type are identified by the attributes of a strong entity type in combination of one of their attribute, the strong entity types are known as **identifying** or **parent** entity type. The weak entity type, on the other hand, is known as **child** or **subordinate** entity type.

2.1.2 Relationships

An association between two or more entities is known as a **relationship**. A relationship describes how two or more entities are related to each other. For example, the relationship `PUBLISHED_BY` (shown in Figure 2.5) associates a book with its publisher. Like entity types and entity sets, relationship types and a relationship sets also exist. Consider *n* possibly distinct entity types $E_1, E_2, \dots, E_n$(where, $n >= 2$). A **relationship type** R among these n entity types defines a set of associations (known as **relationship set)** among instances from these entity types. Like entity type and entity set, the relationship type and relationship set are also referred by the same name, say R. A **relationship instance** represents an association between individual entity instances. For example, an association between the entity types BOOK and PUBLISHER represents a relationship type; however, an association between the instances C++ and P001 represents a relationship instance.

Each relationship instance includes only one instance of each entity type that is participating in the relationship.

Mathematically, a relationship type R can be defined as a subset of the Cartesian product $E_1 \times E_2 \times \dots \times E_n$. Similarly, the relationship set R can be defined as a set of relationship instances r_i, where each r_j associates n individual entity instances $e_1, e_2, \dots, e_n$ of each entity type $E_1, E_2, \dots, E_n$, respectively. Each of the individual entity types $E_1, E_2, \dots, E_n$ is said to **participate** in the relationship type R and each of the individual entity instances $e_1, e_2, \dots, e_n$ is said to participate in the relationship instance r_i. Figure 2.5 shows a relationship type PUBLISHED_BY between two entity types BOOK and PUBLISHER, which associates each book with its publisher. Here, $r_1, r_2, r_3, \dots, r_n$ denote relationship instances.

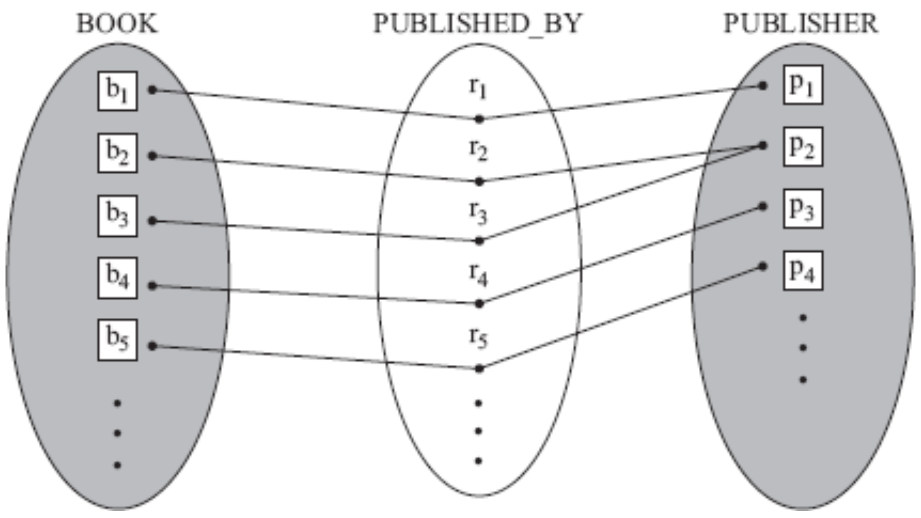


Fig. 2.5 *Relationship* PUBLISHED_BY *between two entity types* BOOK *and*

A relationship type may also have descriptive attributes. Consider two entity types AUTHOR and BOOK, which are associated with the relationship type REVIEWS. {Each author reviews some books and gives rating to that book. An attribute Rating could be associated with the relationship to specify the rating of the book given by the author (see Figure 2.6).} Eg:

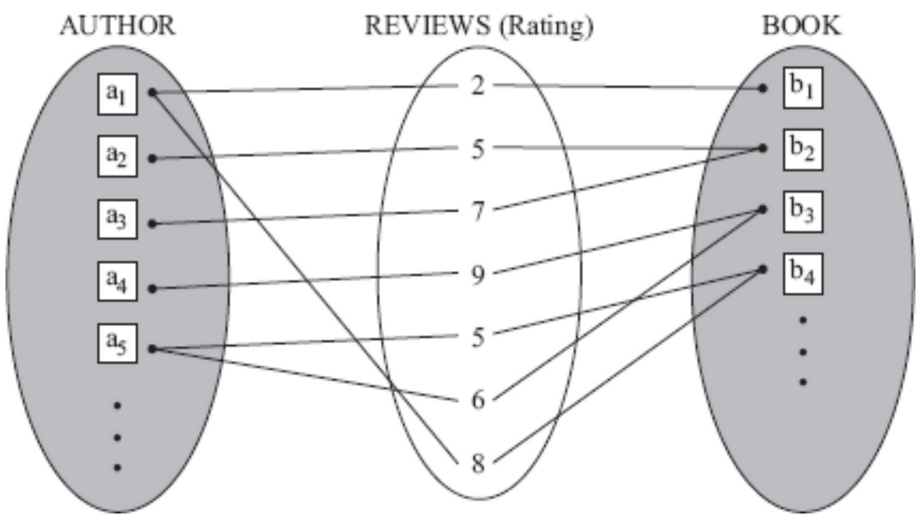


Fig. 2.6 *Relationship type REVIEWS with attribute Rating*

Identifying Relationship

The relationship between a weak entity type and its identifying entity type is known as **identifying relationship** of the weak entity type. For example, consider the entity types EDITION (weak entity type) and BOOK (strong entity type) and a relationship type HAS that exists between them. It specifies that each book has an edition. Since, an edition cannot exist without a book, the relationship type HAS is an identifying relationship as it associates a weak entity type with its identifying entity type.

Relationship Degree

The number of entity types participating in a relationship type determines the **degree of the relationship type**. The relationship that involves just one entity type is called **unary relationship**. In other words, the relationship that exists between the instances of the same entity type is a unary relationship. The relationship between two entity types is known as

binary relationship and the relationship between three entity types is known as **ternary relationship**. In general, the relationship between *n* entity types is known as *n*-ary relationship. Thus, a unary relationship has a degree 1, a binary relationship has a degree 2, a ternary relationship has a degree 3, and an *n*-ary relationship has a degree *n*. The binary relationships are the most common relationships. For example, the relationship type `PUBLISHED_BY` between the entity types `BOOK` and `PUBLISHER` is a binary relationship.

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a specific function or a role in the relationship. The role of each participating entity type is represented by the **role name**. If the participating entity types are distinct, their role names are implicit and are not usually specified. For example, in `PUBLISHED_BY` relationship, the entity type `BOOK` plays the role of a *book* and the entity type `PUBLISHER` plays the role of a *publisher*. Thus, the role names in this relationship are implicit.

Now, consider another entity `AUTHOR` which represents the authors who can write books as well as review the books written by other authors and give feedbacks to them. Thus, an author can be a writer as well as a reviewer. Thus, in a relationship type `FEEDBACKS` (given in Figure 2.7), the entity type `AUTHOR` plays two different roles (author and a reviewer). Such a relationship is called a **recursive relationship**. In such relationships, it is necessary to explicitly specify the role names to distinguish the meaning of each participation. Figure 2.7 shows a recursive relationship `FEEDBACKS`, where each relationship instance r_i has two lines each marked by ‘r’ for the role of a reviewer and ‘w’ for the role of an author, respectively. Here, the entity instance a_1 plays the role of a reviewer for the instances a_2, a_3, and a_4 and the instance a_2 plays the role of a reviewer for the instance a_5.

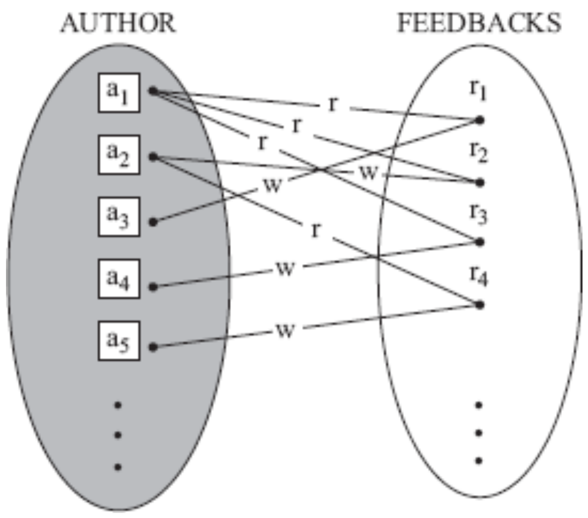


Fig. 2.7 *Recursive relationship*

A recursive relationship is always a unary relationship as it represents relationship between the instances of the same entity type.

Constraints on Relationship Type

Each relationship type has certain constraints that restrict the possible combination of instances participating in the relationship set. The constraints on the relationship type are of two types, namely, *mapping cardinalities* and *participation constraint*. These two constraints are collectively known as **structural constraints** of a relationship type.

Mapping Cardinalities The maximum number of instances of one entity type to which an instance of another entity type can relate to is expressed by the **mapping cardinalities** or **cardinality ratio**. The mapping cardinalities can be used to describe the relationships between two or more entities. However, they are most commonly used to describe the binary relationships. For a binary relationship between two entity types E_1 and E_2, the mapping cardinalities can be of four types (see Figure 2.8).

Learn More

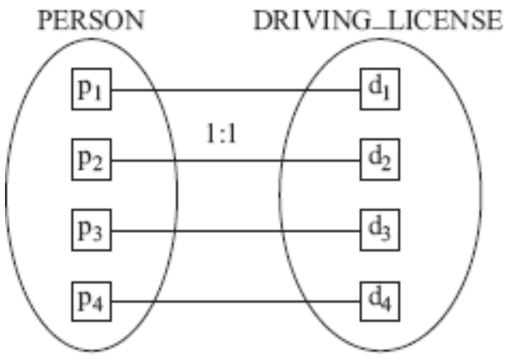
Like a binary relationship, a ternary relationship also has four types of mapping cardinalities, namely, one-to-one-to-one (1 : 1 : 1), one-to-one-to-many (1 : 1 : M) or many-to-one-to-one (M : 1 : 1), one-to-many-to-many (1 : M : N) or many-to-many-to-one (M : N : 1), and many-to-many-to-many (M : N : P).

- **One-to-one:** In one-to-one mapping, each instance of entity type E_1 is associated with at most one instance of entity type E_2 and vice-versa. It is represented by 1 : 1 relationship. For example, consider two entities PERSON and DRIVING_LICENSE. Each person can only have one driving license, and one driving license with a particular number can only be issued to a single person. That is, no two persons can have the same driving license. Thus, these two entities have 1 : 1 relationship [see Figure 2.8(a)]. A one-to-one relationship is generally implemented by combining the attributes of both the entity types into one entity type. In this example, the DRIVING_LICENCE being a weak entity type can be specified as one of the attributes of the entity type PERSON.
- **One-to-many:** In one-to-many mapping, each instance of entity type E_1 can be associated with zero or more (any number of) instances of type E_2. However, an instance of type E_2 can only be associated to at most one instance of E_1. It is represented by 1 : M relationship. For example, one publisher can publish many books; however, one book (with a particular ISBN) can only be published by one publisher. Thus, the entities PUBLISHER and BOOK have 1 : M relationship [see Figure 2.8(b)].
- **Many-to-one:** In many-to-one mapping, each instance of entity type E_1 can be associated with at most one instance of type E_2. However, an instance of type E_2 can be associated with zero or more (any number of) instances of type E_1. It is represented by M : 1 relationship. For example, different books can be published by one publisher; however, one book can only be published by one publisher. Thus, the entities BOOK and PUBLISHER have M : 1 relationship [see Figure 2.8(c)].

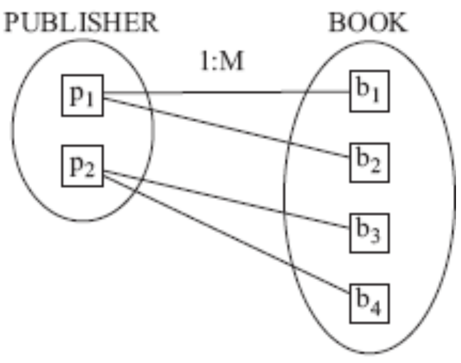
  NOTE A many-to-one relationship is same as one-to-many; however, from a different point of view.

- **Many-to-many:** In many-to-many mapping, each instance of entity type E_1 can be associated with zero or more (any number of) instances of type E_2 and an instance of type E_2 can also be associated with zero or more (any number of) instances of type E_1. It is represented by M : N relationship. For example, one author can write many books and one book can be written by more than one authors. Thus, the entities AUTHOR and BOOK have M : N relationship [see Figure 2.8(d)].

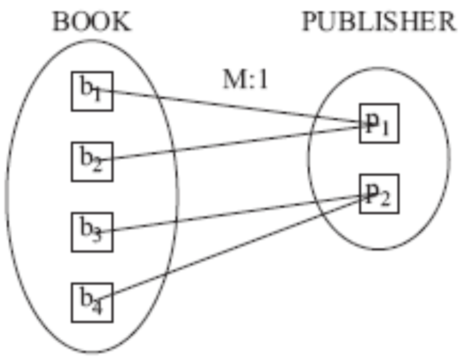
Participation Constraint The **participation constraint** specifies whether an entity instance can exist without participating in a relationship with another entity. In some notations, this constraint is also known as **minimum cardinality constraint**. The participation constraints can be of two types, namely, *total and partial.* If every instance of entity type E participates in at least one relationship instance of type R, the participation is said to be **total.** On the other hand, if only some of the instances of entity type E participate in the relationship, the participation is said to be **partial.**



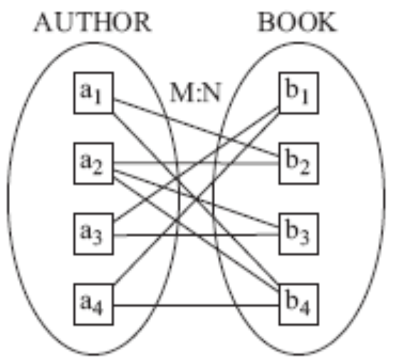
(a) One-to-one



(b) One-to-many



(c) Many-to-one



(d) Many-to-many

Fig. 2.8 *Mapping cardinalities*

The total participation constraint is also known as **mandatory participation**, which specifies that an entity instance cannot exist without participating in the relationship. It implies that the participation of each entity instance in a relationship type is mandatory. For example, the participation of each instance of the entity type `BOOK` in the relationship type `PUBLISHED_BY` is total, as each book must be published by a publisher. On the other hand, the partial participation constraint, also known as **optional participation**, specifies that an entity instance can exist without participating in the relationship. It implies that the participation of entity instance in a relationship type is optional. For example, the participation of the entity `PUBLISHER` in the relationship type `PUBLISHED_BY` is partial, as not all publishers publish books—some may publish only journals, magazines, and newspapers.

