UNIT - I

Introduction to Database

1.0 Introduction

Database is a collection of related data. Database management system is software designed to assist the maintenance and utilization of large scale collection of data. DBMS came into existence in 1960 by Charles. Integrated data store which is also called as the first general purpose DBMS. Again in 1960 IBM brought IMS-Information management system. In 1970 EdgorCodd at IBM came with new database called RDBMS. In 1980 then came SQL Architecture- Structure Query Language. In 1980 to 1990 there were advances in DBMS e.g. DB2, ORACLE.

Data

• Data is raw fact or observations.
• When activities in the organization takes place, the effect of these activities need to be recorded which is known as Data.

Information

• Processed data is called information
• The purpose of data processing is to generate the information required for carrying out the business activities.

Data and Information

Data is defined as a known fact that can be recorded and that have implicit meaning. Actually data are raw or isolated facts from which the required information is produced.

Information is defined as a collection of related data when put together communicate meaningful and useful message to a recipient who uses it, to make decision or to interpret the data to get the meaning.

Data versus Information

Data	Information
Data is raw fact and figures. For example, 89 is data.	Data when stored in some form like Marks: 89 ; then becomes an information
Data is not significant to a business.	Information is significant to a business.
Data are atomic level pieces of information.	Information is a collection of data.
Data does not help in decision making.	Information helps in decision making.
Data is generally in unorganised form.	Information is in organised form.
Data is collected from the source directly and hence is not dependent on information.	Information is dependent on data that is gathered.

Metadata is defined as the data about the data. Metadata is the data which describes the objects in the database. So it is easy to aces those objects. It describes the database structure, constraints, applications, size of data types. It is also known as System Catalogue.

Data item is a container of data that is registered with the sever.

Records are composed of fields each of which contains one item of information.

A **file** is a set of records.

Database System Applications

Database are widely used. Here some representative applications:

- **Banking**: For customer information, accounts, loans and banking transactions.
- **Airlines**: For reservations and schedule information.
- **Universities**: For student information, course registration and grades.
- **Credit card transactions**: For purchase on credit cards and generation of monthly statements.
- **Telecommunication**: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards and storing information about the communication networks.
- **Sales**: For customer, product and purchase information.
- **On-line retailers**: For sales data noted above plus on-line order tracking, generation of recommendation lists, and maintenance of on-line product evaluations.
- **Manufacturing**: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores and for orders of items.
- **Human Resource**: For information about employees, salaries, payroll taxes, benefits and for generation of pay cheques.

In general data management consists of following tasks

• Data capture: Which is the task associated with gathering the data as and when they originate.
• Data classification: Captured data has to be classified based on the nature and intended usage.
• Data storage: The segregated data has to be stored properly.
• Data arranging: It is very important to arrange the data properly
• Data retrieval: Data will be required frequently for further processing,
Hence it is very important to create some indexes so that data can be retrieved easily.
• Data maintenance: Maintenance is the task concerned with keeping the data upto-

date.
• Data Verification: Before storing the data it must be verified for any error.
• Data Coding: Data will be coded for easy reference.
• Data Editing: Editing means re-arranging the data or modifying the data for presentation.
• Data transcription: This is the activity where the data is converted from one form into another.
• Data transmission: This is a function where data is forwarded to the place where it would be used further.
Metadata (**meta data**, or sometimes **meta information**) is "data about data", of any sort in any media. An item of metadata may describe a collection of data including multiple content items and hierarchical levels, for example a database schema. In data processing, metadata is definitional data that provides information about or documentation of other data managed within an application or environment. The term should be used with caution as all data is about something, and is therefore metadata.

Database
• Database may be defined in simple terms as a collection of relateddata
• A database is a collection of related data.
• The database can be of any size and of varying complexity.
• A database may be generated and maintained manually or it may be computerized.
In addition, a database must have the following properties:

- ✓ It represents some aspect of the real world, called miniworld. changes to the miniworld are reflected in the database.
- ✓ it is a logically collection of data, to which some meaning can be attached.

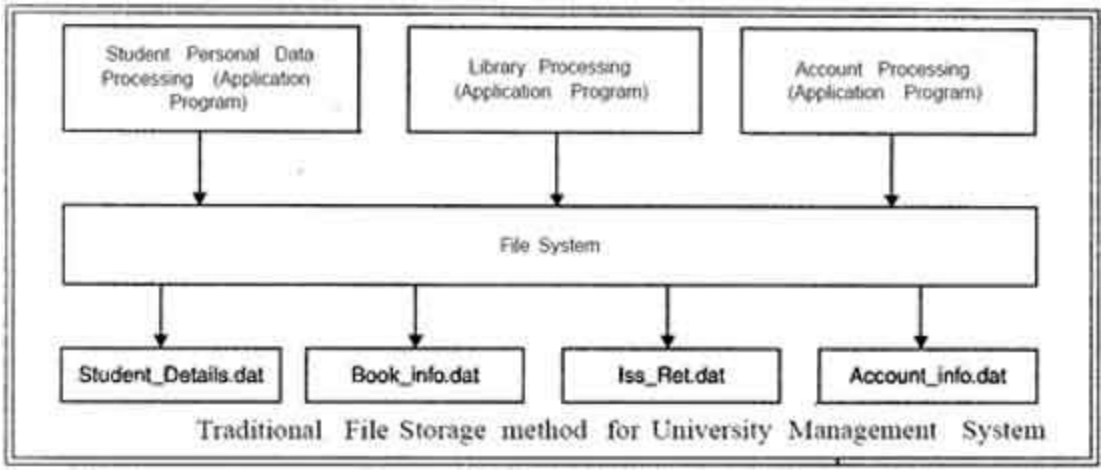
Traditional File Processing System

A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File system may use a storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files.

The manual filing system works well when the number of items to be stored is small. It even works quite adequately when there are large numbers of items and we have only to store and retrieve them. However, the manual filing system breaks down when we have to cross-reference or process the information in the files.

Clearly the manual system is inadequate for this' type of work. The file based system was developed in response to the needs of industry for more efficient data access. In early processing systems, an organization's information was stored as groups of records in separate files.

In the traditional approach, we used to store information in flat files which are maintained by the file system under the operating system's control. Here, flat files are files containing records having no structured relationship among them. The file handling which we learn under C/C ++ is the example of file processing system. The Application programs written in C/C ++ like programming languages go through the file system to access these flat.files as shown.



Traditional File Storage method for University Management System

Characteristics of File Processing System

Here is the list of some important characteristics of file processing system:

• It is a group of files storing data of an organization.

• Each file is independent from one another.

• Each file is called a flat file.

• Each file contained and processed information for one specific function, such as accounting or inventory.

• Files are designed by using programs written in programming languages such as COBOL, C, C++.

• The physical implementation and access procedures are written into database application; therefore, physical changes resulted in intensive rework on the part of the programmer.

• As systems became more complex, file processing systems offered little flexibility, presented many limitations, and were difficult to maintain.

Limitations of the File Processing System *I* File-Based Approach

There are following problems associated with the File Based Approach:

1.Separated and Isolated Data: To make a decision, a user might need data from two separate files. First, the files were evaluated by analysts and programmers to determine the specific data required from each file and the relationships between the data and then applications could be written in a programming language to process and extract the needed data. Imagine the work involved if data from several files was needed.

2.Duplication of data: Often the same information is stored in more than one file. Uncontrolled duplication of data is not required for several reasons, such as:

• Duplication is wasteful. It costs time and money to enter the data more than once

• It takes up additional storage space, again with associated costs.

• Duplication can lead to loss of data integrity; in other words the data is no longer consistent.

For example, consider the duplication of data between the Payroll and Personnel departments. If a member of staff moves to new house and the change of address is communicated only to Personnel and not to Payroll, the person's pay slip will be sent to the wrong address. A more serious problem occurs if an employee is promoted with an associated increase in salary. Again, the change is notified to Personnel but the change does not filter through to Payroll. Now, the employee is receiving the wrong salary. When this error is detected, it will take time and effort to resolve. Both these examples, illustrate inconsistencies that may result from the duplication of data. As there is no automatic way for Personnel to update the data in the Payroll files, it is difficult to foresee such inconsistencies arising. Even if Payroll is notified of the changes, it is possible that the data will be entered incorrectly.

3. Data Dependence: In file processing systems, files and records were described by specific physical formats that were coded into the application program by programmers. If the format of a certain record was changed, the code in each file containing that format must be updated. Furthermore, instructions for data storage and access were written into the application's code. Therefore, .changes in storage structure or access methods could greatly affect the processing or results of an application.

In other words, in file based approach application programs are data dependent. It means that, with the change in the physical representation (how the data is physically represented in disk) or access technique (how it is physically accessed) of data, application programs are also affected and needs modification. In other words application programs are dependent on the how the data is physically stored and accessed.

If for example, if the physical format of the master/transaction file is changed, by making he modification in the delimiter of the field or record, it necessitates that the application programs which depend on it must be modified.

Let us consider a student file, where information of students is stored in text file and each field is separated by blank space as shown below:

1Rohit 35 Nagapar

Now, if the delimiter of the field changes from blank space to semicolon as shown below:

1; Rohit; 35; Nagapar

Then, the application programs using this file must be modified, because now it will token the field on semicolon; but earlier it was blank space.

4. Difficulty in representing data from the user's view: To create useful applications for the user, often data from various files must be combined. In file processing it was difficult to determine relationships between isolated data in order to meet user requirements.

5. Data Inflexibility: Program-data interdependency and data isolation, limited the flexibility of file processing systems in providing users with ad-hoc information requests

6. Incompatible file formats: As the structure of files is embedded in the application programs, the structures are dependent on the application programming language. For example, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a 'C' program. The direct incompatibility of such files makes them difficult to process jointly.

7. Data Security. The security of data is low in file based system because, the data is maintained in the flat file(s) is easily accessible. For Example: Consider the Banking System. The Customer Transaction file has details about the total available balance of all customers. A Customer wants information about his account balance. In a file system it is difficult to give the Customer access to only his data in the· file. Thus enforcing security constraints for the entire file or for certain data items are difficult.

8. Transactional Problems. The File based system approach does not satisfy transaction properties like Atomicity, Consistency, Isolation and Durability properties commonly known as ACID properties.

For example: Suppose, in a banking system, a transaction that transfers Rs. 1000 from account A to account B with initial values' of A and B being Rs. 5000 and Rs. 10000 respectively. If a system crash occurred after the withdrawal of Rs. 1000 from account A,

but before depositing of amount in account B, it will result an inconsistent state of the system. It means that the transactions should not execute partially but wholly. This concept is known as Atomicity of a transaction (either 0% or 100% of transaction). It is difficult to achieve this property in a file based system.

9. Concurrency problems. When multiple users access the same piece of data at same interval of time then it is called as concurrency of the system. When two or more users read the data simultaneously there is ll(problem, but when they like to update a file simultaneously, it may result in a problem.

Database Management System

• A Database Management System (DBMS) is a collection of program that enables user to create and maintain a database.

• The DBMS is hence a general purpose software system that facilitating each of the following :

1. definition: specifying data types
2. construction: the process of storing the data on some medium that is controlled by the DBMS.
3. manipulation: querying, updating, report generation
4. sharing: allowing multiple users and programs to access the dataset "simultaneously"
5. system protection: preventing database from becoming corrupted when hardware or software failures occur.
6. security protection: preventing unauthorized or malicious access to database

Characteristics of DBMS

1) Self-Describing Nature of a Database System
2) Insulation between Programs and Data, and Data Abstraction
3) Support of Multiple Views of the Data
4) Sharing of Data and Multiuser Transaction Processing

Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the system **catalog,** which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data,** and it describes the structure of the primary database.