A weak entity type always has a total participation constraint, as each instance of weak entity type must be related to at least one of the instances of its parent entity type.

2.2 ENTITY-RELATIONSHIP DIAGRAM

Once the entity types, relationships types, and their corresponding attributes have been identified, the next step is to graphically represent these components using entity-relationship (E-R) diagram. An **E-R diagram** is a specialized graphical tool that demonstrates the interrelationships among various entities of a database. It is used to represent the overall logical structure of the database. While designing E-R diagrams, the emphasis is on the schema of the database and not on the instances. This is because the schema of the database is changed rarely; however, the instances in the entity and relationship sets change frequently. Thus, E-R diagrams are more useful in designing the database. An E-R diagram serves several purposes listed as follows:

Learn More

Entity-relationship models can be classified in two types, namely, *Binary Entity Relationship Model (BERM)* and *General Entity Relationship Model (GERM)*. In a BERM, only binary relationships are allowed; however, in GERM, relationships between three or more entities (‘*n*-ary’ relationships) are also allowed.

- It is used to communicate the logical structure of the database to the end users.
- It helps the database designer in understanding the information to be contained in the database.
- It serves as a documentation tool.

2.2.1 E-R Diagram Notations

There is no standard for representing the data objects in E-R diagrams. Each modeling methodology uses its own notation. The original notation used by Chen is the most commonly used notation. It consists of a set of symbols that are used to represent different types of information. The symbols include *rectangles, ellipses, diamonds, lines, double lines, double ellipses, dashed ellipses,* and *double rectangles.* All these symbols are described here and shown in Figure 2.9.

- The entity types such as `BOOK`, `PUBLISHER`, etc. are shown in rectangular boxes labelled with the name of the entity types. Weak entity types such as `EDITION` are shown in double rectangles.
- The relationship types such as `PUBLISHED_BY`, `REVIEWS`, `FEEDBACKS`, etc., are shown in diamond-shaped boxes labelled with the name of the relationship types. The identifying relationships such as `HAS` are shown in double diamonds.
- The attributes of the entity types such as `Book_title`, `Price`, `Year`, etc., are shown in ellipses. The key attributes are also shown in ellipses; however, with their names underlined. Multivalued attributes such as `Email_ID` and `Phone` are shown in double ellipses and derived attributes are shown in dashed ellipses.
- Straight lines are used to link attributes to entity types and entity

types to relationship types. Double lines indicate the total participation of an entity type in a relationship type.
- The cardinality ratios of each binary relationship type are represented by specifying a 1, M or N on each participating edge.

2.2.2 E-R Diagram Naming Conventions

Before initiating the designing of an E-R model, it is necessary to decide the format of the names to be given to the entity types, relationship types, attributes and roles, which will be used in the model. Some points must always be kept in mind while choosing the proper naming conventions.

- The names should be chosen in such a way that the meaning of the context is conveyed as much as possible.
- The usage of names should be consistent throughout the E-R diagram.
- The usage of hyphens, spaces, digits, and special characters should be avoided.
- The use of unpopular abbreviations and acronyms should be avoided.
- Singular names should be used for entity types.
- The binary relationship names should be chosen in such a way that makes the E-R diagram readable from left to right and from top to bottom. For example, consider the relationship type PUBLISHED_BY in Figure 2.12(a). This relationship type is read from left to right as 'the entity type BOOK is *published by* the entity type PUBLISHER'. However, if the position of the entities in the E-R diagram is changed, the relationship type PUBLISHED_BY becomes PUBLISHES and then it will be read as 'the entity type PUBLISHER *publishes* the entity type BOOK'.

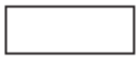
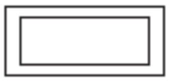
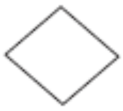
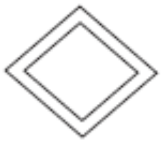
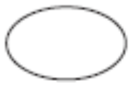
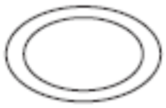
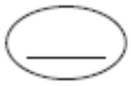
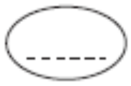
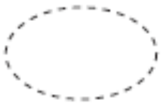
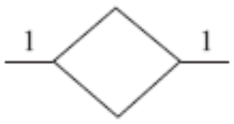
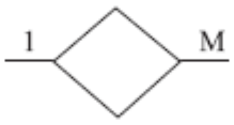
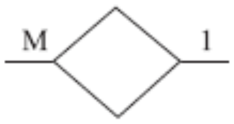
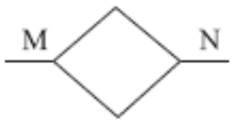
Notation	Purpose
	represents entity types
	represents weak entity types
	represents relationship type
	represents identifying relationship
	represents attributes
	represents multivalued attribute
	represents key attribute
	represents partial key of weak entity type
	represents derived attribute
—	connects attributes to entity types and entity types to relationship types
═	represents total participation
	represents 1:1 relationship
	represents 1:M relationship
	represents M:1 relationship
	represents M:N relationship

Fig. 2.9 *E-R Diagram notations*

- If a primary key of one entity type is used as a foreign key in another entity type, it should be used unchanged or with some extensions.

Learn More

There are many tools to construct E-R diagrams. Some of them are Avolution, ConceptDraw, ER/Studio, PowerDesigner, Rational Rose, SmartDraw, Microsoft Visio, and Visual Paradigm.

2.2.3 E-R Diagram for *Online Book* Database

As discussed in Chapter 01, the *Online Book* database maintains the details such as ISBN, title, price, year of publishing, number of pages, author and publisher of various textbooks, language books and novels. The user can also view the details of various authors and publishers. The author details include the name, address, URL, and phone number. The publisher details include the publisher name, address, e-mail ID, and phone number. The database also maintains the details of the ratings and feedbacks of other authors about a particular book. The ratings of a particular book are displayed along with the other details of the book. However, the detailed feedback about the book can be viewed on the URL of the authors who have reviewed the book.

Based on this specification, the E-R diagram for *Online Book* database can be designed. This section discusses the representation of all the entity types, relationship types, attributes, role names and recursive relationships, weak entity types, and identifying relationship, and cardinality ratios and participation constraints of *Online Book* database in an E-R diagram.

Representing Entity Types and Attributes

The *Online Book* database consists of three entity types, namely, `BOOK`, `PUBLISHER`, and `AUTHOR`. The entity type `BOOK` consists of the attributes `Book_title`, `Price`, `ISBN`, `Year`, `Page_count`, and `Category`. The entity type

PUBLISHER has the attributes P_ID, Name, Address, Phone, and Email_ID. The entity type AUTHOR has the attributes A_ID, Name, Address, URL, and Phone. The attributes ISBN, P_ID, and A_ID are the key attributes. All these entities with their corresponding attributes are shown in Figure 2.10

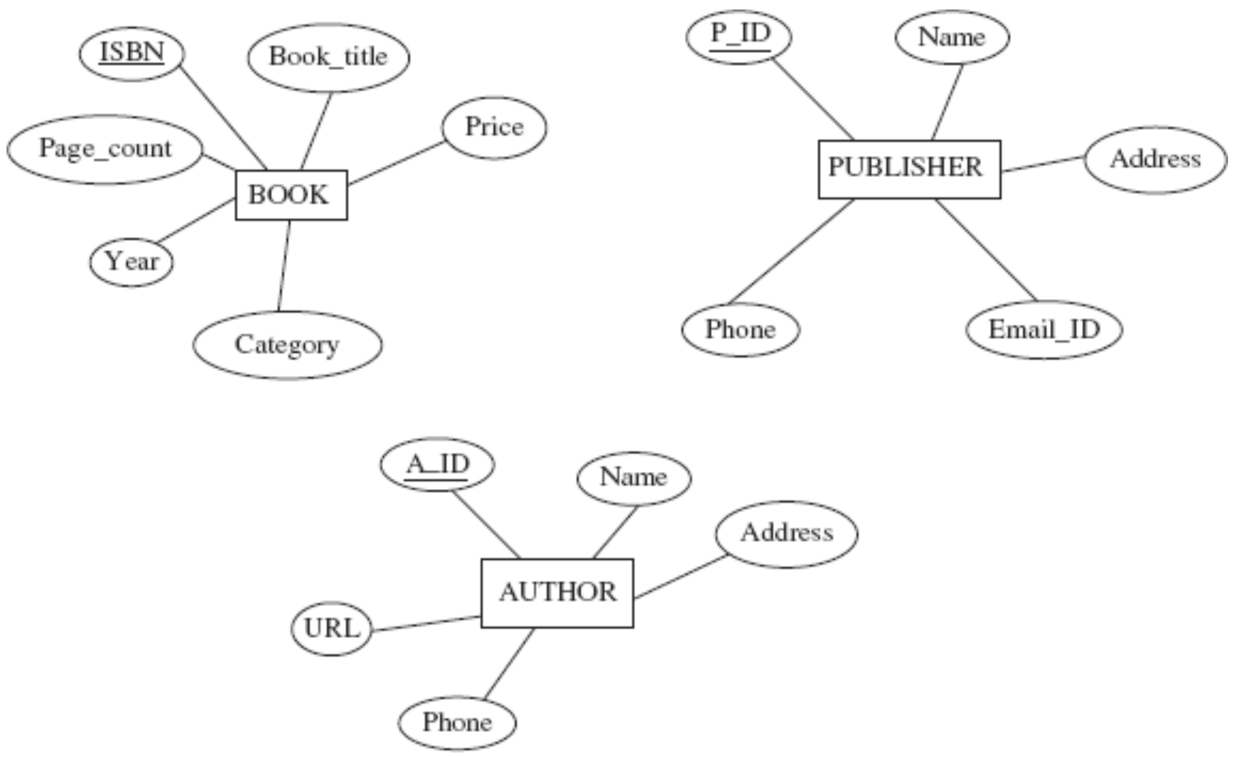


Fig. 2.10 *Entity types of Online Book database*

Representing Multivalued, Composite, and Derived Attributes

The attributes Phone and Email_ID are the multivalued attributes and Address is a composite attribute comprising House_number, Street, City, State and Zip_code as its component attributes. The Name of the author can also be taken as composite attribute with First_name, Middle_name, and Last_name as its component attributes. The composite attributes are shown by attaching ellipses containing the component attributes. Figure 2.11 shows the multivalued and composite attributes. It also shows some attributes in dashed ellipses. These attributes are the derived attributes. For example, the value of the attribute Books_published (for the PUBLISHER) can be calculated by counting the book entities associated with that publisher. Similarly, the attribute Books_written (for the AUTHOR) can be calculated by counting the book entities associated with that author.

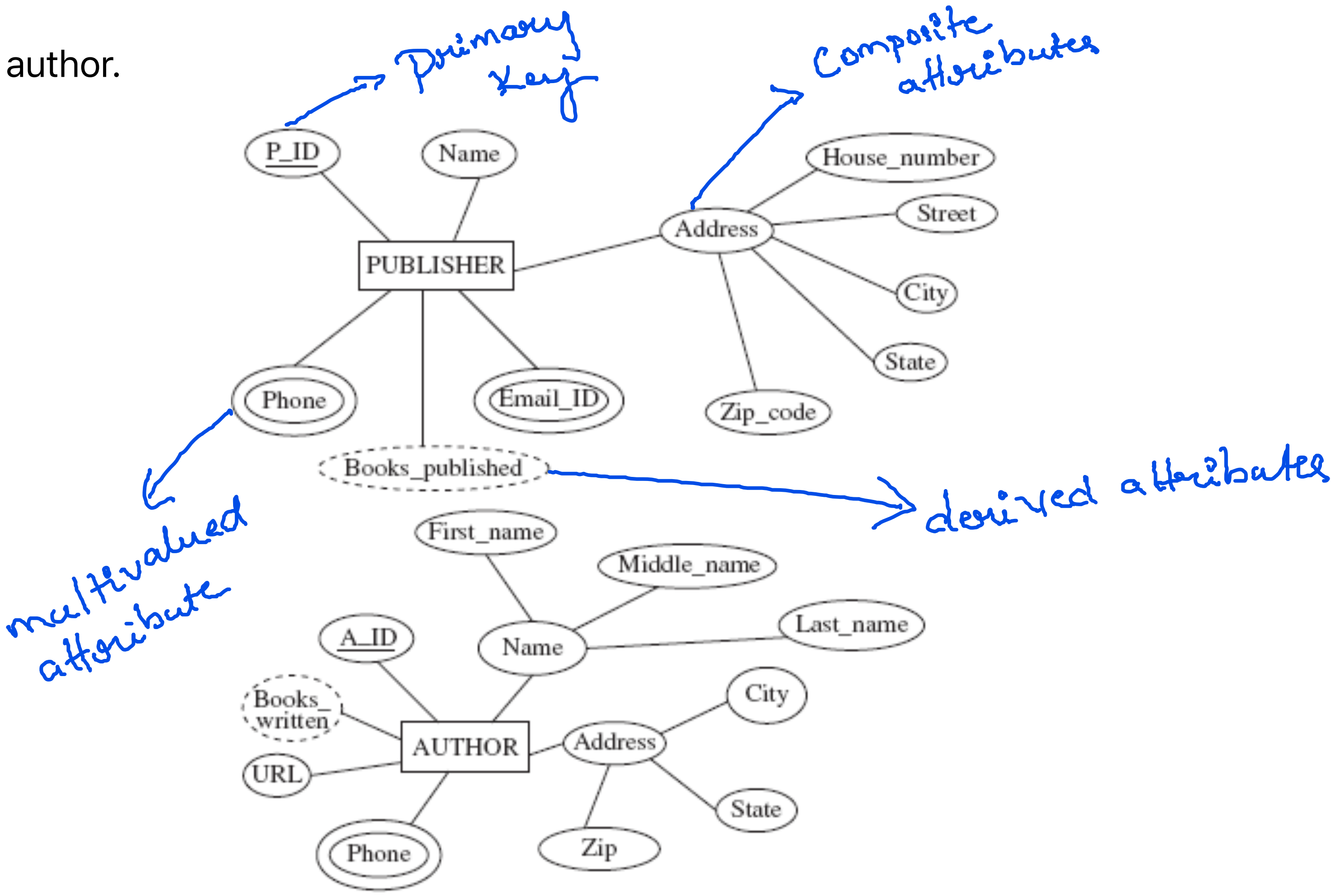


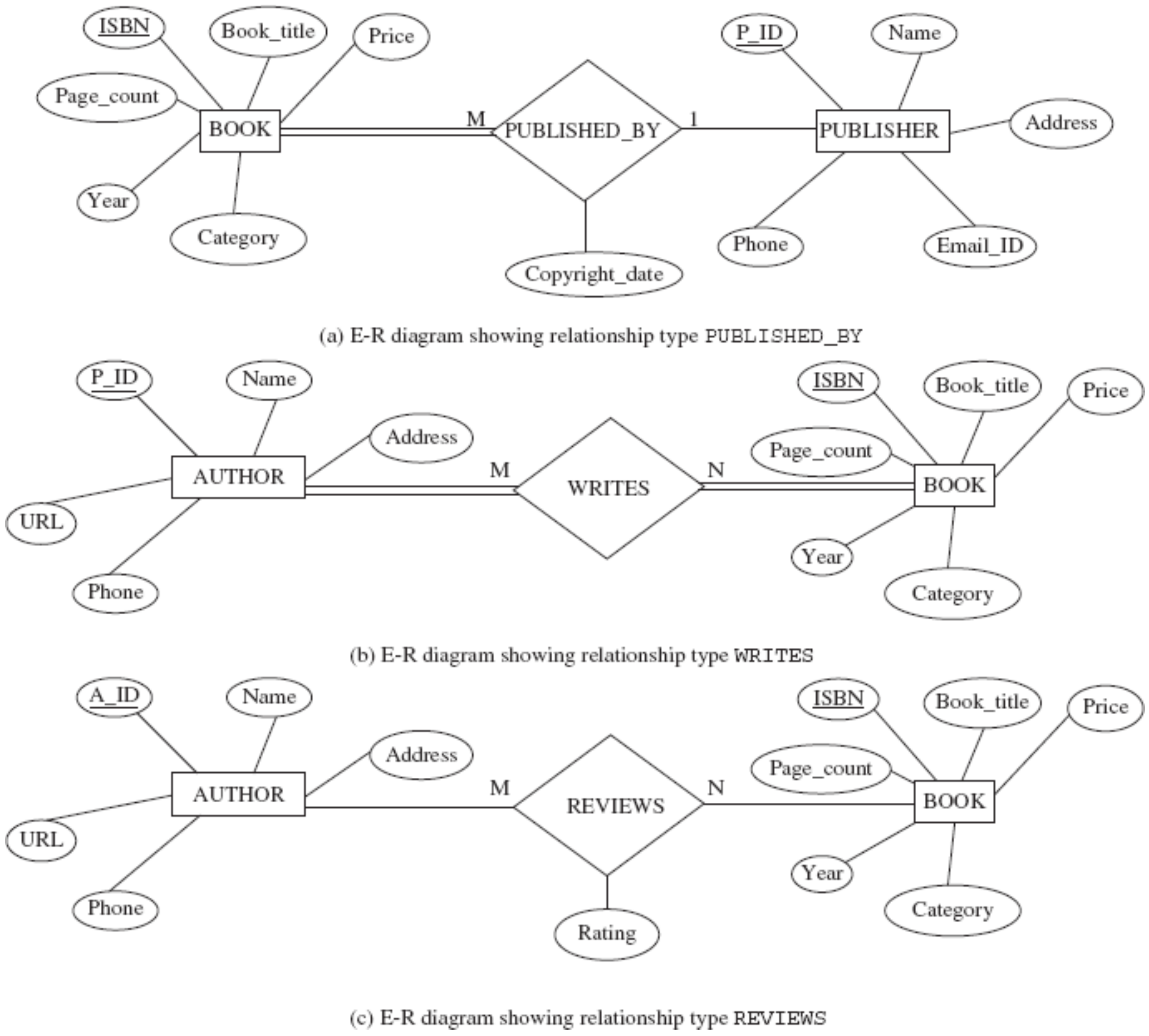
Fig. 2.11 *Multivalued, composite, and derived attributes*

Representing Relationship Types and Their Attributes

The entity types BOOK and PUBLISHER are associated with a relationship type PUBLISHED_BY by M : 1 relation. That is, many books can be published by one publisher; however, one book of a particular ISBN can be published by one publisher only. The entity types AUTHOR and BOOK are associated by the relation type WRITES by M : N relation. That is, one book can be written by many authors and one author can write many books. The entity types AUTHOR and BOOK are also associated with the relationship type REVIEWS by M : N relation, which specifies that one author can review many books and one book can be reviewed by many authors.

The participation (total or partial) of each entity in the relationship type can also be shown in an E-R diagram. For example, the participation of entity type BOOK in the relationship type PUBLISHED_BY is total, as every book must be published by a publisher. Similarly, the participation of entity types `BOOK` and `AUTHOR` in the relationship type `WRITES` is also total, as every book must be written by at least one author and each author must write at least one book. However, the participation of the entity types `BOOK` and `AUTHOR` in the relationship type `REVIEWS` is partial, as some books may not be reviewed by any author and some authors may not review any book. The E-R diagrams showing the relationship type `PUBLISHED_BY`, `WRITES` and `REVIEWS`, the participation of each entity in the relationship types, and their descriptive attributes are shown in Figure 2.12.

If a relationship type has some descriptive attributes, then these attributes can also be shown in an E-R diagram. Figure 2.12 shows the relationship types `PUBLISHED_BY` and `REVIEWS` with the descriptive attributes `Copyright_date` and `Rating`, respectively. The attribute `Copyright_date` specifies the date on which the publisher gets the right of publishing the book for the first time. The attribute `Rating` contains a rated value (between 1 and 10) given to the book by other authors after reviewing the book.



(a) E-R diagram showing relationship type `PUBLISHED_BY`

(b) E-R diagram showing relationship type `WRITES`

(c) E-R diagram showing relationship type `REVIEWS`

Fig. 2.12 *E-R diagrams showing relationship between* `BOOK`, `AUTHOR`, *and* `PUBLISHER`

Representing Role Names and Recursive Relationships

An E-R diagram can also be used to represent the role names and recursive relationships. For example, the author of one book acts as a reviewer for other books. Thus, a recursive relationship exists between the instances of the same entity type `AUTHOR`. This recursive relationship with the role names is shown in Figure 2.13. The role names (*reviewer* and *author*) are indicated by labelling the lines connecting the relationship types to entity types. Since, the relationship type `FEEDBACKS` involves only one entity type, it is a unary relationship.

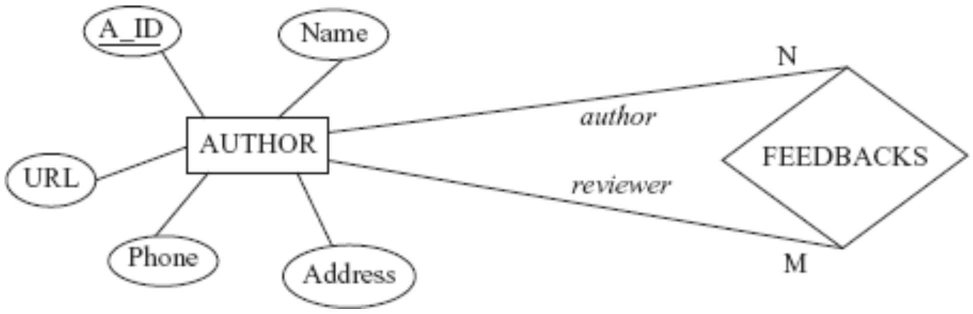


Fig. 2.13 *E-R diagram showing role names and recursive relationship*

Representing Weak Entity Types and Identifying Relationships

In the *Online Book* database, the entity type EDITION is a weak entity type with Edition_no as its partial key. It depends on the strong entity type BOOK. It has a total participation in the identifying relationship type HAS, which specifies that an edition cannot exist without a book (see Figure 2.14).

The complete E-R diagram for the *Online Book* database is shown in Figure 2.15.

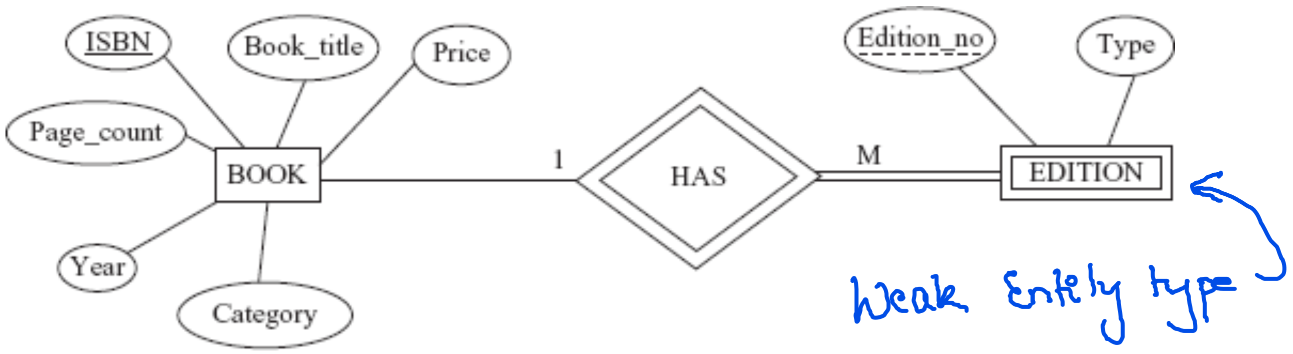


Fig. 2.14 *E-R diagram showing weak entity, identifying relationship, and partial key*

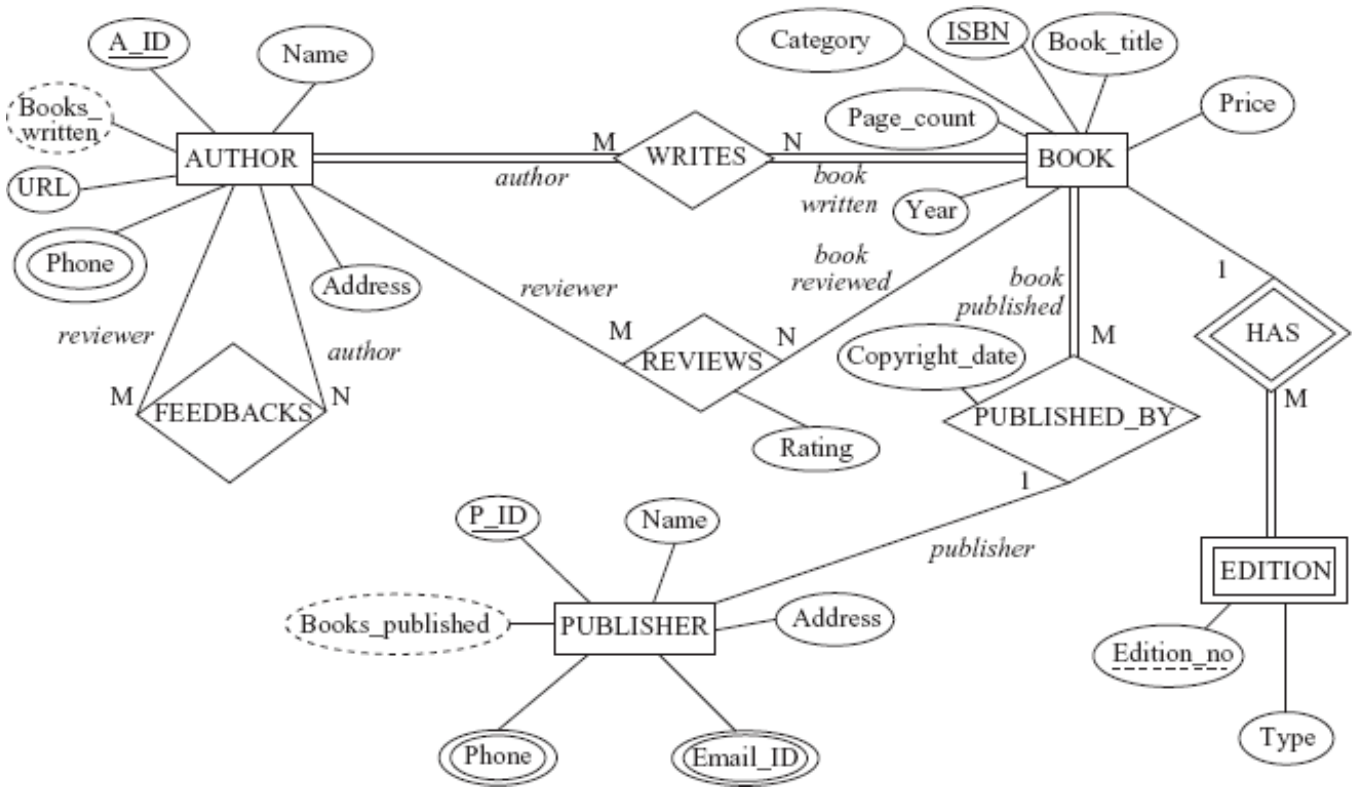


Fig. 2.15 *E-R diagram of Online Book database*

2.2.4 E-R Diagram Representing Ternary Relationships

The E-R diagram shown in Figure 2.15 includes only unary and binary relationships. An E-R diagram can also be used to represent higher degree relationships (relationships of degree greater than two). The most common higher degree relationship is ternary relationship, which describes the relationship between three entity types. For example, consider the entity types AUTHOR, BOOK, and SUBJECT. The entity type SUBJECT has the attributes S_ID and Subject_name. The subject name could be Database Systems, C++, C, Computer Architecture, etc.