Insulation between Programs and Data, and Data Abstraction

in traditional file processing approach, data definition is a part of the application program. Hence programs would be able to work with only one specific database. However in the database approach, data definition is stored in the DBMS catalog separately from access programs. This property is called as "**Program-data independence".** Further application programs can operate on the data by invoking operations (functions) regardless of how these operations are implemented. This is termed as "**Program operation independence".**

This characteristic of the database that allows program-data independence and program-operation independence is called **data abstractions.**

Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or **view** of the database. A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of applications must provide facilities for defining multiple views.

Sharing of Data and Multiuser Transaction Processing

Traditional file processing approach did not support sharing of data. However, the modern database approach supports sharing of data as well as multi-user transactions. For this the DBMS includes features such as concurrency control to ensure that several users trying to update the same data do so in a controlled manner. It also enforces isolation property, atomicity property etc. to promote this.

Actors on the Scene

1) Database Administrators
2) Database Designers
3) End Users
4) System Analysts and Application Programmers (Software Engineers)

1 Database Administrators (DBA)

DBA: person in the organization who controls the design and the use of the database refers as DBA..The job of a DBA is to create the database and implement the technical controls and security**.DBA** an IT professional who manages a DBMS for an enterprise.The work of a DBA is :

- Managing the resources
- Creation of user accounts
- Providing security and authorization

- Manage poor system response time
- Restoration of system after failure(system recovery)
- Tuning the database depending upon the need of the user
- Coordinate and monitor and
- Acquiring software and hardware resources as needed.

The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.

2. Database Designers

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
It is the responsibility of database designers to communicate with all prospective database users, in order to understand their requirements, and to come up with a design that meets these requirements.
Database designers typically interact with each potential group of users and develop a **view** of the database that meets the data and processing requirements of this group.
These views are then analyzed and *integrated* with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

3 End Users

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

- **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.
- **Naive** or **parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called **canned transactions—**that have been carefully programmed and tested.

Sophisticated end users include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.

- **Stand-alone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu- or graphics-based interfaces. An example

is the user of a tax package that stores a variety of personal financial data for tax purposes.

Naive end users need to learn very little about the facilities provided by the DBMS.
Casual users learn only a few facilities that they may use repeatedly.
Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements.
Stand-alone users typically become very proficient in using a specific software package.

4 System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements. **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers (nowadays called **software engineers**) should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

Workers behind the Scene

In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS *software and system environment.* These persons are typically not interested in the database itself. We call them the "workers behind the scene," and they include the following categories.

• **DBMS system designers and implementers** are persons who design and implement the DBMS modules and interfaces as a software package. A DBMS is a complex software system that consists of many components or **modules,** including modules for implementing the catalog, query language, interface processors, data access, concurrency control, recovery, and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages.

- **Tool developers** include persons who design and implement **tools**—the software packages that facilitate database system design and use, and help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.
- **Operators and maintenance personnel** are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Although the above categories of workers behind the scene are instrumental in making the database system available to end users, they typically do not use the database for their own purposes.

Advantages of Using a DBMS

1 Controlling Redundancy
2 Restricting Unauthorized Access
3 Providing Persistent Storage for Program Objects and Data Structures
4 Permitting Inferencing and Actions Using Rules
5 Providing Multiple User Interfaces
6 Representing Complex Relationships among Data
7 Enforcing Integrity Constraints
8 Providing Backup and Recovery

1 Controlling Redundancy

A good DBMS must control redundancy . Redundancy refers to the existence of same data multiple times in the database. It leads to **duplication of efforts, wastage of storage space and inconsistency.** Ideally a good DBMS design must store each logical data item at only one place in the database.

2 Restricting Unauthorized Access

When multiple users share a database, it is likely that some users will not be authorized to access all information in the database. For example, financial data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas others are allowed both to retrieve and to update. Hence, the type of access operation—retrieval or update—must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a **security and authorization subsystem,** which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically. Notice that we can apply similar controls to the DBMS software. For example, only the DBA's staff may be allowed to use certain **privileged software,** such as the software for creating new accounts. Similarly, parametric users may be allowed to access the database only through the canned transactions developed for their use.

3 Providing Persistent Storage for Program Objects and Data Structures

It is another important expectation from a good database. Traditional database system normally suffered from "impedance-mismatch problem" since the data structures provided by DBMS and the programming languages were incompatible. However object-oriented database systems typically offer data structure compatibility.

4 Providing Multiple User Interfaces

Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include query languages for casual users; programming language interfaces for application programmers; forms and command codes for parametric users; and menu-driven interfaces and natural language interfaces for stand-alone users. Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs).** Many specialized languages and environments exist for specifying GUIs. Capabilities for providing World Wide Web access to a database—or web-enabling a database—are also becoming increasingly common.

5 Representing Complex Relationships Among Data

A database may include numerous varieties of data that are interrelated in many ways. A DBMS must have the capability to represent a variety of complex relationships among the data as well as to retrieve and update related data easily and efficiently.

6 Enforcing Integrity Constraints

Most database applications have certain **integrity constraints** that must hold for the data such as specifying a 'data type' for each item, specifying ' uniqueness' on data item values etc., a DBMS should provide capabilities for defining and enforcing these constraints.

7 Providing Backup and Recovery

A DBMS must provide facilities for recovering from hardware or software failures. The **backup and recovery subsystem** of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the program started executing. Alternatively, the recovery subsystem could ensure that the program is resumed from the point at which it was interrupted so that its full effect is recorded in the database.

Difference between File System and DBMS

	Difference factor	File system	DBMS
1	Definition	File system is an abstraction to store,retrieve,manage and update a set of files. It keep track on the files and also manage them	DBMS is a collection of interrelated data and a set of programs to access those data.
2	Data redundancy	Each user defines and implements the needed files for a specific application to run.same files exists in different places	DBMS controls the amount of redundancy .
3	Sharing data	It doesnot allow sharing of data	In DBMS data can be shared very easily due to centralized system
4	Data consistency	When data is redundanct, it is difficult to update the data in different places	In DBMS , as there is no or less data redundancy, data remains consistent
5	Difficult to search/access data	In file system, if we want to search/retrieve/access some data item, it becomes very difficult because in file system for every operation we have to make different program	in DBMS,it is easy because searching and querying operations are already available in the system
6	Data isolation	There is no standard format of data or we can say data is scattered in various formats or files which also make data retrieval difficult	Due to centralized system the format of similar type of data remains same
7	Data integrity	The value of data must follow or satisfy some rules or consistency constraints	DBMS maintains the data integrity by enforcing the constraints by adding appropriate code
8	Security problem	There is no or very less security	DBMS have high level security like encryption, passwords, biometric etc.

Implications of the Database Approach

1) Potential for Enforcing Standards

2) Reduced Application Development Time
3) Flexibility
4) Availability of Up-to-Date Information
5) Economies of Scale

Potential for Enforcing Standards

The database approach permits the DBA to define and enforce standards among database users in a large organization. This facilitates communication and cooperation among various departments, projects, and users within the organization. Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on. The DBA can enforce standards in a centralized database environment more easily than in an environment where each user group has control of its own files and software.

Reduced Application Development Time

A prime selling feature of the database approach is that developing a new application—such as the retrieval of certain data from the database for printing a new report—takes very little time. Designing and implementing a new database from scratch may take more time than writing a single specialized file application. However, once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.

Flexibility

It may be necessary to change the structure of a database as requirements change. For example, a new user group may emerge that needs information not currently in the database. In response, it may be necessary to add a file to the database or to extend the data elements in an existing file. Modern DBMSs allow certain types of changes to the structure of the database without affecting the stored data and the existing application programs.

Availability of Up-to-Date Information

A DBMS makes the database available to all users. As soon as one user's update is applied to the database, all other users can immediately see this update. This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases, and it is made possible by the concurrency control and recovery subsystems of a DBMS.

Economies of Scale

The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its own (weaker) equipment. This reduces overall costs of operation and management.

When Not to Use a DBMS

Main inhibitors (costs) of using a DBMS:

- High initial investment and possible need for additional hardware.
- Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- When a DBMS may be unnecessary:
- If the database and applications are simple, well defined, and not expected to change.
- If there are stringent real-time requirements that may not be met because of DBMS overhead.
- If access to data by multiple users is not required.
- When no DBMS may suffice:
- If the database system is not able to handle the complexity of data because of modeling limitations
- If the database users need special operations not supported by the DBMS.

Database System Concepts and Architecture

Data Models, Schemas, and Instances

data model:

it a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction.

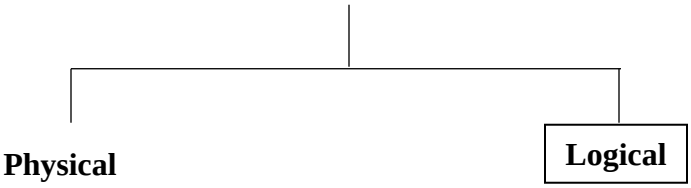
By *structure of a database* we mean the data types, relationships, and constraints that should hold on the data.

Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

Types of Data Models

1. High Level- Conceptual data model.
2. Low Level – Physical data model.
3. Relational or Representational
4. Object-oriented Data Models:
5. Object-Relational Models:

Classification of data model



Heaps ISAM HASH VSAM BTREE	Object based	Record based
	E-R Model	Hierarchical Network Relational

1. High Level-conceptual data model: User level data model is the high level or conceptual model. This provides concepts that are close to the way that many users perceive data.

2 .Low level-Physical data model : provides concepts that describe the details of how data is stored in the computer model. Low level data model is only for Computer specialists not for end-user.

3. Representation data model: It is between High level & Low level data model Which provides concepts that may be understood by end-user but that are not too far removed from the way data is organized by within the computer.

The most common data models are

1. Relational Model

The Relational Model uses a collection of tables both data and the relationship among those data. Each table have multiple column and each column has a unique name .

Relational database comprising of two tables

Customer-ame	SecurityNumber	Address	City	Account-number
Preethi	111-222-3456	Yelhanka	Bangalore	A-101
Sharan	111-222-3457	Hebbal	Bangalore	A-125

Preethi	112-123-9878	Jaynagar	Bangalore	A-456
Arun	123-987-9909	MG road	Bangalore	A-987
Preethi	111-222-3456	Yelhanka	Bangalore	A-111
Rocky	222-232-0987	Sanjay Nagar	Bangalore	A-111

Account –Table

Account-Number	Balance
A-101	1000.00
A-125	1200.00
A-456	5000.00
A-987	1234.00
A-111	3000.00

Customer Preethi and Rocky share the same account number A-111

Advantages
1. The main advantage of this model is its ability to represent data in a simplified format.
2. The process of manipulating record is simplified with the use of certain key attributes used to retrieve data.
3. Representation of different types of relationship is possible with this model.

2. Network Model

The data in the network model are represented by collection of records and relationships among data are represented by links, which can be viewed as pointers.

The records in the database are organized as collection of arbitrary groups.

Advantages:
1. Representation of relationship between entities is implemented using pointers which allows the representation of arbitrary relationship
2. Unlike the hierarchical model it is easy.
3. data manipulation can be done easily with this model.