Each author writes a book on a particular subject. This relationship is ternary as it associates three entity types. Figure 2.16 shows the relationship type WRITES among three entity types AUTHOR, BOOK, and SUBJECT. The mapping cardinality of the relationship type WRITES is M : 1 : 1, which implies that a group of particular authors (many authors) can write at most one book on a particular subject.

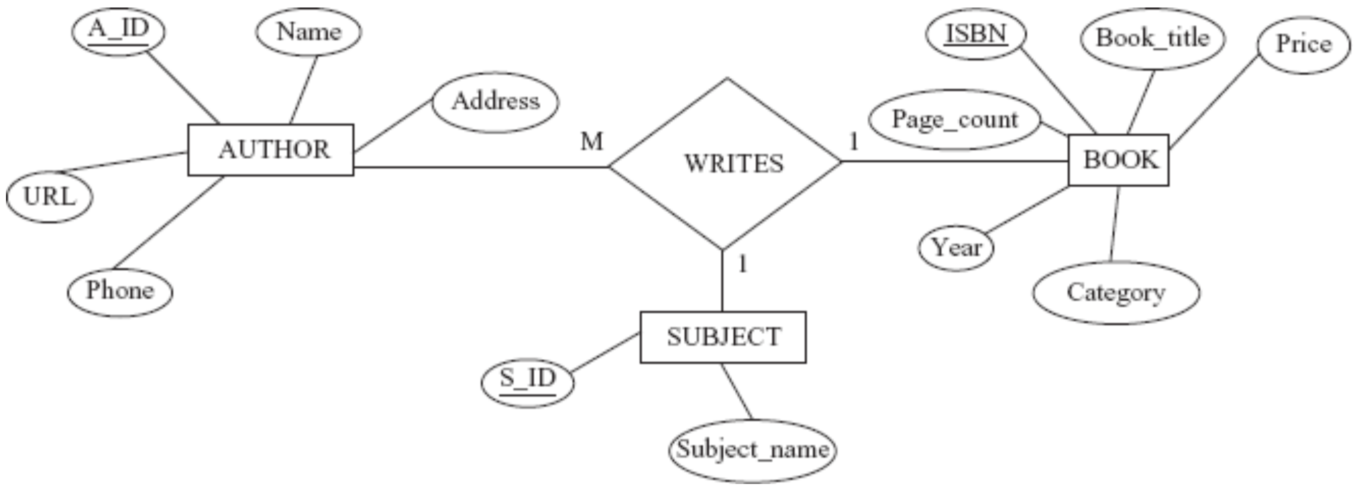


Fig. 2.16 *E-R Diagram showing a ternary relationship*

NOTE Relationships beyond ternary relationships are not very common; however, can sometimes exists. The discussion of the relationships greater than the degree three is beyond the scope of this book.

2.2.5 E-R Diagram—Design Choices

While designing an E-R diagram, a number of design choices are to be considered. In other words, it is possible to represent a set of entities and their relationships in several ways. This section discusses each of these choices.

Entity Types versus Attributes

When the attributes of a particular entity type are identified, it becomes difficult to determine whether a property of the entity should be represented as an entity type or as an attribute. For example, consider the entity type `AUTHOR` with attributes `A_ID`, `Name`, `Address`, `Phone`, and `URL`. Treating phone as an attribute implies that each author can have at the most one telephone number. In case multiple phone numbers have to be stored for each author, the attribute `Phone` can be taken as a multivalued attribute. However, treating telephone number as a separate entity type better models a situation where the user may want to store some extra information such as location (*home, office* or *mobile*) of the telephone. Thus, `Phone` can be taken a separate entity type with attributes `Phone_number` and `Location`, which implies that an author can have multiple phone numbers at different locations.

After making Phone as an entity type, the E-R diagram for the entity type AUTHOR can be redesigned as shown in Figure 2.17. In this figure, the entity types AUTHOR and PHONE are related with the relationship type HAS_PHONE with the cardinality ratio 1 : M. It implies that one author can have many phone numbers; however, one phone number can only be associated with one author.

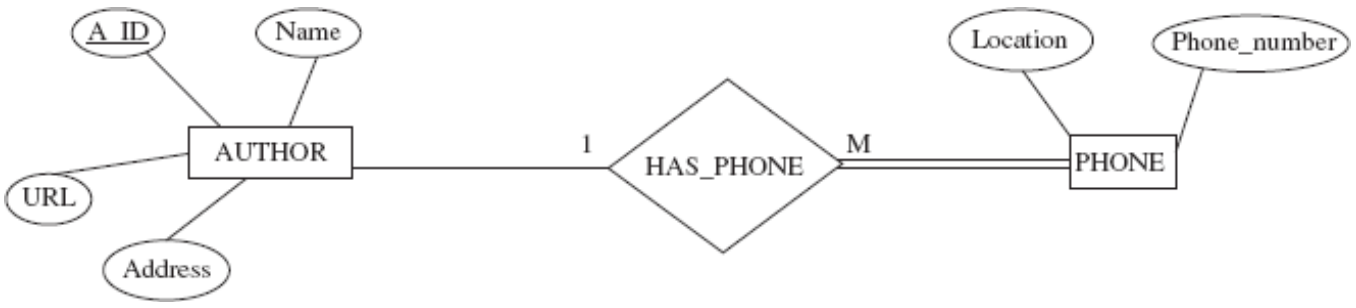


Fig. 2.17 *E-R Diagram showing PHONE as an entity type*

Binary versus Ternary Relationship

Generally, a database contains binary relationships, as it is easy to represent and understand the binary relationships. However, in some cases, ternary relationships need to be created to represent certain situations. For example, consider the ternary relationship WRITES in Figure 2.16. Its equivalent E-R diagram comprising three distinct binary relationships, WRITES_ON, WRITES and OF with cardinality ratios M : N, M : N, and M : 1, respectively, is shown in Figure 2.18. By comparing Figures 2.16 and 2.18, it appears that the relationships between three entity types have changed from a single M : 1 : 1 to three binary relationships—two of them with M : N mapping cardinality and one with M : 1 mapping cardinality.

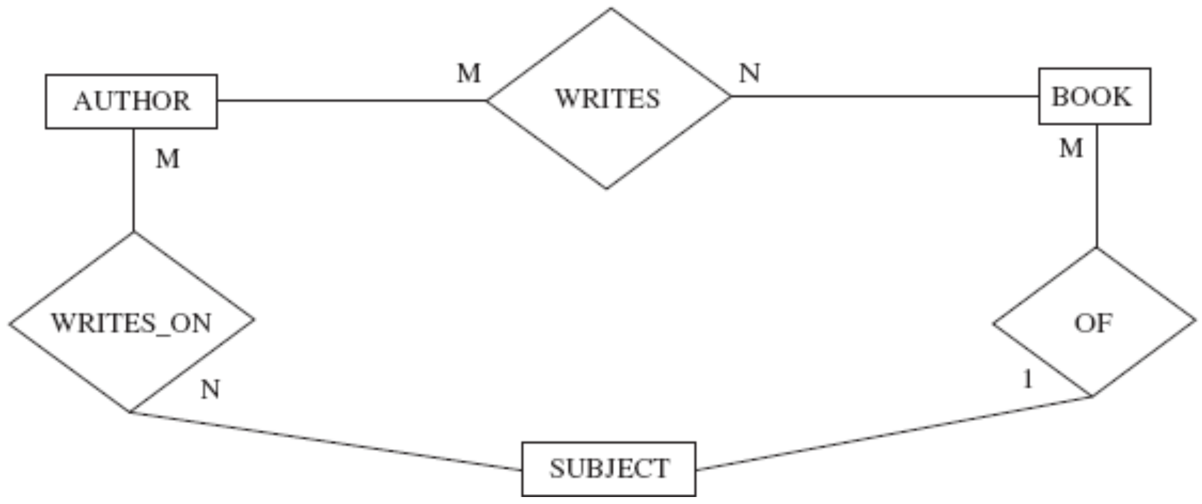


Fig. 2.18 *E-R Diagram showing three binary relationships*

The E-R diagram shown in Figure 2.16 conveys the meaning that a group of particular authors can write at most one book on a particular subject. That is, same group of authors cannot write many books on the same subject. On the other hand, the E-R diagram shown in Figure 2.18 conveys three different relationships. The relationship WRITES shows that one author can write many books and one book can be written by many authors. The relationship WRITES_ON shows that one author can write books on many subjects and books on one subject can be written by many authors. Finally, the relationship OF shows that many books can be written on a particular subject; however, one book can belong to only one subject. The information conveyed by these three binary relationships is not same as that of the ternary relationship shown in Figure 2.16. Thus, it is the responsibility of the database designer to make an appropriate decision depending on the situation being represented.

2.3 ENHANCED E-R MODEL

The E-R diagrams discussed so far represents the basic concepts of a database schema. However, some aspects of a database such as inheritance among various entity types cannot be expressed using the basic E-R model. These aspects can be expressed by enhancing the E-R model. The resulting diagrams are known as **enhanced E-R** or **EER diagrams** and the model is called **EER model**.

The basic E-R model can represent the traditional database applications such as typical data processing application of an organization effectively. On the other hand, the EER model is used to represent the new and complex database applications such as telecommunications, Geographical Information Systems (GIS), etc. This section discusses the extended E-R features including *specialization, generalization,* and *aggregation* and their representation using EER diagrams.

2.3.1 Specialization and Generalization

In some situations, an entity type may include sub-groupings of its entities in such a way that entities of one subgroup are distinct in some way from the entities of other subgroups. For example, the entity type `BOOK` can be classified further into three types, namely, `TEXTBOOK`, `LANGUAGE_BOOK`, and `NOVEL`. These entity types are described by a set of attributes that includes all the attributes of the entity type `BOOK` and some additional set of attributes that differentiate them from each other. These additional attributes are also known as **local** or **specific** attributes. For example, the entity type `TEXTBOOK` may have the additional attribute `Subject` (for example, Computer, Maths, Science, etc.), `LANGUAGE_BOOK` may have the attribute `Language` (for example, French, German, Japanese, etc.), and the entity type `NOVEL` may have the attribute `Type` (Fiction, Mystery, Fantasy, etc.). This process of defining the subgroups of a given entity type is called **specialization**.

The entity type containing the common attributes is known as the **superclass** and the entity type, which is a subset of the superclass, is known as its **subclass**. For example, the entity type `BOOK` is a superclass and the entity types `TEXTBOOK`, `LANGUAGE_BOOK`, and `NOVEL` are its subclasses. This process of refining the higher-level entity types (superclass) into lower-level entity types (subclass) by adding some additional features to each of them is a **top-down** design approach.

The design process may also follow a **bottom-up** approach in which multiple lower-level entity types are combined on the basis of common features to form higher-level entity types. For example, the database designer may first identify the entity types TEXTBOOK, LANGUAGE_BOOK, and NOVEL and then combine the common attributes of these entity types to form a higher-level entity type BOOK. This process is known as **generalization**. bottom up approach top down approach

In simple terms, generalization is the reverse of specialization.

The two approaches are different in terms of their starting and ending point. Specialization starts with a single higher-level entity type and ends with a set of lower-level entity types having some additional attributes that distinguish them from each other. Generalization, on the other hand, starts with the identification of a number of lower-level entity types and ends with the grouping of the common attributes to form a single higher-level entity type. Generalization represents the similarities among lower-level entity types; however, suppresses their differences.

Specialization and generalization can be represented graphically with the help of an EER diagram in which the superclass is connected with a line to a circle, which in turn is connected by a line to each subclass that has been defined (see Figure 2.19). The '∪' shaped symbol on each line connecting a subclass to the circle indicates that the subclass is a subset '⊂' of the superclass. The circle can be ' '(empty) or it can contain a symbol **d** (for disjointness) or **o** (for overlapping), which is explained later in this section.

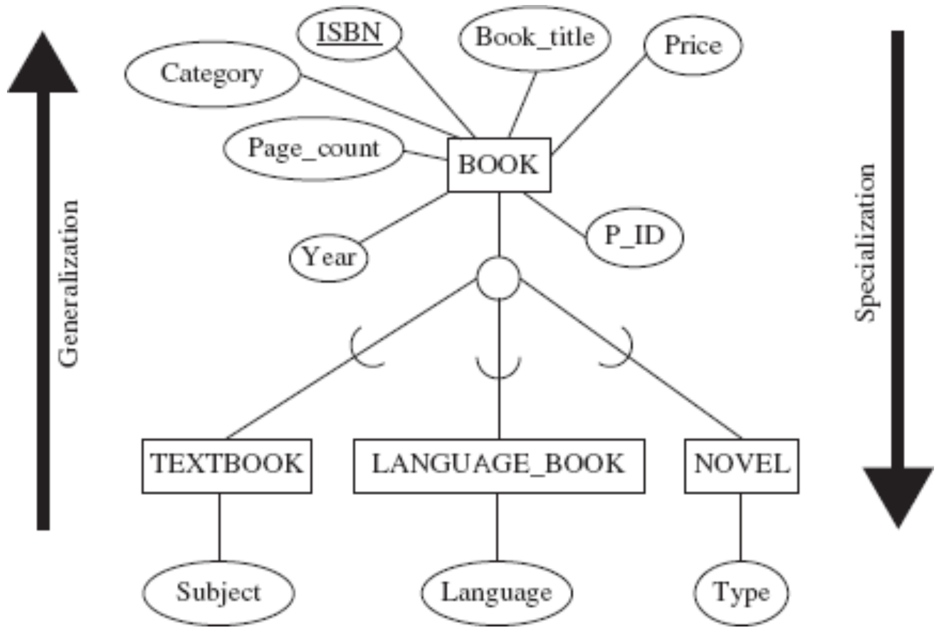


Fig. 2.19 *Specialization and generalization*

Attribute Inheritance

As discussed earlier, the higher-level and lower-level entity types are created on the basis of their attributes. The higher-level entity type (or superclass) has the attributes that are common to all of its lower-level entity types (or subclasses). These common attributes of the superclass are inherited by all its subclasses. This property is known as **attribute inheritance**. For example, the attributes ISBN, Book_title, Category, Price, Page_count, Year, and P_ID of the entity type BOOK are inherited by the entity types TEXTBOOK, LANGUAGE_BOOK, and NOVEL. This inheritance of entity types represents a hierarchy of classes.

Note that all the lower-level entity types of a particular higher-level entity types also participate in all the relationship types in which the higher-level entity type participates. For example, the entity type BOOK participates in the relationship type PUBLISHED_BY, its subclasses TEXTBOOK, LANGUAGE_BOOK, and NOVEL also participate in this relationship type. That is, all textbooks, language books, and novels are published by a publisher.

All the attributes and relationship types of a higher-level entity type are also applied to its lower-level entity types but not vice-versa.

If a subclass is a subset of only one superclass, that is, it has only one parent, it is known as **single inheritance** and the resulting structure is known as a **specialization** (or **generalization**) **hierarchy**. For example, Figure 2.19 shows a single inheritance since the three subclasses are derived from only one superclass. On the other hand, if a subclass is derived from more than one superclass, it is known as **multiple inheritance** and the resulting structure is known as a **specialization** (or **generalization**) **lattice**. In this case, the subclass is known as a **shared subclass**. For example, consider two entity types EMPLOYEE and STUDENT. A student who is working as a part-time employee in an organization as well as pursuing a course from a university can be derived from these two entity types. Figure 2.20 shows a specialization lattice with a shared subclass PARTTIME_EMPLOYEE.

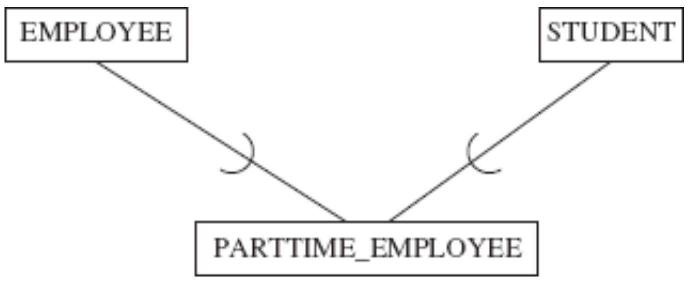


Fig. 2.20 *Specialization lattice with shared subclass PARTTIME_EMPLOYEE*

Constraints on Specialization and Generalization

To design an accurate and effective model for a particular organization, some constraints can be placed on specialization or generalization. All the constraints can be applied to both generalization and specialization. However, in this section the constraints are discussed in terms of specialization only.

Constraints on Subclass Membership In general, a single higher-level entity type (superclass) may have several lower-level entity types (subclasses). Thus, it is necessary to determine which entities of the superclass can be the members of a given subclass. The membership of the higher-level entities in the lower-level entity types can be either condition-defined or user-defined.

- **Condition-defined:** In condition-defined, the membership of entities is determined by placing a condition on the value of some attribute of the superclass. The subclasses formed on the basis of the specified condition are called **condition-defined** (or **predicate-defined**) **subclasses**. For example, all the instances of type BOOK can be evaluated based on the value of the attribute `Category`. The entities that satisfy the condition `Category = "Textbook"` are allowed to belong to the subclass TEXTBOOK only. Similarly, the entities that satisfy the condition `Category = "Language Book"` are allowed to belong to the subclass LANGUAGE_BOOK only. Lastly, the entities that satisfy the condition `Category = "Novel"` are allowed to belong to the `NOVEL` subclass only. Thus, these conditions are the constraints that determine which entities of the superclass can belong to a given subclass.

If the membership condition for all subclasses in specialization is defined on the same attribute (in this case, `Category`), it is known as an **attribute-defined specialization**, and the attribute is called the **defining attribute** of the specialization. The condition-defined subclass is shown by writing the condition next to the line connecting the subclass to the specialization circle (see Figure 2.21). The symbol **d** denotes the disjointness constraint.

- **User-defined:** If the membership of the superclass entities in a given subclass is determined by the database users and not by any membership condition, it is called **user-defined specialization,** and the subclasses that are formed are known as **user-defined subclasses**. For example, consider an entity `CELEBRITY`. A celebrity could be a film star, a politician, a newsreader or a player. Thus, the assignment of the entities of type `CELEBRITY` to a given subclass is specified explicitly by the database users when they apply the operation to add an entity to the subclass (see Figure 2.22). The symbol **o** denotes the overlapping constraint.

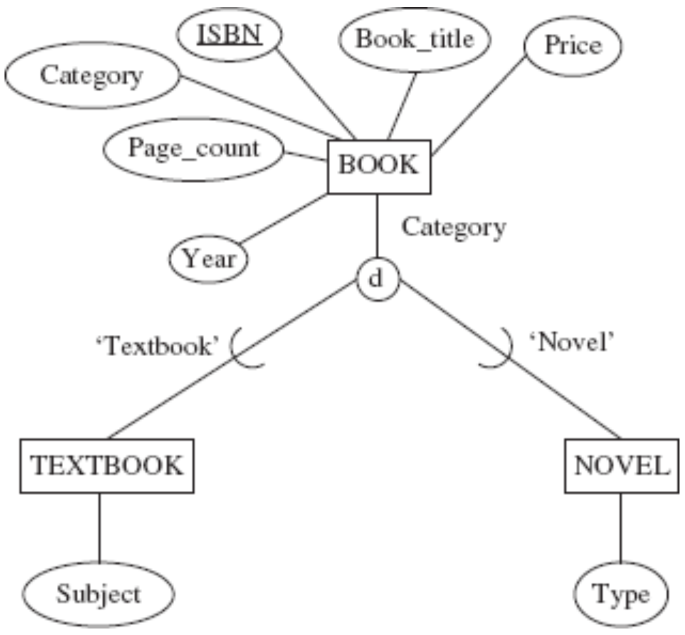


Fig. 2.21 *EER diagram for an attribute-defined specialization with disjoint constraint*

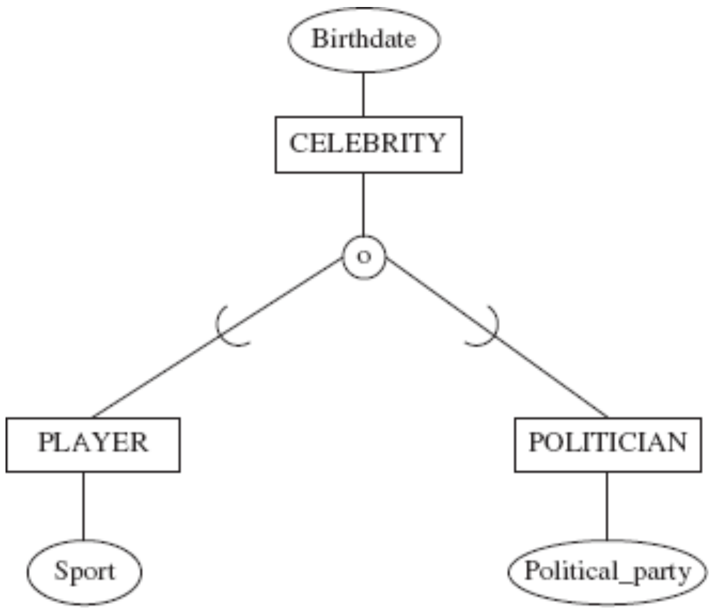


Fig. 2.22 *EER diagram for a user-defined specialization with overlapping constraint*

Disjoint and Overlapping Constraints In addition to subclass membership constraints, two other types of constraints, namely, *disjointness* and *overlapping,* can also be applied to a specialization. These constraints determine whether a higher-level entity instance can belong to more than one lower-level entity type within a single specialization or not.

- **Disjointness constraint:** This constraint specifies that the same higher-level entity instance cannot belong to more than one lower-level entity types. That is, the subclasses of any superclass must be disjoint. For example, an entity of type BOOK can belong to either TEXTBOOK or NOVEL but not both. An attribute-defined specialization in which the defining attribute is single-valued implies a disjointness constraint. The disjoint constraint is represented by a symbol **d** written in a circle in an EER diagram as shown in Figure 2.21.
- **Overlapping constraint:** This constraint specifies that the same higher-level entity instance can belong to more than one lower-level entity types. That is, the subclasses of any superclass need not to be disjointed and entities may overlap. In terms of EER diagram, the overlapping constraint is represented by a symbol **o** written in a

circle joining the superclass with its subclasses. For example, the entity types PLAYER and POLITICIAN show an overlapping constraint, as the celebrity can be a player as well as a politician (see Figure 2.22). Similarly, an entity of type BOOK can belong to both TEXTBOOK and LANGUAGE_BOOK since a language book can also be a prescribed textbook in a university [see Figure 2.23(b)].

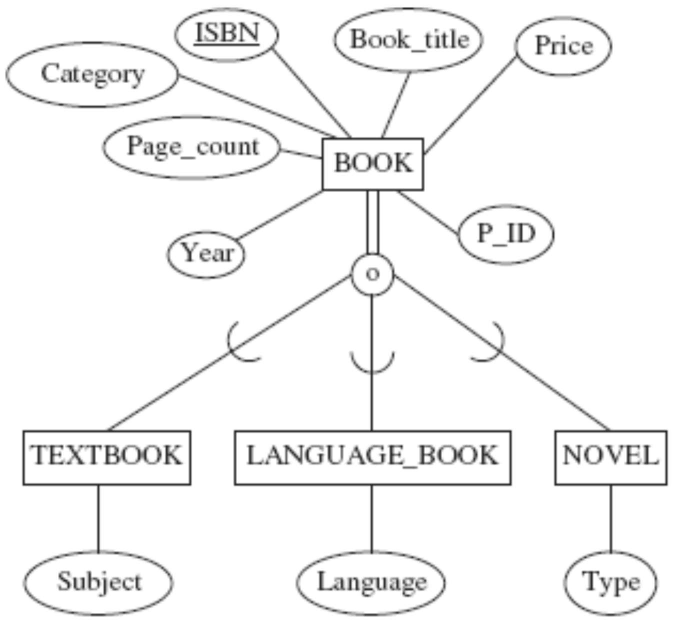
NOTE The lower-level entity types are by default overlapped; however, the disjointness constraint must be explicitly placed on generalization and specialization.

Completeness Constraint The last constraint that can be applied to generalization or specialization is the completeness constraint. It determines whether an entity in higher-level entity set must belong to at least one of the lower-level entity sets or not. The completeness constraint can be total or partial.

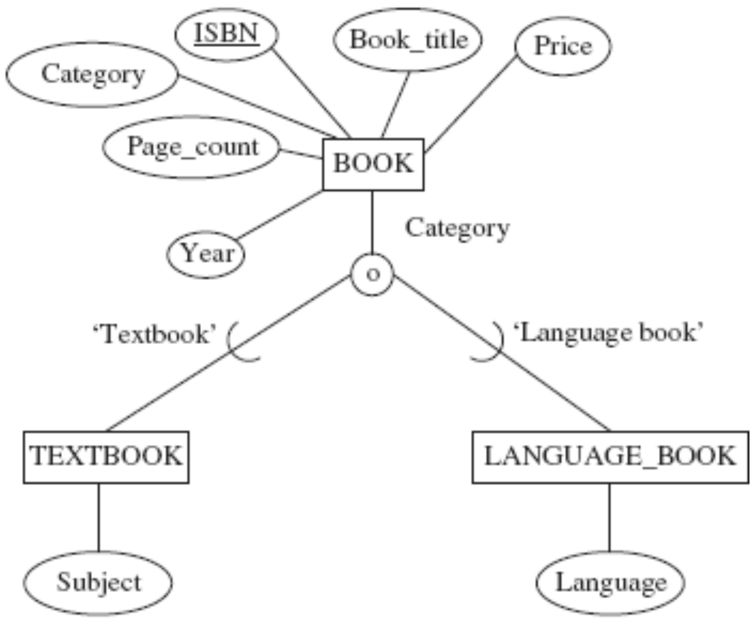
- **Total specialization:** It specifies that each higher-level entity must belong to at least one of the lower-level entity types in the specialization. Figure 2.23(a) shows the total specialization of the entity type BOOK. Here, each book entity must belong to either textbook or language book or novel category. The total specialization is represented by double lines connecting the superclass with the circle.
- **Partial Specialization:** It allows some of the instances of higher-level entity type not to belong to any of the lower-level entity types. Figure 2.23(b) shows the partial specialization of the entity type BOOK, as all books do not necessarily belong to the category textbooks or language books, some may belong to the category novels also.