3. Hierarchical Model

A hierarchical data model is a data model which the data is organized into a tree like structure. The structure allows repeating information using parent/child relationships: each parent can have many children but each child only has one parent. All attributes of a specific record are listed under an entity type.

Advantages:
1. The representation of records is done using an ordered tree, which is natural method of implementation of one–to-many relationships.
2. Proper ordering of the tree results in easier and faster retrieval of records.
3. Allows the use of virtual records. This result in a stable database especially when modification of the data base is made.

Object-oriented Data Models
• Several models have been proposed for implementing in a database system.
• One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).
• Additionally, systems like O2, ORION (at MCC – then ITASCA), IRIS (at H.P.-used in Open OODB).

3.2 The Entity-Relationship Model

An *entity-relationship model* describes data in terms of the following:

1. Entities
2. Relationship between entities
3. Attributes of entities

We graphically display an E-R model using an ***entity-relationship diagram*** (or E-R diagram) like the sample in Figure 3.1. While this figure may seem to be confusing at first glance, its meaning should become very clear by the end of this chapter.

We will now discuss the components of an E-R diagram in detail.

Object-Relational Models
• Most Recent Trend. Started with Informix
• Universal Server.
• Relational systems incorporate concepts from object databases leading to objectrelational.
• Object Database Standard: ODMG-93, ODMG-version 2.0,ODMG-version 3.0.
• Exemplified in the latest versions of Oracle-10i,DB2, and SQL Server and other DBMSs.
• Standards included in SQL-99 and expected to be enhanced in future SQL standards.

Schemas versus Instances
• Database Schema:
The *description of a database.*

Includes descriptions of the database structure, data types, and the constraints on the database.

• Schema Diagram:

An ***illustrative display of (most aspects of) a*** database schema.

• Schema Construct:

A ***component of the schema or an object within*** the schema, e.g., STUDENT, COURSE.

• Database State:

The actual data stored in a database at aparticular moment in time. This includes the collection of all the data in the database.Also called database instance (or occurrence or snapshot).

• The term *instance is also applied to individual* database components, e.g. *record instance, table instance, entity instance*

Database Schema vs. Database State

The description of a database is called the database schema. It is specified during the database design phase and is not expected to change frequently. A pictorial representation of the database schema is called as the schema diagram.

The actual data present in the database at any particular point of time is called as a database state(or snapshot). The database state(actual data) may change from time to time frequently.

The distinction between database schema and database state is very important. When we define a new database, we specify only the database schema to the DBMS. At this point, the corresponding database state is the empty state with no data. We get the initial state of the database when the database is first populated with data. From then on, every time an update operation is applied to a database state, we get another database state.

The database schema is sometimes called as the “intension” and a database state is called an “extension” of the schema.

Example of a database schema

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

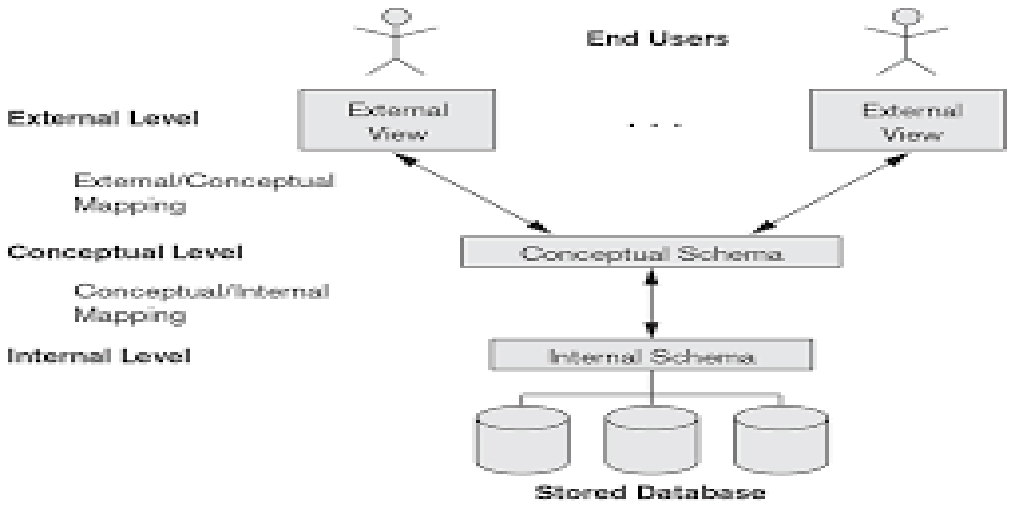
DBMS Architecture and Data Independence

- The Three-Schema Architecture
- Data Independence

The Three-Schema Architecture

The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema,** which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema,** which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.
3. The **external** or **view level** includes a number of **external schemas** or **user views.** Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.



The 3 schemas given above are only descriptions of the data. The actual data exists at the physical level. In the 3-schema architecture, each user group refers to its own external schema. Hence the DBMS must transform a request specified on the external schema into a request on the conceptual schema which in turn must transform to a request on the internal schema. The data extracted from the database must be re-formatted to match the user's external view. This process of transforming requests and results between levels is called "MAPPING".

Data Independence

Data independence,can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. Changes to conceptual schema may include the expansion of the database, adding constraints, reducing the database etc.,
2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may include reorganization of files, creation of new access structures etc.,

Normally physical data independence exists in most databases. The exact location of data on the disk, encoding, compression, splitting, merging of records etc are hidden from the user. On the other hand logical data independence is hard to achieve. Since it demands the schema and the next higher level schema to be intact and only the mapping between the 2 levels to be changed.

DBMS Languages

• Data Definition Language (DDL)
• Data Manipulation Language (DML)
• High-Level or Non-procedural Languages: These include the relational language SQL
• May be used in a standalone way or may be embedded in a programming language
• Low Level or Procedural Languages:
These must be embedded in a programming language

Data Definition Language (DDL)

Used by the DBA and database designers to specify the conceptual schema of a database.
• In many DBMSs, the DDL is also used to define internal and external schemas (views).
Database Management System
VTU-EDUSAT Page 26
• In some DBMSs, separate **storage definition language (SDL) and view definition language (VDL) are used to define internal and external** schemas.
• SDL is typically realized via DBMS commands provided to the DBA and database designers

Data Manipulation Language (DML)

Used to specify database retrievals and updates DML commands (data sublanguage) can be*embedded in a general-purpose programming* language (host language), such as COBOL, C, C++, or Java.
• A library of functions can also be provided to access the DBMS from a programming language
• Alternatively, stand-alone DML commands can be applied directly (called a *query language).*

Types of DML

• **High Level or Non-procedural Language:**
For example, the SQL relational language are "set"-oriented and specify what data to retrieve rather than how to retrieve it.
Also called **declarative languages.**
• **Low Level or Procedural Language:**
Retrieve data one record-at-a-time;
Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

DBMS Interfaces

1. Stand-alone query language interfaces

Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)

2. Programmer interfaces for embedding DML in programming languages
3. User-friendly interfaces
4. Menu-based, forms-based, graphics-based etc.

1. DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
- **Embedded Approach: e.g embedded SQL (for C,**C++, etc.), SQLJ (for Java)
- **Procedure Call Approach: e.g. JDBC for Java,** ODBC for other programming languages

2. **Database Programming Language Approach:**
   e.g. ORACLE has PL/SQL, a programming language based on SQL;
   language incorporates SQL and its data types as integral components/

3. User-Friendly DBMS Interfaces

• Menu-based, popular for browsing on the web
• Forms-based, designed for naïve users
• Graphics-based (Point and Click, Drag and Drop, etc.)
• Natural language: requests in written English
• Combinations of the above:For example, both menus and forms usedextensively in Web database interfaces

5. Other DBMS Interfaces

• Speech as Input and Output
• Web Browser as an interface
• Parametric interfaces, e.g., bank tellers using function keys.
• Interfaces for the DBA:
• Creating user accounts, granting authorizations
• Setting system parameters
• Changing schemas or access paths

The database system environment

1 DBMS Component Modules

2 Database System Utilities

The DBMS is a complex software system.

1.Typical DBMS Component Modules

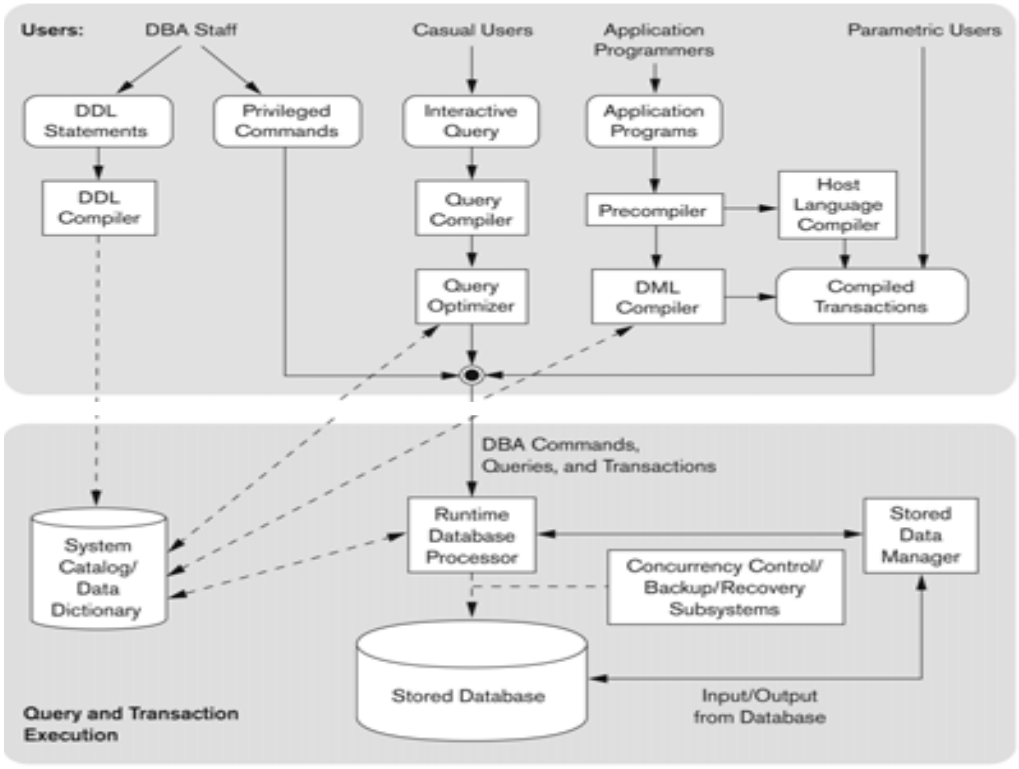


Figure 2.3
Component modules of a DBMS and their interactions.

The figure is divided into two halves. The top half of the figure refers to the various users of the database environment and their interfaces. The lower half shows the internals of the DBMS responsible for storage of data and processing of transaction.
The database and the DBMS catalog are usually stored on disk. Access to the disk is Primarily controlled by operating system(OS).which includes disk input/output. A higher level stored data manager module of DBMS controls access to DBMS information that is stored on the disk.
If we consider the top half of the figure, It shows interfaces to DBA staff, casual users, application programmers and parametric users The DDL compiler processes schema definitions, specified in the DDL and stores the description of the schema in the DBMS Catalog..The catalog includes information such as names and sizes of the sizes of the

files, data types of data of data items. Storage details of each file, mapping information among schemas and constraints.
Casual users and persons with occasional need of information from database interact using some of interface which is interactive query interface. The queries are parsed, analysed for correctness of the operations for the model. The names of the data elements and so on by a query compiler that compiles them into internal form. The internal query is subjected to query optimization..The query optimizer is concerned with rearrangement and possible recording of operations, eliminations of redundancies.
Application programmer writes programs in host languages. The precompiler extracts DML commands from an application program.