2.3.2 Aggregation

The E-R diagrams discussed so far represent the relationships between two or more entities. An E-R diagram cannot represent the relationships among relationships. However, in some situation, it is necessary to represent a relationship among relationships. The best way to represent these types of situations is aggregation. The process through which one can treat the relationships as higher-level entities is known as **aggregation**. For example, in the *Online Book* database, the relationship `WRITES` between the entity types `AUTHOR` and `BOOK` can be treated as a single higher-level entity called `WRITES`. The relationship `WRITES` and the entity types `AUTHOR` and `BOOK` are aggregated into a single entity type to show the fact that once the author has written a book then only it gets published. The relationship type `PUBLISHED_BY` can be shown between the entity types `PUBLISHER` and `WRITES` as shown in Figure 2.24. The relationship type `PUBLISHED_BY` is a many-to-one relationship. It implies that a book written by a group of authors can be published by only one publisher; however, one publisher can print many books written by different authors.



(a) EER diagram showing total participation



(b) EER diagram showing partial participation

Fig. 2.23 *EER diagrams showing completeness constraints*

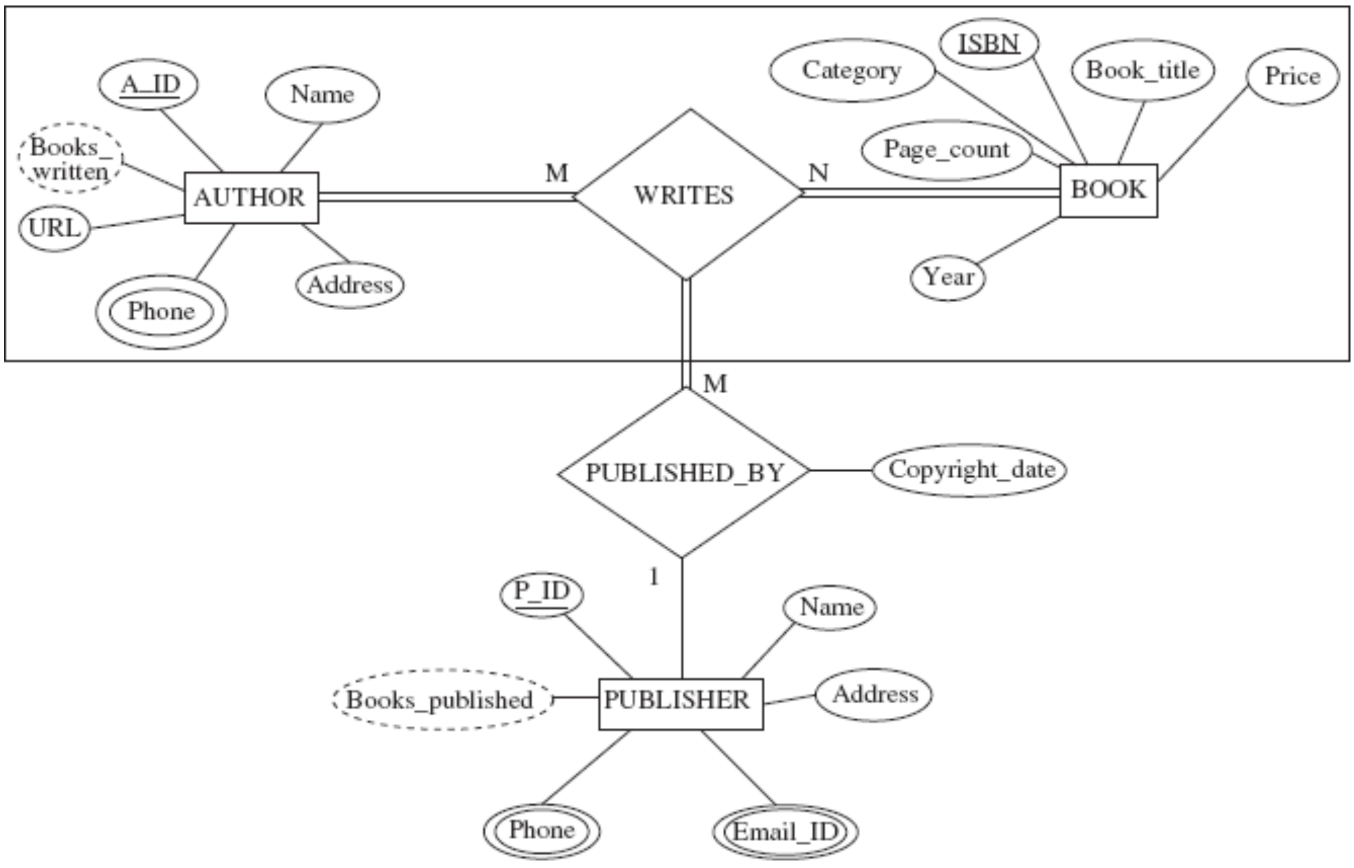


Fig. 2.24 *EER diagram for Online Book database showing aggregation*

Note that the EER diagram shown in Figure 2.24 cannot be expressed as ternary relationship between three entity types AUTHOR, BOOK, and PUBLISHER, as WRITES and PUBLISHED_BY are two different relationship types and hence, cannot be combined into one.

2.4 ALTERNATIVE NOTATIONS FOR E-R DIAGRAMS

An E-R diagram can also be represented using some other diagrammatic notations. One alternative E-R notation allows specifying the cardinality of the relationship, that is, the number of entities that can participate in a relationship. In this notation, a pair of integers (*min, max*) is associated with each participation of an entity type E in a relationship type R, where $0 \le min \le max$ and $max \ge 1$. This implies that each entity instance e of type E must participate in at least *min* and at most *max* relationship instances in R. If *min* is specified as zero, it means partial participation and if $min > 0$, it means total participation.

Figure 2.25 shows the modified E-R diagram of *Online Book* database with (*min, max*) notation. For example, the cardinality (1, N) of the AUTHOR implies that an author must write at least one book; however, an author can write any number of books. Similarly, the cardinality (1, 4) for the BOOK implies that a book must be written by at least one author and maximum by four authors.

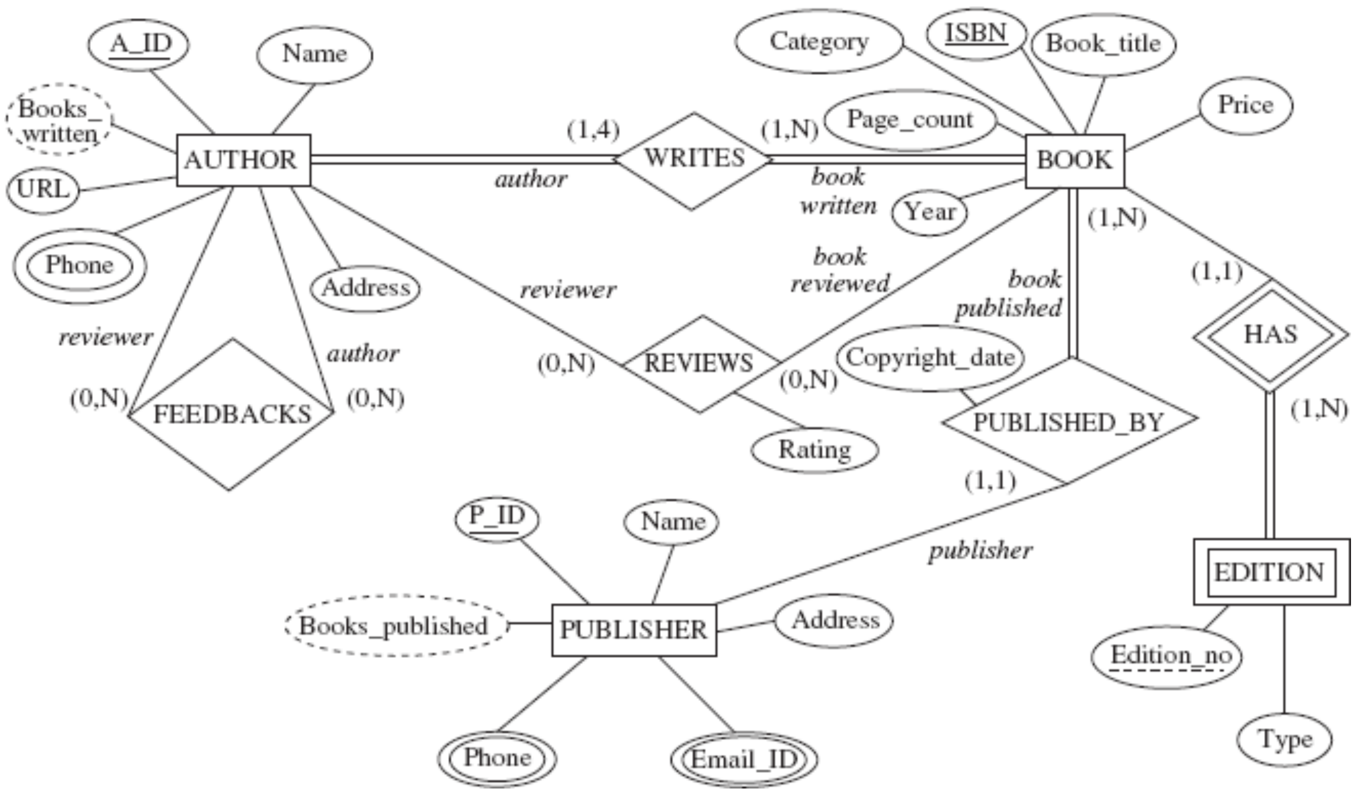


Fig. 2.25 *E-R diagram for Online Book database with (min, max) notation*

Another notation is **crow's feet** notation in which the symbol used to represent the many side of the relationship resembles the forward digits of a bird's claw. Figure 2.26 shows the modified E-R diagram of *Online Book* database with crow's feet notation. It actually consists of three symbols, namely, *empty circle 'O', vertical bar '|'* and *crow's feet '<'*. These symbols can be used in four combinations listed as follows:

- < + O = optional many (either zero, one or more)
- | + < = mandatory many (at least one or many)

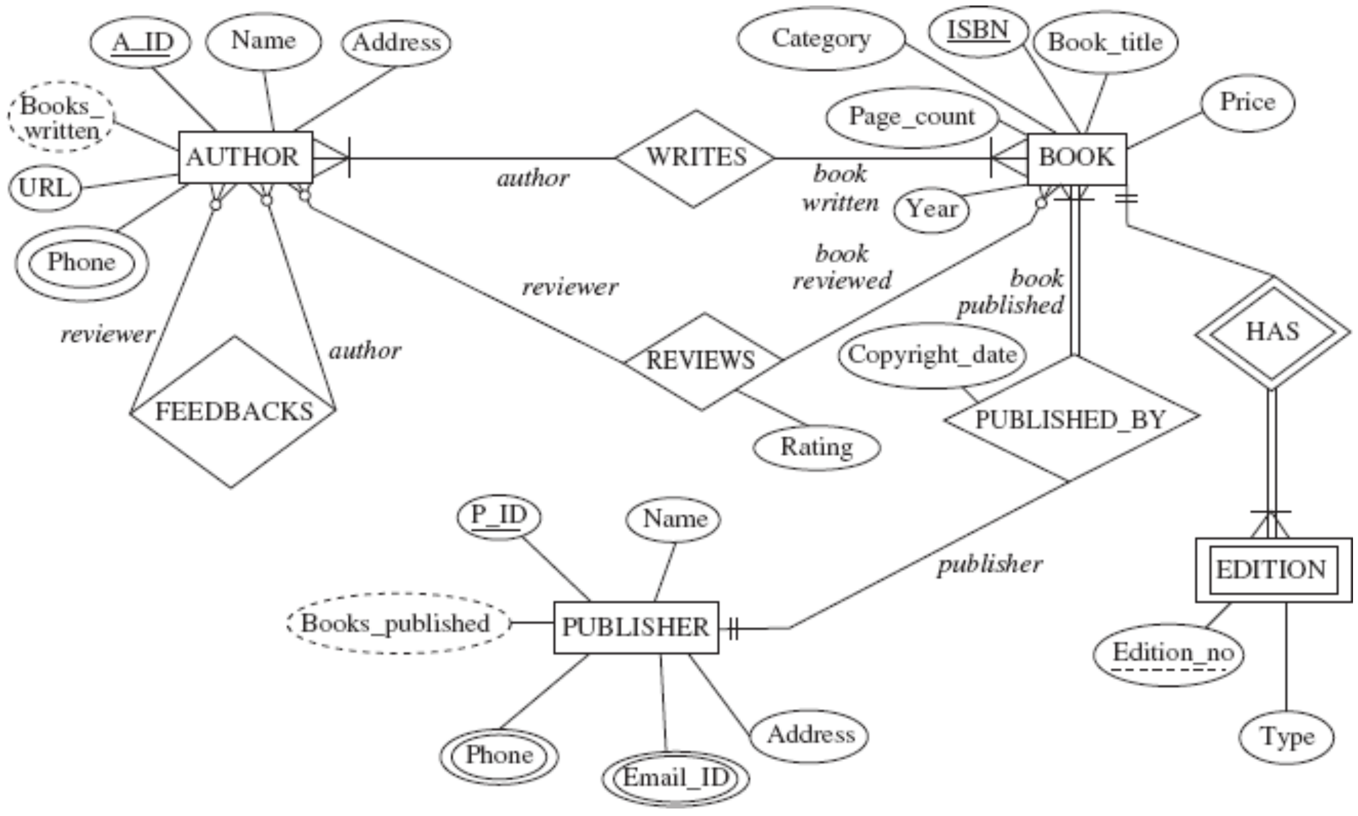


Fig. 2.26 *E-R diagram Online Book database with crow's feet notation*

- | + | = mandatory one (one and only one)
- | + O = optional one (either zero or one but not many)

2.5 UNIFIED MODELING LANGUAGE

E-R diagrams only represent various entities of an organization and relationships among them. The entities and their relationships form only one part of the overall software design. A software design includes some other components also such as functional modules of the system and interactions among them, user interactions with the system, etc. These components can be specified using a standard called the **Unified Modeling Language (UML)**, which was officially defined by the Object Management Group (OMG).

Learn More

Rational Rose is currently a popular tool used to draw UML diagrams. It helps the software developers in designing clear and easy-to-understand UML models.