2 Database System Utilities

In addition to possessing the software modules just described, most DBMSs have **database utilities** that help the DBA in managing the database system. Common utilities have the following types of functions:

1.*Loading:* A loading utility is used to load existing data files—such as text files or sequential files—into the database. Usually, the current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database. With the proliferation of DBMSs, transferring data from one DBMS to another is becoming common in many organizations. Some vendors are offering products that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas). Such tools are also called **conversion tools.**

2. ***Backup:*** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape. The backup copy can be used to restore the database in case of catastrophic failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex but it saves space.
3. ***File reorganization:*** This utility can be used to reorganize a database file into a different file organization to improve performance.
4. ***Performance monitoring:*** Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files to improve performance.

Other utilities may be available for sorting files, handling data compression, monitoring access by users, and performing other functions.

Centralized and Client-Server DBMS Architectures

Centralized DBMS:

• Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.

• User can still connect through a remote terminal – however, all processing is done at centralized site.

A Physical Centralized Architecture

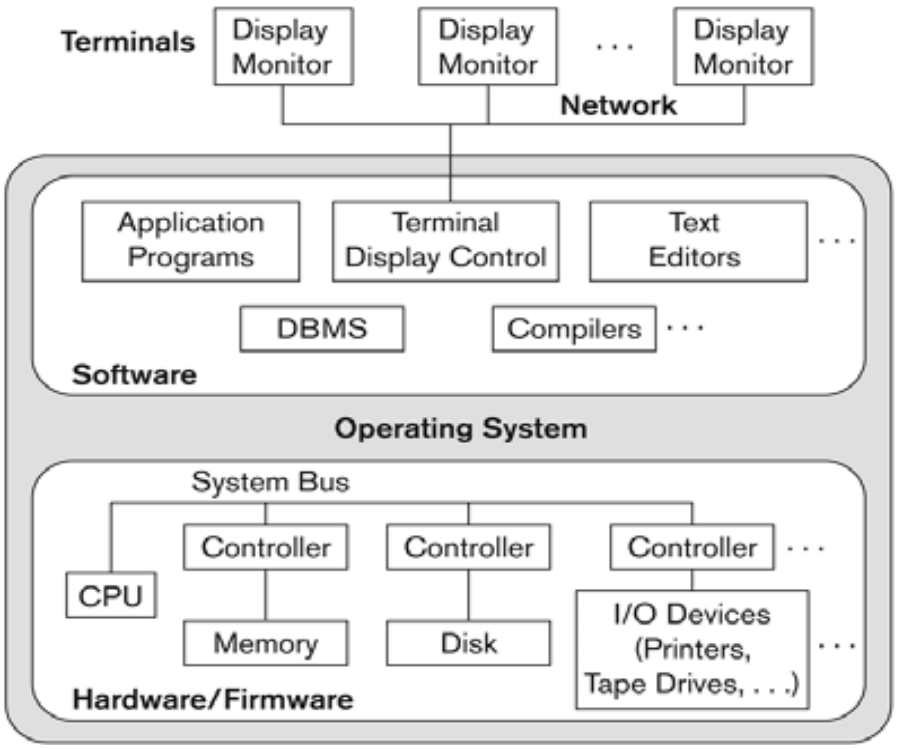


Figure 2.4
A physical centralized architecture.

Architectures for DBMS have followed trends similar to those generating computer system architectures. Earlier architectures used mainframes computers to provide the main processing for all system functions, including user application programs and user interface programs as well all DBMS functionality. The reason was that most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities. Therefore all processing was performed remotely on the computer system, and only display information and controls were sent from the computer to the display terminals, which were connected to central computer via various types of communication networks.

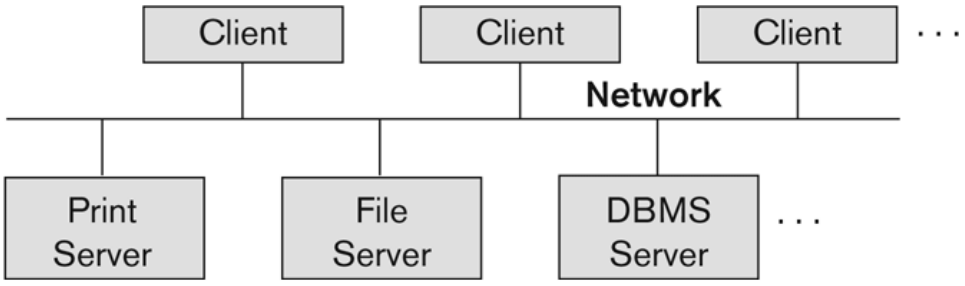
As prices of hardware declined, most users replaced their terminals with PCs and workstations. At first database systems used these computers similarly to how they have used is play terminals, so that DBMS itself was still a Centralized DBMS in which all the DBMS functionality, application program execution and user interface processing were carried out on one Machine.

Basic 2-tier Client-Server Architectures

• Specialized Servers with Specialized functions
• Print server
• File server
• DBMS server
• Web server
• Email server
• Clients can access the specialized servers as needed

Logical two-tier client server architecture

Figure 2.5
Logical two-tier client/server architecture.



Clients

• Provide appropriate interfaces through a client software module to access and utilize the various server resources.
• Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.
• Connected to the servers via some form of a network.
• (LAN: local area network, wireless network, etc.)

DBMS Server

• Provides database query and transaction services to the clients
• Relational DBMS servers are often called SQL servers, query servers, or transaction servers
• Applications running on clients utilize an Application Program Interface (**API) to access server databases via** standard interface such as:
• ODBC: Open Database Connectivity standard
• JDBC: for Java programming access
• Client and server must install appropriate client module and server module software for ODBC or JDBC

Two Tier Client-Server Architecture

• A client program may connect to several DBMSs, sometimes called the data sources.
• In general, data sources can be files or other non-DBMS software that manages data. Other variations of clients are possible: e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

Three Tier Client-Server Architecture

• Common for Web applications
• Intermediate Layer called Application Server or Web Server:
• Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
• Acts like a conduit for sending partially processed data between the database server and the client.
• Three-tier Architecture Can Enhance Security:
• Database server only accessible via middle tier
• Clients cannot directly access database server

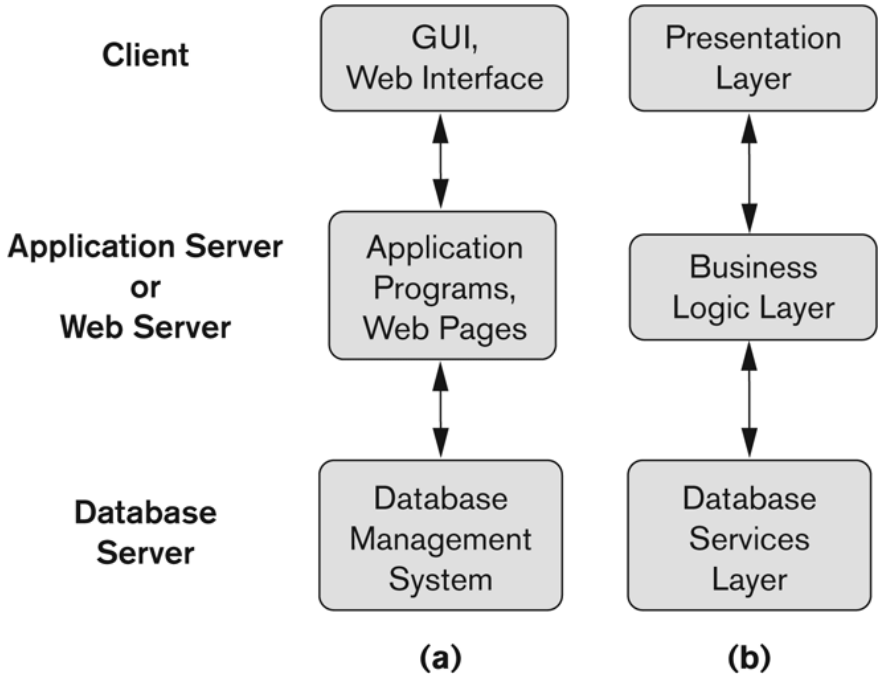


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

Classification of DBMSs

• Based on the data model used
• Traditional: Relational, Network, Hierarchical.
• Emerging: Object-oriented, Object-relational.
• Other classifications
• Single-user (typically used with personal computers) vs. multi-user (most DBMSs).
• Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)

Variations of Distributed DBMSs (DDBMSs)

• Homogeneous DDBMS
• Heterogeneous DDBMS
• Federated or Multidatabase Systems
• Distributed Database Systems have now come to be known as client-server based database systems because:
    • They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

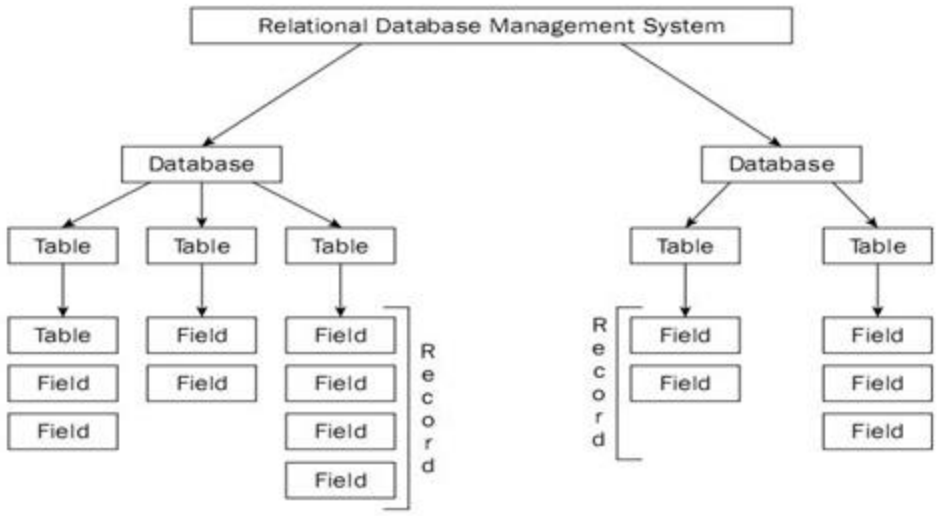
Cost considerations for DBMSs

• Cost Range: from free open-source systems to configurations costing millions of dollars
• Examples of free relational DBMSs: MySQL, PostgreSQL, others

Classification Of Database Management System

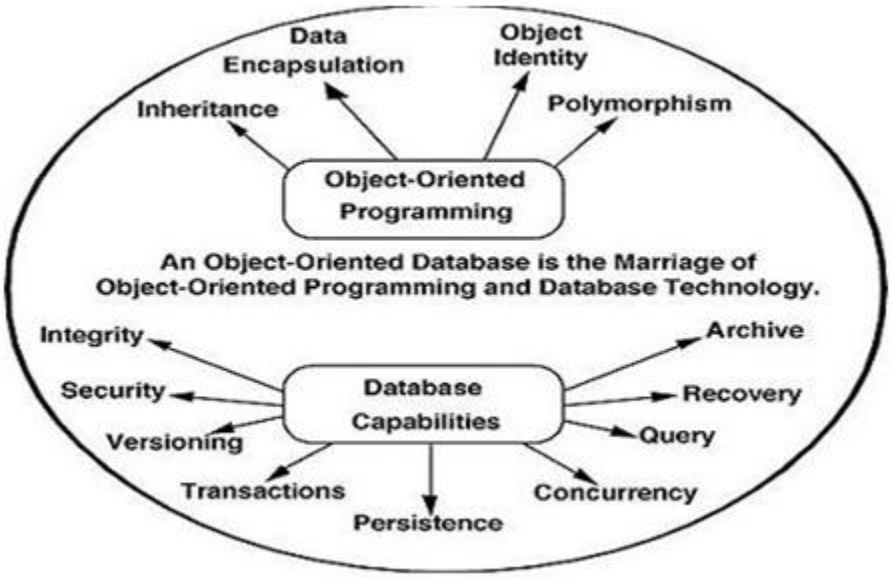
Based on the data model

Relational database – This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Few examples are MYSQL(Oracle, open source), Oracle database (Oracle), Microsoft SQL server(Microsoft) and DB2(IBM).



Object oriented database – The information here is in the form of the object as used in object oriented programming. It adds the database functionality to object programming languages. It requires less code, use more natural data and also code bases are easy to maintain. Examples are ObjectDB (ObjectDB software).

Object relational database – Relational DBMS are evolving continuously and they have been incorporating many concepts developed in object database leading to a new class called extended relational database or object relational database.



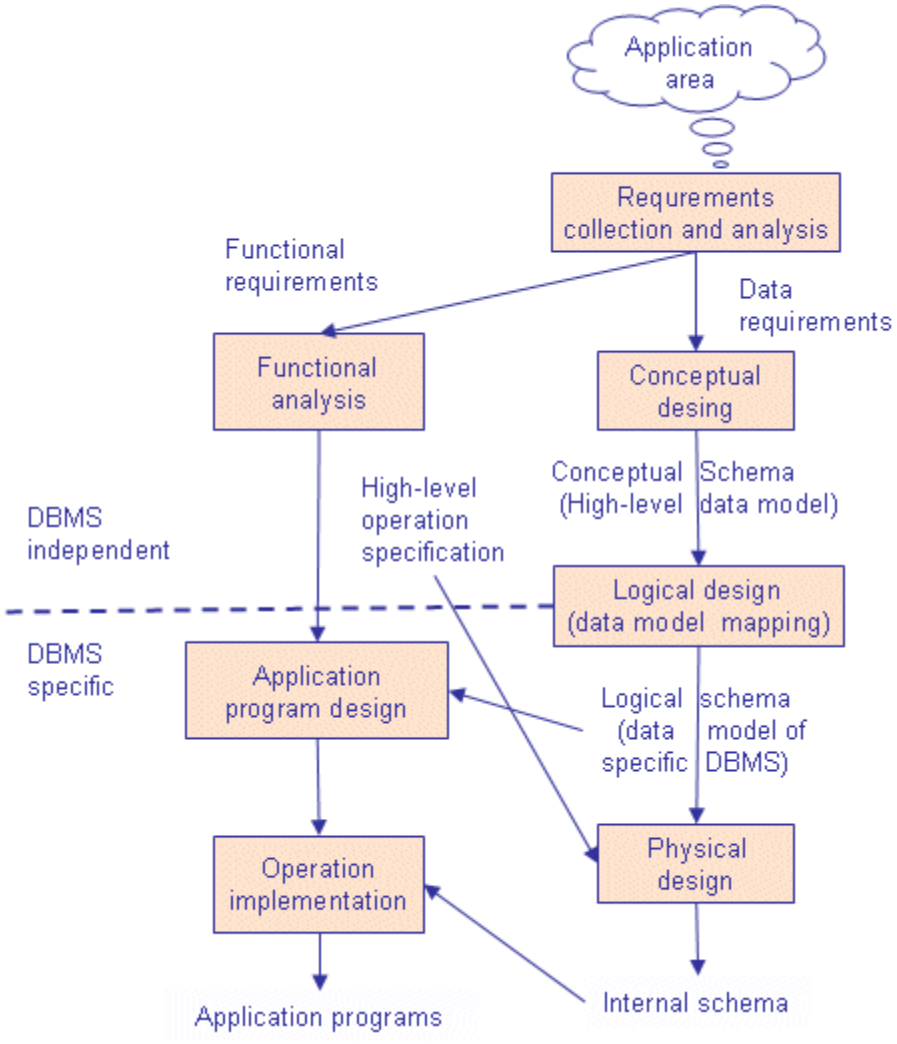
Hierarchical database – In this, the information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree. Here the data follows a series of records, set of values attached to it. They are used in industry on mainframe platforms. Examples are IMS(IBM), Windows registry(Microsoft).

Data Modeling Using the Entity-Relationship Model

Using High-Level Conceptual Data Models for Database Design

Or

What are the phases of a database design



The first step is called as the requirement collection and analysis phase. During this phase, the database designer interviews the database users to understand their expectations. At the end of the first phase, the database designers generate the Data Requirements and Functional Requirements.

Once the first step has been completed, the second step involves the functional analysis of the functional requirements and the preparation of the conceptual design using the

Data Requirements. This phase includes the creation of Entity types, relationships and constraints.

The third step in the database design is the actual implementation of the database using a commercial DBMS such as ORACLE. This phase generates the database schema. This phase also marks the shift from the high level data model into implementation data model.

The last step in the database design is the physical design phase during which internal structures, indexes, access paths, file organization etc. are specified. Along with this, this phase also involves the design of database transactions to match the functional requirements.

3.2 The Entity-Relationship Model

An *entity-relationship model* describes data in terms of the following:

1. Entities
2. Relationship between entities
3. Attributes of entities

We graphically display an E-R model using an ***entity-relationship diagram*** (or E-R diagram) like the sample in Figure 3.1. While this figure may seem to be confusing at first glance, its meaning should become very clear by the end of this chapter.

We will now discuss the components of an E-R diagram in detail.

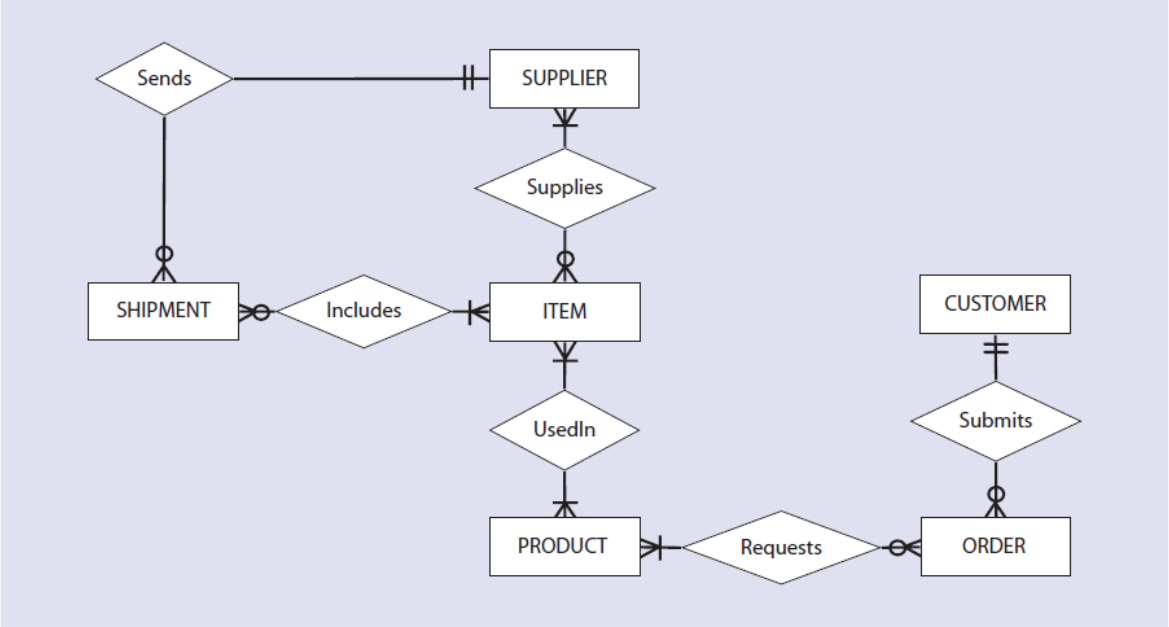


Figure 3.1 Example of an E-R diagram.

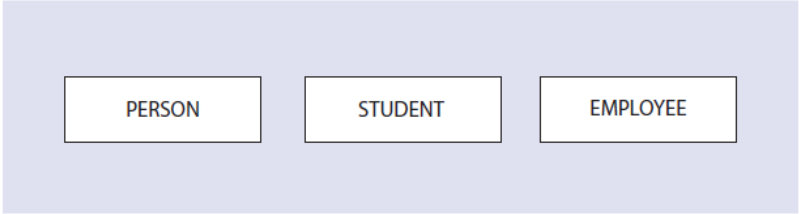


Figure 3.2 The entity representation in an E-R diagram.

3.4 *Attributes*

We represent an entity with a set of attributes. An ***attribute*** is a property or characteristic of an entity type that is of interest to an organization. Some attributes of common entity types include the following:

STUDENT = {Student ID, SSN, Name, Address, Phone, Email, DOB}
ORDER = {Order ID, Date of Order, Amount of Order}
ACCOUNT = {Account Number, Account Type, Date Opened, Balance}
CITY = {City Name, State, Population}

We use the following conventions while naming attributes:

1. Each word in a name starts with an uppercase letter followed by lower case letters.
2. If an attribute name contains two or more words, the first letter of each subsequent word is also in uppercase, unless it is an article or preposition, such as "a," "the," "of," or "about" (see Figure 3.3).

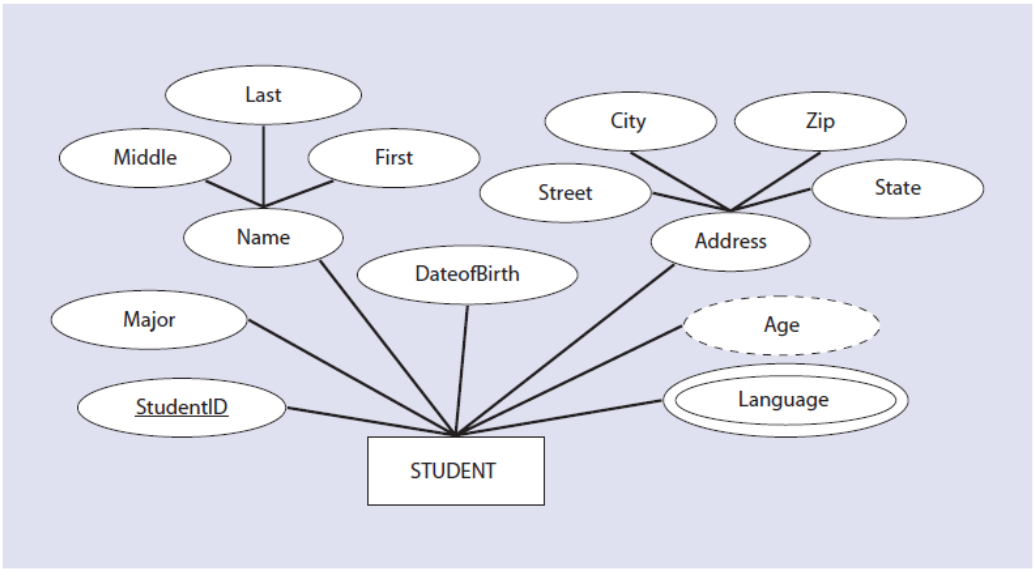


Figure 3.3 Attributes of the STUDENT entity type.