Unified Modeling Language (UML) is used as one of the techniques to implement object data modeling. The main advantage of object data modeling is that it models the elements of the system as real-world objects and hence, represents the real world very closely. Initially, it was designed for modeling object-oriented software; however, nowadays it is also used for business process modeling, systems engineering modeling, and representing organizational structures.

2.5.1 UML Diagrams

UML includes a number of diagrams that are used to create an abstract model of a system. This abstract model is usually known as a **UML model**. UML includes ten standard diagrams, which are divided into two main categories, namely, *structural diagrams* and *behavioural diagrams*. Figure 2.27 shows the hierarchy of these UML diagrams.

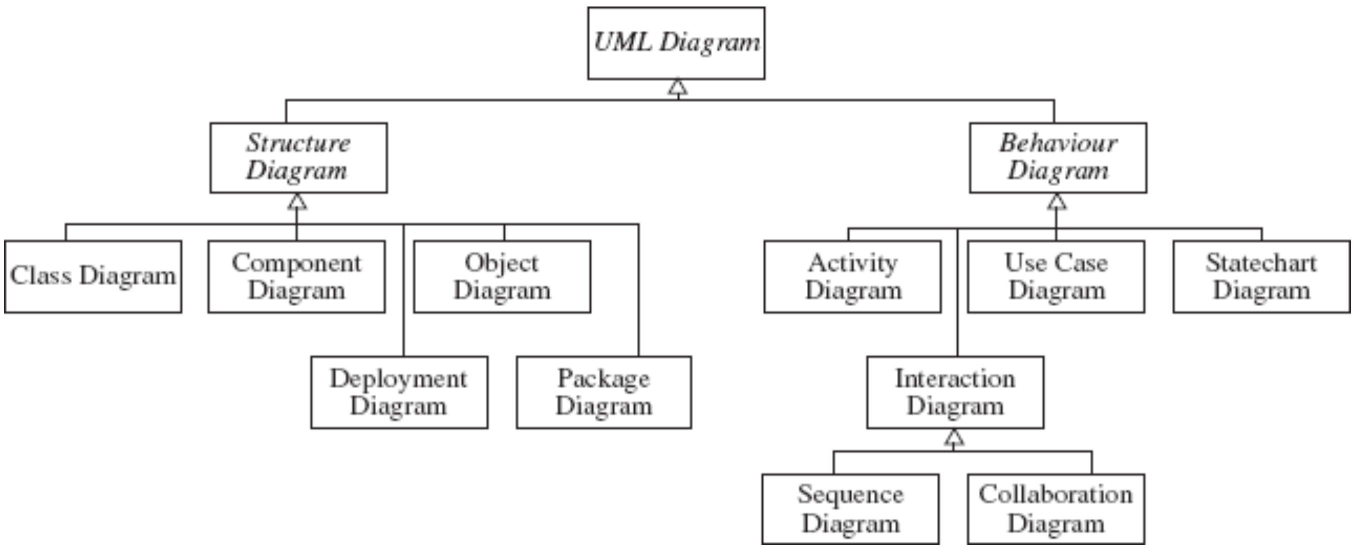


Fig. 2.27 *Hierarchy of UML diagrams*

Structural Diagrams

The **structural diagrams** describe the static or structural relationships among various components of the system. The structural diagrams are of five types.

- **Class diagram:** A class diagram describes the static structure of the entire system by showing the classes of the system, their attributes, and the relationships between them. Like E-R diagrams, a UML class diagram is also used to represent the conceptual database schema. The entity types in an E-R diagram become the classes in the UML class diagram.

- **Package diagram:** A package diagram can be treated as a subset of a class diagram. It organizes the elements of a system into related groups such that the dependencies between the various packages is minimized.
- **Object diagram:** An object diagram describes a set of objects (instances of a class) and their relationships. It is used to represent the static view of a system at a particular point of time and is typically used to test the class diagrams for accuracy.
- **Component diagram:** A component diagram displays the dependencies among various components of the system. The components include source code, run-time (binary) code, and executable code.
- **Deployment diagram:** A deployment diagram displays the physical resources in the system such as nodes, components, and connections. The **nodes** are the run-time processing elements.

Behavioural Diagrams

The **behavioural diagrams** describe the dynamic or behavioural relationships among various components of the system. The behavioural diagrams are of four types.

- **Use case diagram:** A use case diagram displays the relationship among actors and use cases. An **actor** represents a user or another system that interacts with the system and a **use case** is an external view of the system that represents some action.
- **Statechart diagram:** A statechart diagram describes the dynamic behaviour of the system by displaying the sequences of states that an object goes through during its life in response to some external events. It shows the start/initial state of an object before the occurrence of an external event and its stop/final state in response to that external event.

- **Activity diagram:** An activity diagram describes the dynamic behaviour of the system by modeling the flow of control from one activity to another. An **activity** is an operation performed on any class in the system that results in a change in the system state.
- **Interaction diagrams:** An interaction diagram models the dynamic characteristics of a system by representing the set of messages exchanged among a set of objects. They are further divided into two categories.
    - **Sequence diagram:** It describes the interactions among classes in terms of an exchange of messages over time. It consists of the vertical and horizontal dimensions, where vertical dimension represents the time and horizontal dimension represents different objects participating in the interactions.
    - **Collaboration diagram:** It represents the interactions among various objects in terms of a series of sequenced messages. In a collaboration diagrams, the objects are shown as icons and the messages are numbered to show a particular sequence of messages.

In sequence diagrams the emphasis is on the time-ordering of messages; however, in collaboration diagrams the emphasis is on structural organization of objects that interacts with each other by sending messages.

2.5.2 Representing Conceptual Schema Using Class Diagrams

In a class diagram each entity type of an E-R diagram becomes the class and is displayed as a box containing three sections. The top section contains the **class name**, the middle section contains the **attributes** of the class and the last section contains the **operations** that can be performed on the objects of the class. A binary relationship type is represented by just drawing a line connecting the participating classes. The name of the relationship type is written adjacent to the line. The role names can also be specified by writing the role name on the line. If the relationship type has descriptive attributes, they are shown in a box containing the name of the relationship type. This box is connected to the relationship line by a dashed line. The (*min, max*) notation is represented as *min* .. *max* notation. For example, the cardinality (1, 1) is shown as 1 .. 1, (1, N) is shown as 1 .. *, and (0, N) is shown as * or 0..*. Here, the symbol * indicates no maximum limit on participation. Figure 2.28 shows the UML class diagram for the *Online Book* database.

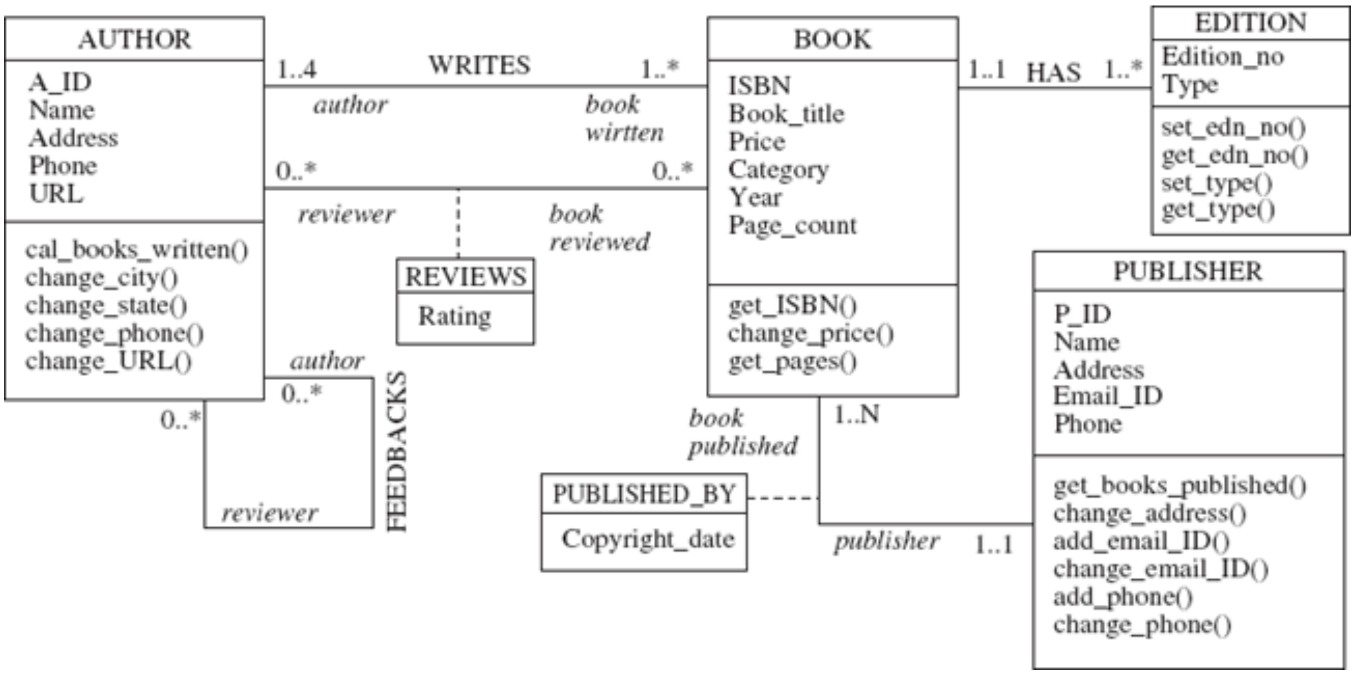
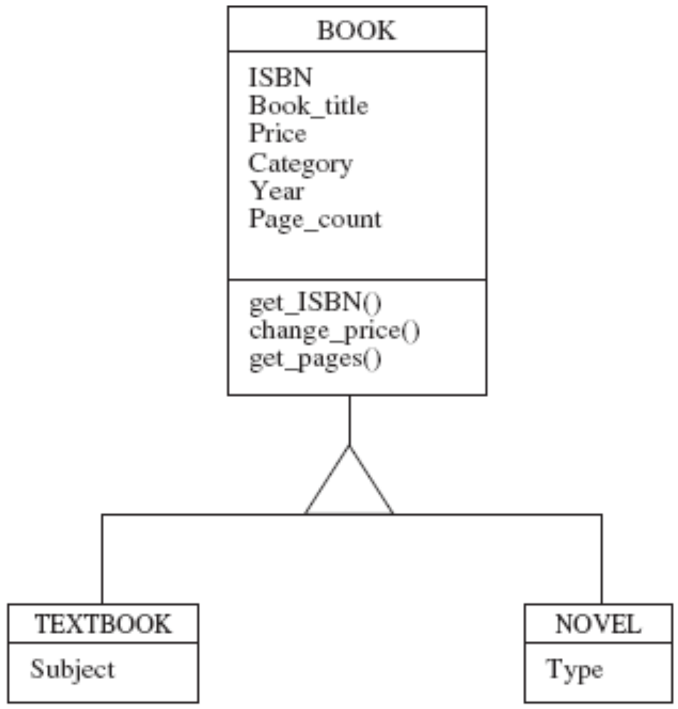


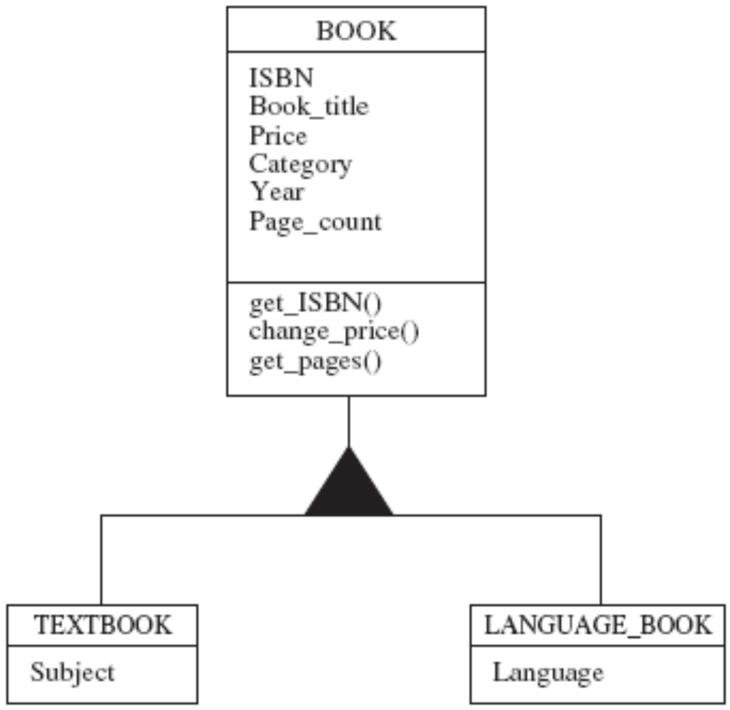
Fig. 2.28 *UML class diagram for the Online Book database*

Representing Specialization and Generalization

In UML class diagrams, specialization or generalization is represented by connecting the subclasses with the superclass by a blank or filled triangle. A blank triangle indicates specialization with disjoint constraint and a filled triangle indicates specialization with overlapping constraint. Figure 2.29 shows UML class diagrams representing specialization and generalization with disjoint and overlapping constraint. Note that both single and multiple inheirtance can be shown using UML class diagram notation. The topmost superclass is known as **base class** and the bottommost subclasses are known as **leaf classes**.



(a) UML class diagram showing specialization with disjoint constraint



(b) UML class diagram showing specialization with overlapping constraint

Fig. 2.29 *Specialization and generalization in UML class diagram*

SUMMARY

1. The conceptual modeling (or semantic modeling) is an important phase in designing a successful database. It represents various pieces of data and their relationships at a very high-level of abstraction. It mainly focuses on what data is required and how it

should be organized rather than what operations are to be performed on the data.