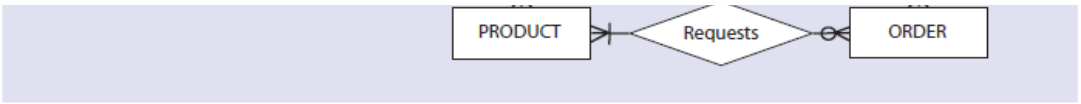


Figure 3.1 Example of an E-R diagram.

3.3 *Entity*

An ***entity*** is an object that exists and which is distinguishable from other objects. An entity can be a person, a place, an object, an event, or a concept about which an organization wishes to maintain data. The following are some examples of entities:

Person: STUDENT, EMPLOYEE, CLIENT
Object: COUCH, AIRPLANE, MACHINE
Place: CITY, NATIONAL PARK, ROOM, WAREHOUSE
Event: WAR, MARRIAGE, LEASE
Concept: PROJECT, ACCOUNT, COURSE

It is important to understand the distinction between an *entity type*, an *entity instance*, and an *entity set*. An ***entity type*** defines a collection of entities that have same attributes. An ***entity instance*** is a single item in this collection. An ***entity set*** is a set of entity instances. The following example will clarify this distinction: STUDENT is an entity type; a student with ID number 555-55-5555 is an entity instance; and a collection of all students is an entity set.

In the E-R diagram, we assign a name to each entity type. When assigning names to entity types, we follow certain naming conventions. An entity name should be a concise singular noun that captures the unique characteristics of the entity type. An E-R diagram depicts an entity type using a rectangle with the name of the entity inside (see Figure 3.2).

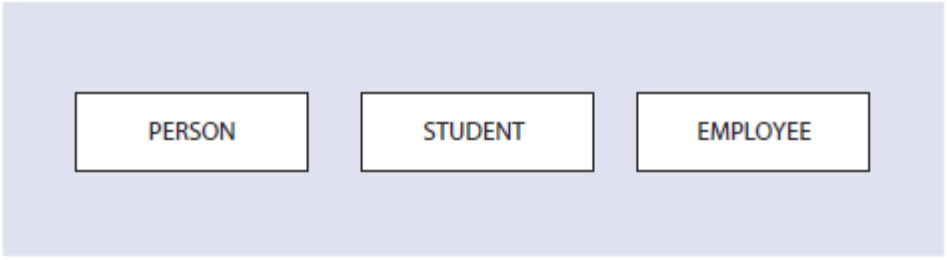


Figure 3.2 The entity representation in an E-R diagram.

Attributes

We represent an entity with a set of attributes. An ***attribute*** is a property or characteristic of an entity type that is of interest to an organization. Some attributes of common entity types include the following:

STUDENT = {Student ID, SSN, Name, Address, Phone, Email, DOB}
ORDER = {Order ID, Date of Order, Amount of Order}
ACCOUNT = {Account Number, Account Type, Date Opened, Balance}
CITY = {City Name, State, Population}

We use the following conventions while naming attributes:

1. Each word in a name starts with an uppercase letter followed by lower case letters.
2. If an attribute name contains two or more words, the first letter of each subsequent word is also in uppercase, unless it is an article or preposition, such as "a," "the," "of," or "about" (see Figure 3.3).

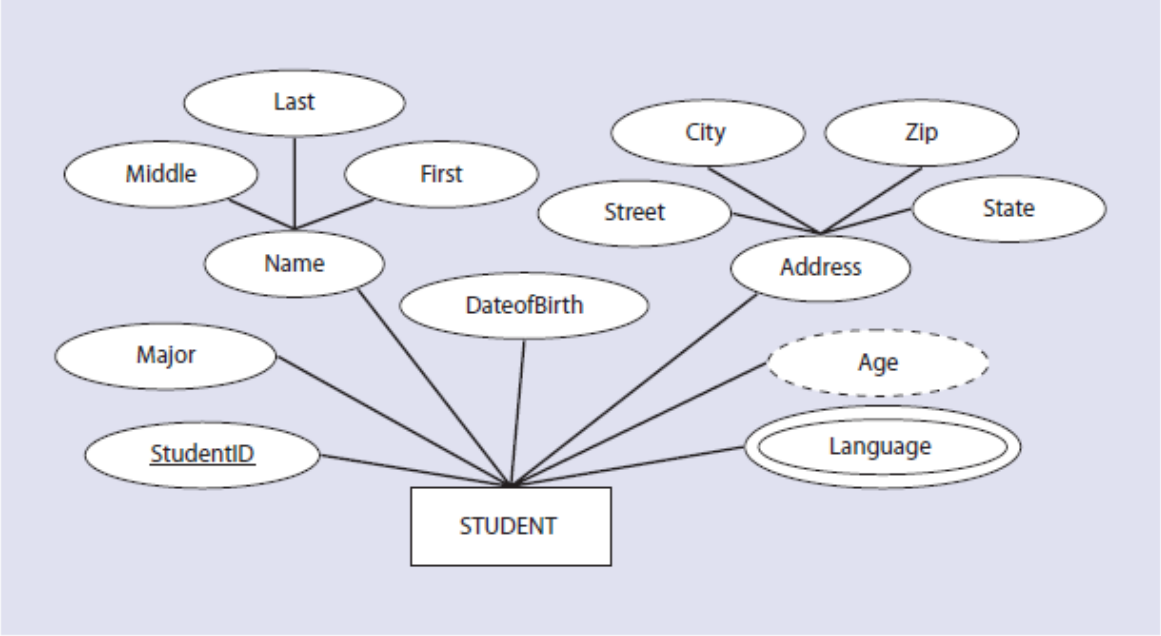


Figure 3.3 Attributes of the STUDENT entity type.

E-R diagrams depict an attribute inside an ellipse and connect the ellipse with a line to the associated entity type. Figure 3.3 illustrates some of the possible attributes in an E-R diagram for the entity STUDENT.

Notice that not all of the attributes in Figure 3.3 are marked in the same way. There are actually several types of attributes featured in this figure. These include: simple, composite, single-valued, multi-valued, stored, and derived attributes. In the following subsections, we discuss the distinctions between these types of attributes.

3.4.1 Simple and Composite Attributes

A ***simple*** or an ***atomic attribute***, such as *City* or *State*, cannot be further divided into smaller components. A ***composite attribute***, however, can be divided into smaller subparts in which each subpart represents an independent attribute. *Name* and *Address* are the only composite attributes in Figure 3.3. All other attributes, even those that are subcategories of *Name* and *Address*, are simple attributes. The figure also presents the notation that depicts a composite attribute.

3.4.2 Single-Valued and Multi-Valued Attributes

Most attributes have a single value for an entity instance; such attributes are called ***single-valued attributes***. A ***multi-valued attribute***, on the other hand, may have more than one value for an entity instance. Figure 3.3 features one multi-valued attribute, *Languages*, which stores the names of the languages that a student speaks. Since a student may speak several languages, it is a multi-valued attribute. All other attributes of the STUDENT entity type are single-valued attributes. For example, a student has only one date of birth and one student identification number. In the E-R diagram, we denote a multi-valued attribute with a double-lined ellipse. Note that in a multi-valued attribute, we always use a double-lined ellipse, regardless of the number of values.

3.4.3 Stored and Derived Attributes

The value of a ***derived attribute*** can be determined by analyzing other attributes. For example, in Figure 3.3 *Age* is a derived attribute because its value can be derived from the current date and the attribute *DateofBirth*. An attribute whose value cannot be derived from the values of other attributes is called a ***stored attribute***. As we will learn, a derived attribute *Age* is not stored in the database. Derived attributes are depicted in the E-R diagram with a dashed ellipse.

3.4.4 Key Attribute

A ***key attribute*** (or identifier) is a single attribute or a combination of attributes that uniquely identify an individual instance of an entity type. No two instances within an entity set can have the same key attribute value. For the STUDENT entity shown in Figure 3.3, *StudentID* is the key attribute since each student identification number is unique. *Name*, by contrast, cannot be an identifier because two students can have the same name. We underline key attributes in an E-R diagram (also see Figure 3.4).

Sometimes no single attribute can uniquely identify an instance of an entity type. However, in these circumstances, we identify a set of attributes that, when combined, is unique for each entity instance. In this case the key attribute, also known as ***composite key***, is not a simple attribute, but a composite attribute that uniquely identifies each entity instance.

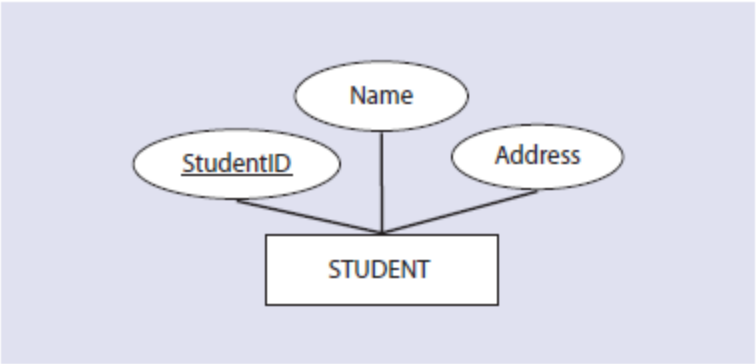


Figure 3.4 The key attribute.

A composite key must be minimal in the sense that no subset of a composite key can form the key of the entity instance. For example, if a composite key has four attributes, A1 to A4, then any subset, say A2, A4 or A2, A3 (or any of 16 combinations), should not form a key for an entity. In other words, we need all attributes, A1–A4, to identify each instance of an entity uniquely. In the E-R diagram, we underline each attribute in the composite key.

For example, consider the CITY entity type (see Figure 3.5). This category includes, potentially, all the cities in the United States. Notice that none of the attributes (i.e. *Name*, *State* or *Population*) can serve as a key attribute since there are many cities in each state and two cities could possibly have the same name or population. However, the composite attribute {*Name*, *State*} is a valid key attribute for the CITY entity as no two cities within a state can have the same name.

An entity can have more than one attribute that qualifies to be an identifier. For the entity shown in Figure 3.6, each of the attributes *Name*, *StateAbbr*, and *UnionOrder* (the order in which the state entered the union of the United States) can be an identifier. In this case, it is a matter of preference as to which attribute is made an identifier or key attribute.

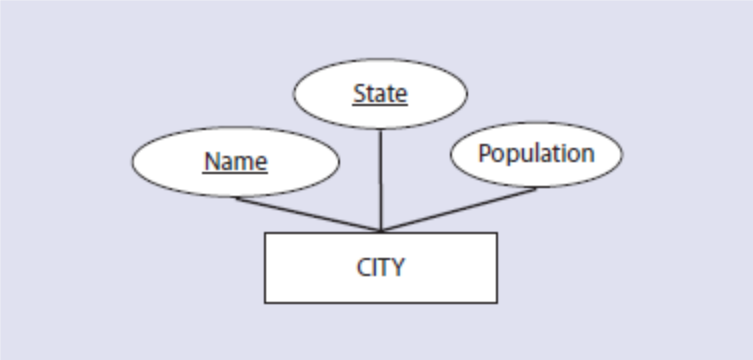


Figure 3.5 The composite key attribute.

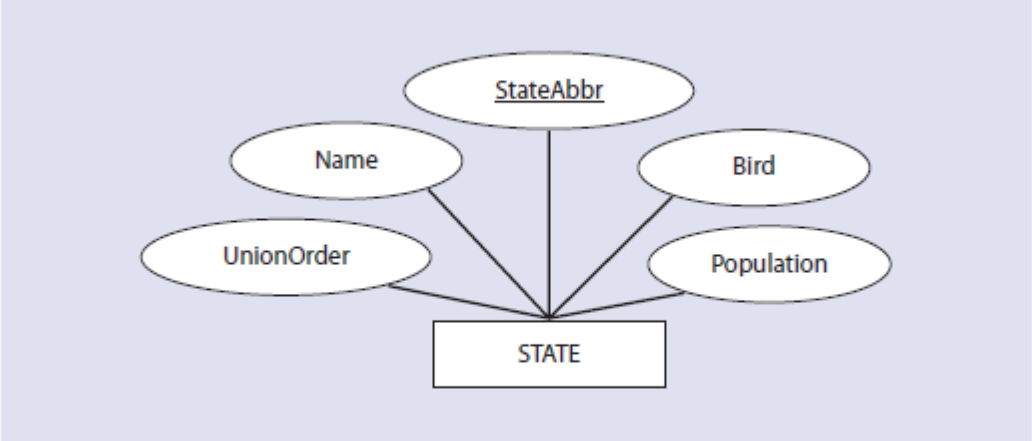


Figure 3.6 An example of more than one key attribute.

Relationships

Entities in an organization do not exist in isolation but are related to each other. Students take courses and each STUDENT entity is related to the COURSE entity. Faculty members teach courses and each FACULTY entity is also related to the COURSE entity. Consequently, the STUDENT entity is related to the FACULTY entity through the COURSE entity. E-R diagrams can also illustrate relationships between entities.

We define a ***relationship*** as an association among several entities. Consider, for example, an association between customers of a bank. If customer Williams has a bank account number 523, then the quality of ownership constitutes a ***relationship instance*** that associates the CUSTOMER instance Williams with the ACCOUNT instance *523*. We can think of the relationship instance as a verb that links a subject and an object: customer Williams *has* an account; student John *registers* for a course; professor Smith *teaches* a course. A ***relationship set*** is a grouping of all matching relationship instances, and the term ***relationship type*** refers to the relationship between entity types. For example, Figure 3.7 illustrates a relationship set between the CUSTOMER and the ACCOUNT instances.

In an E-R diagram, we represent relationship types with diamond-shaped boxes connected by straight lines to the rectangles that represent participating entity types. A relationship type is a given name that is displayed in this diamond-shaped box and typically takes the form of a present tense verb or verb phrase that describes the relationship. An E-R diagram may depict a relationship as the following example of the relationship between the entities CUSTOMER and ACCOUNT does:

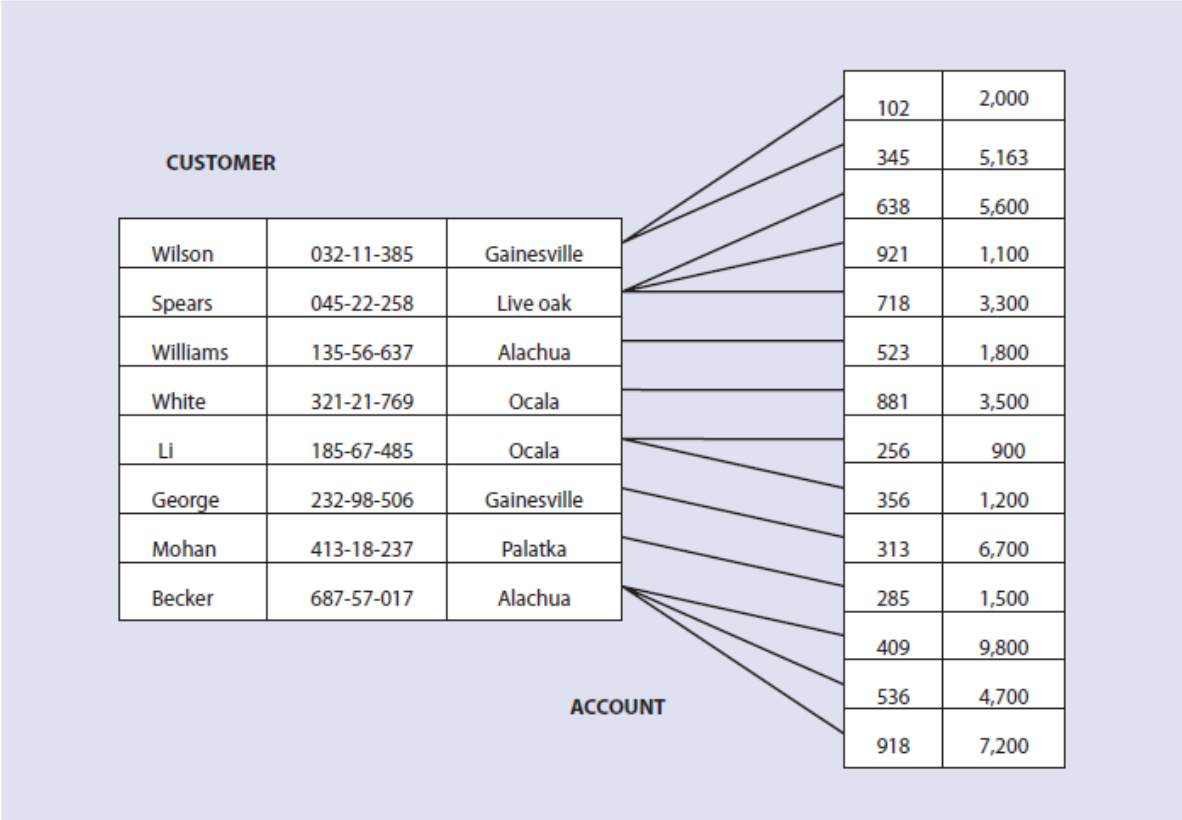


Figure 3.7 The relationship set between the CUSTOMER and ACCOUNT entities.

Degree of a Relationship

The number of entity sets that participate in a relationship is called the ***degree of relationship***. For example, the degree of the relationship featured in Figure 3.8 is two because CUSTOMER and ACCOUNT are two separate entity types that participate in the relationship. The three most common degrees of a relationship in a database are unary (degree 1), binary (degree 2), and ternary (degree 3). We will briefly define these degrees and then explore each kind of relationship in detail in subsequent sections.

Let E_1, E_2, . . . , E_n denote n entity sets and let R be the relationship. The degree of the relationship can also be expressed as follows:

Unary Relationship A unary relationship R is an association between two instances of the same entity type (i.e., $R \in E_1 \times E_1$). For example, two students are roommates and stay together in an apartment. Because they share the same address, a unary relationship exists between them for the attribute *Address* in Figure 3.3.

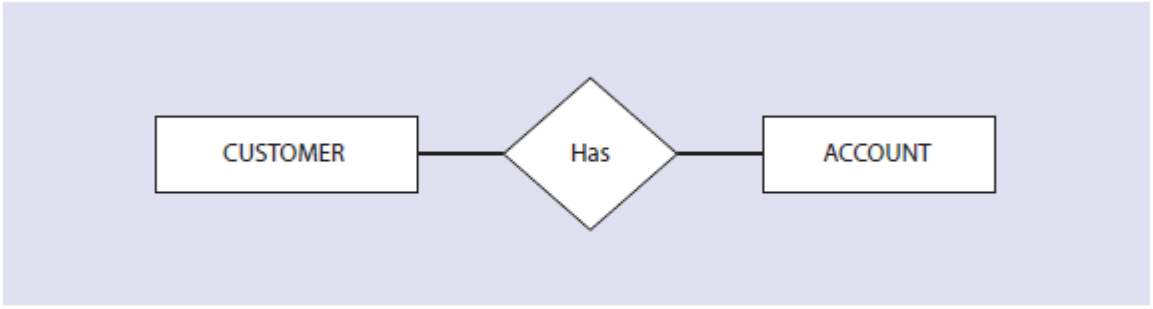


Figure 3.8 The relationship between CUSTOMER and ACCOUNT entities in an E-R diagram.

Binary Relationship A binary relationship R is an association between two instances of two different entity types (i.e., $R \in E_1 \times E_2$). For example, in a university, a binary relationship exists between a student (STUDENT entity) and an instructor (FACULTY entity) of a single class; an instructor *teaches* a student.

Ternary Relationship A ternary relationship R is an association between three instances of three different entity types (i.e., $R \in E_1 \times E_2 \times E_3$). For example, consider a student using certain equipment for a project. In this case, the STUDENT, PROJECT, and EQUIPMENT entity types relate to each other with ternary relationships: a student *checks out* equipment for a project.

Cardinality of a Relationship

The term *cardinal number* refers to the number used in counting. An *ordinal number*, by contrast, emphasizes the order of a number (1st, 7th, etc.). When we say cardinality of a relationship, we mean the ability to count the number of entities involved in that relationship. For example, if the entity types A and B are connected by a relationship, then the ***maximum cardinality*** represents the maximum number of instances of entity B that can be associated with any instance of entity A.

However, we don't need to assign a number value for every level of connection in a relationship. In fact, the term *maximum cardinality* refers to only two possible values: one or many. While this may seem to be too simple, the division between one and many allows us to categorize all of the permutations possible in any relationship. The maximum cardinality value of a relationship, then, allows us to define the four types of relationships possible between entity types A and B. Figure 3.9 illustrates these types of relationships.

One-to-One Relationship In a one-to-one relationship, at most one instance of entity B can be associated with a given instance of entity A and vice versa.

One-to-Many Relationship In a one-to-many relationship, many instances of entity B can be associated with a given instance of entity A. However, only one instance of entity A can be associated with a given instance of entity B. For example, while a customer of a company can make many orders, an order can only be related to a single customer.

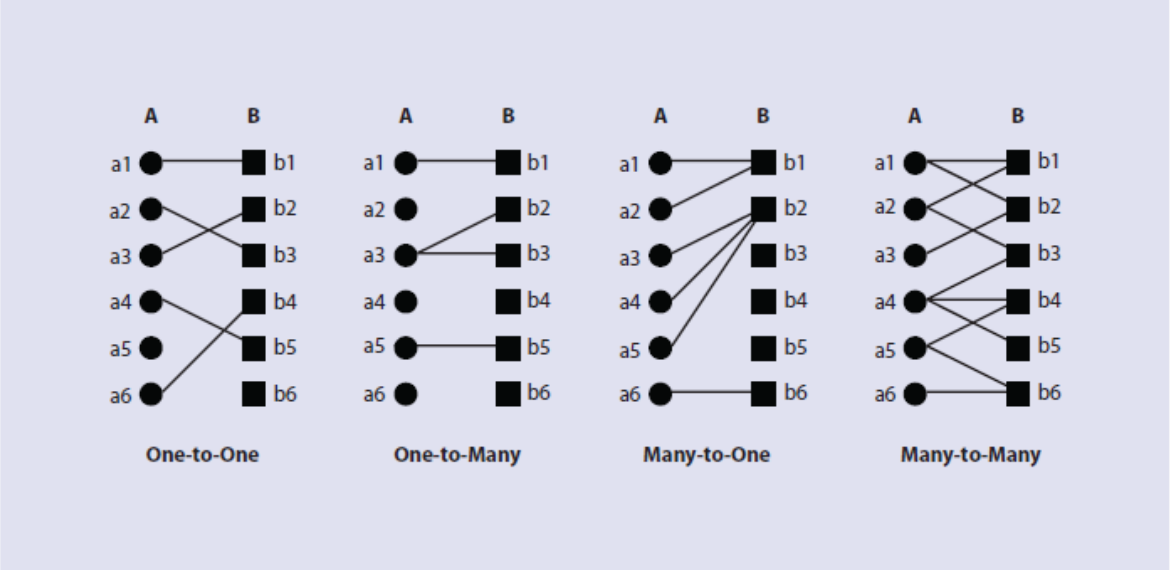


Figure 3.9 The four types of relationships between entity types A and B.