2. The conceptual model can be represented using two major approaches, namely, entity-relationship (E-R) modeling and the object modeling.
3. The E-R model views the real world as a set of basic objects (known as entities), and their characteristics (known as attributes) and associations among these objects (known as relationships).
4. An entity is a distinguishable object that has an independent existence in the real world. If an entity has a physical existence, it is termed as tangible or concrete entity and if it has a conceptual existence, it is termed as non-tangible or abstract entity.
5. Each entity is represented by a set of attributes. The attributes are the properties of an entity that characterize and describe it.
6. Each attribute can accept a value from a set of permitted values, which is called the domain or the value set of the attribute.
7. A set or a collection of entities that share the same attributes but different values is known as an entity type.
8. A specific occurrence of an entity type is called its instance. The collection of all instances of a particular entity type in the database at any point of time is known as an entity set.
9. The attributes of an entity are classified into five categories, namely, identifying and descriptive attributes, simple and composite attributes, stored, and derived attributes, single and multivalued attributes, and complex attributes.
10. Sometimes, there may be a situation where a particular entity may not have an appropriate value for an attribute. In such situations, the attribute takes a special value called *null* value.
11. The attribute (or combination of attributes), whose values are distinct for each individual instance of an entity type is known as a key attribute.
12. A set of one or more attributes, taken together, that helps in uniquely identifying each entity is called a superkey.
13. A minimal superkey that does not contain any superfluous (extra)

attribute in it is called a candidate key.

14. The candidate key, which is chosen by the database designer to uniquely identify entities, is known as the primary key.
15. If a primary key is formed by the combination of two or more attributes it is known as a composite key.
16. An entity type that does not have any key attributes of its own is called a weak entity type. On the other hand, an entity type that has a key attribute is called a strong entity type.
17. A weak entity type has a partial identifying attribute known as partial key (or discriminator), which is a set of attributes that can uniquely identify the instances of a weak entity type related to the same instance of a strong entity type.
18. An association between two or more entities is known as a relationship. The relationship between a weak entity type and its identifying entity type is known as an identifying relationship of the weak entity type.
19. The number of entity types participating in a relationship type determines the degree of the relationship type.
20. Each entity type that participates in a relationship type plays a specific function or a role in the relationship. The role of each participating entity type is represented by the role name.
21. Each relationship type has certain constraints that restrict the possible combination of instances participating in the relationship set. The constraints on the relationship type are of two types, namely, mapping cardinalities and participation constraint.
22. An E-R diagram is a specialized graphical tool that is used to represent the overall logical structure of the database.
23. An E-R diagram consists of a set of symbols that are used to represent different types of information. The symbols include rectangles, ellipses, diamonds, lines, double lines, double ellipses, dashed ellipses, and double rectangles.
24. An enhanced E-R diagram is used to represent some other aspects of a database such as inheritance among various entity types.
25. The process of defining the subgroups of a given entity type is

called specialization. The entity type containing the common attributes is known as the superclass and the entity type, which is a subset of the superclass, is known as its subclass.

26. The design process in which multiple lower-level entity types are combined on the basis of common features to form higher-level entity types is known as generalization.
27. If a subclass is a subset of only one superclass, that is, it has only one parent, it is known as single inheritance and the resulting structure is known as a specialization (or generalization) hierarchy. On the other hand, if a subclass is derived from more than one superclass, it is known as multiple inheritance and the resulting structure is known as a specialization (or generalization) lattice.
28. The process through which one can treat the relationships as higher-level entities is known as aggregation.
29. Unified modeling language (UML) is used to specify some other components of a software design process including functional modules of the system and interactions among them, user interactions with the system, etc.
30. UML includes ten standard diagrams that are divided into two main categories, namely, structural diagrams and behavioural diagrams.

KEY TERMS

- Conceptual modeling (Semantic modeling)
- Entity-relationship (E-R) model
- Entity
- Attributes
- Domain
- Entity type
- Entity set
- Identifying and descriptive attributes
- Simple and composite attributes
- Stored and derived attributes
- Single-valued and multivalued attributes
- Complex attributes

- NULL value
- Key attribute
- Superkey
- Candidate key
- Primary key
- Weak entity type
- Strong entity type
- Partial key
- Relationship
- Relationship type
- Relationship set
- Identifying relationship
- Relationship degree
- Role name
- Recursive relationship
- Mapping cardinalities
- One-to-one relationship
- One-to-many relationship
- Many-to-one relationship
- Many-to-many relationship
- Participation constraint
- Total (or mandatory) participation
- Partial (or optional) participation
- Entity-relationship diagram
- Enhanced E-R model
- Specialization
- Generalization
- Superclass
- Subclass
- Attribute inheritance
- Single and multiple inheritance
- Condition-defined specialization
- User-defined specialization
- Disjoint constraint

- Overlapping constraint
- Completeness constraint
- Total specialization
- Partial specialization
- Aggregation
- Unified modeling language (UML)
- Structural diagrams
- Behavioural diagrams

EXERCISES

A. Multiple Choice Questions

1. The E-R model is relevant to which of the following phases of database design?
    1. Requirement analysis
    2. Conceptual database design
    3. Logical database design
    4. None of these
2. Which of these statements describes an entity correctly?
    1. An entity is a distinguishable object that has an independent existence in the real world
    2. Each entity is represented by a set of attributes
    3. An entity can exist either physically or conceptually
    4. All of these
3. The attribute that is used to uniquely identify an instance of an entity is known as **________.**
    1. Unique attribute
    2. Simple attribute
    3. Identifying attribute
    4. Multivalued attribute
4. Which of these attributes can be considered as identifying attribute for an entity Student?
    1. Roll number
    2. Marks

   3. Address
   4. Any of these
5. The strong entity type and weak entity type should participate in _________.
   1. Many-to-many relationship
   2. Many-to-one relationship
   3. One-to-many relationship
   4. One-to-one relationship
6. In an E-R diagram an entity is represented by a ________.
   1. Square
   2. Rectangle
   3. Diamond-shaped box
   4. Ellipse
7. Which of these statements is correct?
   1. An entity may be related to only one other entity
   2. An entity may be related to itself
   3. An entity may be related to many other entities
   4. All of these
8. The relationship between a weak entity type and its identifying entity type is known as ____________.
   1. Identifying relationship
   2. Weak relationship
   3. Strong relationship
   4. Relationship type
9. The relationship between two entity types is known as ____________.
   1. Binary relationship
   2. Two-way relationship
   3. Multiple relationship
   4. Double relationship
10. A person has a PAN card. This relationship is:
    1. One-to-one
    2. One-to-many
    3. Many-to-many

4. Many-to-one

B. Fill in the Blanks

1. If an entity has a physical existence, it is termed as ________ entity.
2. A set of entities that share the same attributes but different values, is known as an ________.
3. The attribute (or combination of attributes) whose values are distinct for each individual instance of an entity type is known as a ________.
4. The constraints on the relationship type are of two types, namely, ___________ and ______________.
5. An ___________ is a specialized graphical tool that demonstrates the interrelationships among various entities of a database.
6. The process of defining the subgroups of a given entity type is called ___________.
7. In ___________, the membership of entities is determined by placing a condition on the value of some attribute of the superclass.
8. ______________ constraint specifies that the same higher-level entity instance cannot belong to more than one lower-level entity types.
9. The process through which one can treat the relationships as higher-level entities is known as ___________.
10. ____________ is a standard language, which is officially defined by the Object Management Group (OMG) as one of the techniques to implement object data modeling.

C. Answer the Questions

1. Write a short note on conceptual data modeling.
2. What is an E-R model? List some of its advantages and limitations. How do you overcome these limitations?
3. What is an entity? Differentiate between tangible and non-tangible entity.
4. Differentiate between an attribute and a domain.
5. Define the following terms:

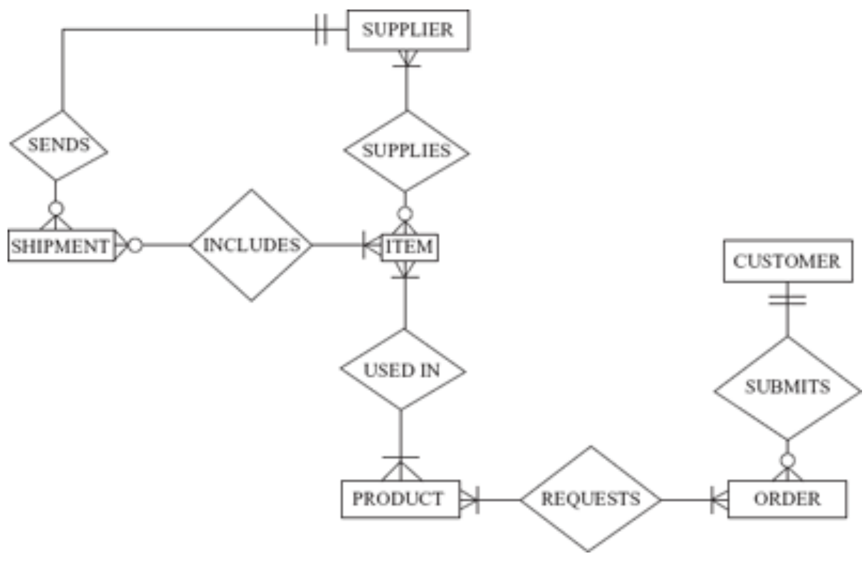
   1. Entity type
   2. Entity set
   3. Instance
   4. NULL value
   5. Attributes
   6. Key attribute
   7. Composite key
   8. Domain
   9. Single-valued and multi-valued attributes
   10. Derived attributes
   11. UML
   12. Attribute inheritance
   13. Cardinality ratio
   14. Degree of relationship
   15. Weak entity
6. What are the various types of attributes? Discuss the various situations in which an attribute can use a *null* value.
7. Explain the difference between primary key, candidate key, and superkey.
8. Differentiate between strong and weak entity type.
9. What are relationship type and relationship instance? Explain the difference between the two.
10. What is identifying relationship?
11. What is the significance of using role names in the description of relationship types? In what situations are role names necessary?
12. Explain the term 'degree of a relationship' with the help of an example.
13. Discuss the various structural constraints that are applied on relationship types.
14. What is meant by recursive relationship type? Give an example also.
15. Discuss various symbols that are used in the E-R diagrams to represent various types of information.
16. Discuss the E-R diagram naming conventions.
17. Illustrate the issues to be considered while developing an E-R

diagram.

18. How are weak entity types represented in an E-R diagram?
19. What is an EER model?
20. Differentiate between specialization and generalization with the help of an example. Can we represent their differences with the help of an E-R diagram? Explain.
21. Differentiate between disjoint and overlapping constraint.
22. List the possible types of relations that may exist between two entities.
23. What is the difference between specialization hierarchy and specialization lattice?
24. What is an aggregation? Explain with the help of an example.
25. What are the various alternative notations to represent E-R diagrams?
26. Define UML. What is UML model? Explain its use and discuss the different categories of UML diagrams.

D. Practical Questions

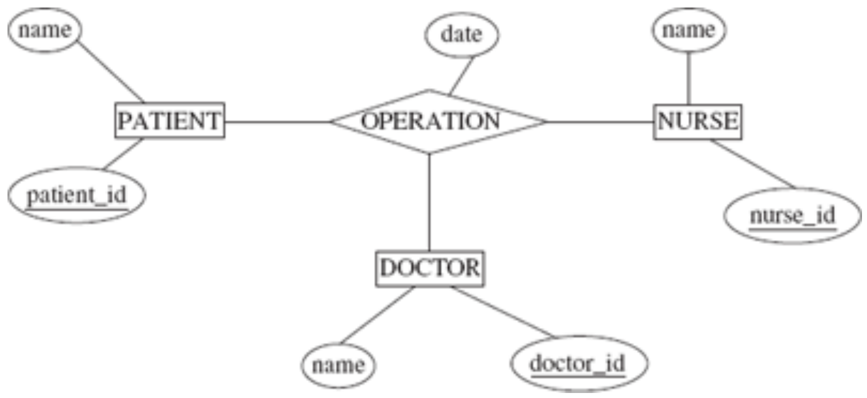
1. Identify the entity types, relationship types, and mapping cardinalities in the following E-R diagram.



2. Consider the following set of requirements for a COMPANY database that is used to keep track of its employees, departments, and projects.
   1. The company stores the information about the currently

    working employees. The information includes employee number, name, gender, salary, date of birth, date of joining, address, and phone number. Each employee works for a department on a particular project for a particular number of hours.
    2. The information about departments includes department number and department name. Each department controls some projects currently running in the company. Also each department is managed by a particular employee who becomes the manager for that department. This employee also supervises all the other employees in that department.
    3. The project information includes project number, project name and its description.
    4. An employee can work for only one department; however, a department can have any number of employees. A department is managed by only one manager and a manager can manage only one department. A department can control any number of projects; however, one project can be handled by only one department. Any number of employees can work on any number of projects.

    Design an E-R diagram for COMPANY database. Specify the key attributes of each entity type, role names, and mapping cardinalities. Make appropriate assumptions to complete the specification.
3. Draw the EER diagram for the COMPANY database described in question 2 by showing the specialization of employee entity type. The subclasses include *secretary, technician, engineer, manager, fulltime employee, and part-time employee*. Show all the possible constraints. Make appropriate assumptions, wherever necessary.
4. Give an example of multiple relationships between entity types. Draw the E-R diagram for illustration.
5. Consider the following E-R diagram describing surgeries or operations in a HOSPITAL database.



1. Represent the relation type OPERATION as a weak entity type with some additional constraints that an operation is performed on exactly one patient by one or more doctors and one or more nurses.
2. A doctor can be a consultant or a registrar, but not both and there are some doctors who are neither consultants nor registrars. Represent this information with the help of an extended E-R model.

6. Construct an E-R diagram for a BANK database having customer, loan, account, employee, and branch as entity types. A customer has an account in a particular branch of the bank. The customer can also borrow loan from the bank. The bank has number of employees working in different branches of the bank. Add appropriate attributes for each entity type. Represent the key attributes, weak entity types (if any), cardinality ratios, and role names of each entity type. Make appropriate assumptions to complete the specification.
7. Draw the UML diagrams equivalent to the E-R diagrams constructed for Questions 1, 2, and 5.