Many-to-Many Relationship In a many-to-many relationship, many instances of entity A can be associated with a given instance of entity B, and, likewise, many instances of entity B can be associated with a given instance of entity A. For example, a machine may have different parts, while each individual part may be used in different machines.

Representing Relationship Types Figure 3.10 displays how we represent different relationship types in an E-R diagram. An entity on the *one* side of the relationship is represented by a vertical line, "I," which intersects the line connecting the entity and the relationship. Entities on the *many* side of a relationship are designated by a crowfoot as depicted in Figure 3.10.

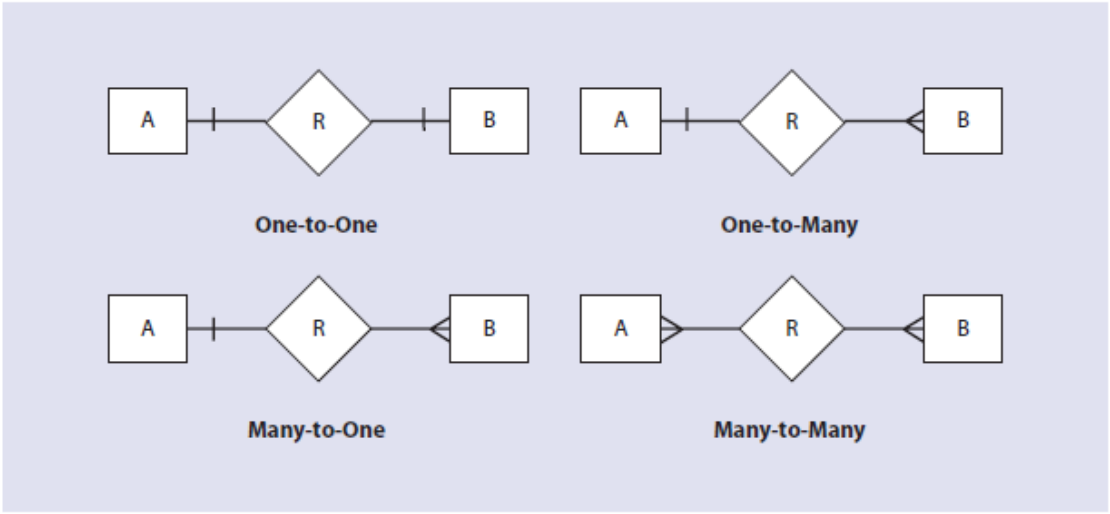


Figure 3.10 The relationship types based on maximum cardinality.

We will now discuss the minimum cardinality of a relationship. The ***minimum cardinality*** between two entity types A and B is defined as the minimum number of instances of entity B that must be associated with each instance of entity A. In an E-R diagram, we allow the minimum cardinality to take two values: zero or one. If the minimum cardinality is zero, we say that entity type B is an *optional* participant in the relationship; otherwise, it is a *mandatory* participant. An optional relationship is represented by an "O" and mandatory relationship is represented by "|" in an E-R diagram.

Figure 3.11 shows the four possibilities of the minimum cardinality of a relationship between two entity types A and B. Figure 3.11(a) depicts a situation in which no minimum cardinality constraints exist between the instances of entities A and B, meaning both entities A and B are optional participants in the relationship. Figure 3.11(b) illustrates a situation in which each instance of entity B must be associated with at least one instance of entity A, but no association is required for an instance of entity A. Figure 3.11(c) illustrates a situation in which each instance of entity A must be associated with at least one instance of entity B, but no association is required for an instance of entity B. Finally, Figure 3.11(d) illustrates a situation in which each instance of entity A and B must be associated with at least one instance of entity B and A, respectively.

An E-R diagram displays both the maximum and the minimum cardinalities of the relationships between two entities. Since there are four basic possibilities of maximum cardinalities and four possibilities of minimum cardinalities between two entities, there are 16 types of relationships possible between two entities in terms of cardinality. We will see several examples of these relationships while studying unary, binary, and ternary relationships.

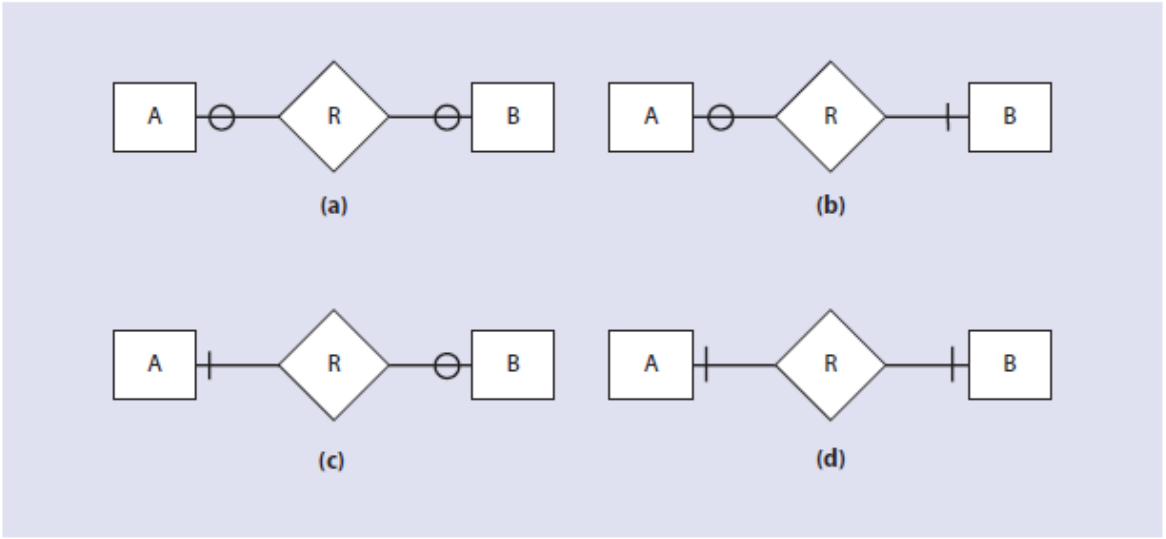
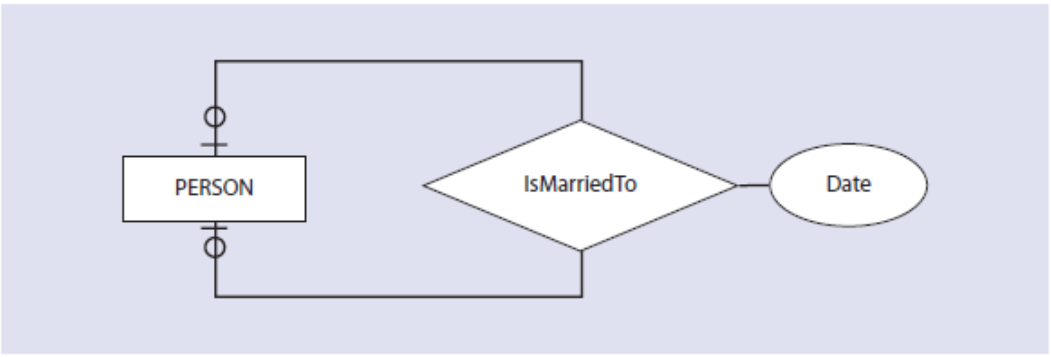


Figure 3.11 The relationship types based on minimum cardinality.

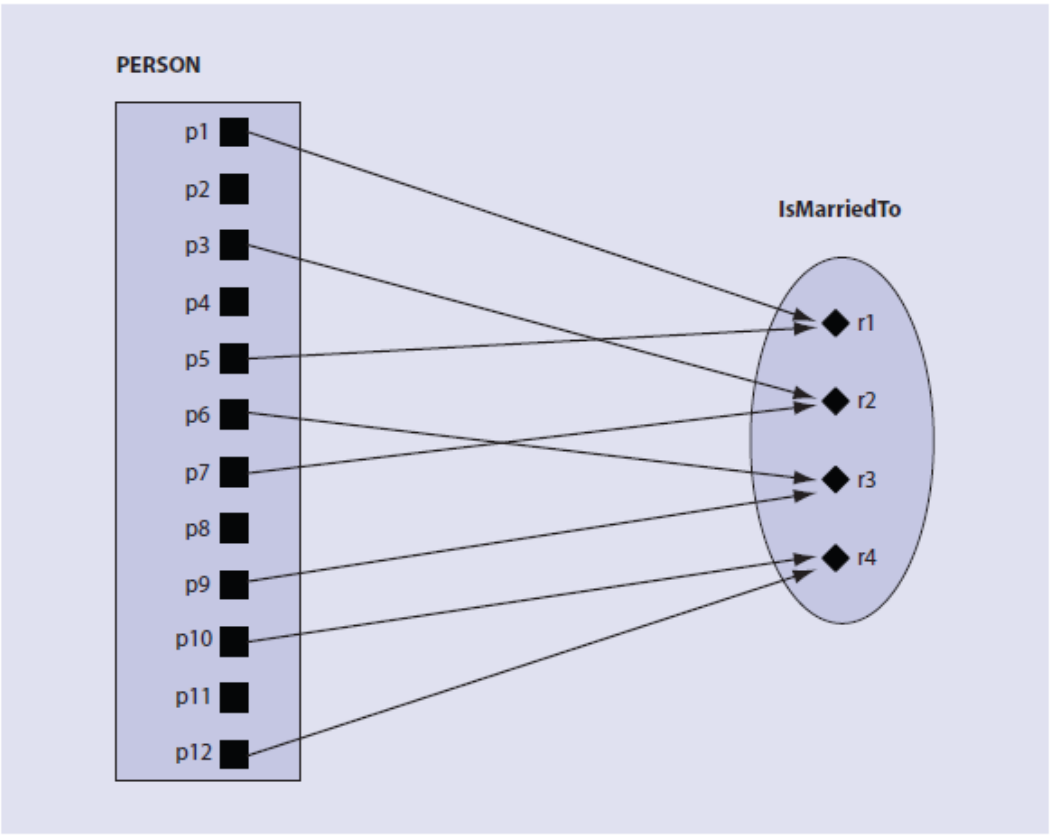
Unary Relationship

As we mentioned in Section 3.5, a unary relationship ($R \in E_1 \times E_1$) is an association between two entities of the same entity type. Figure 3.12(a) displays the E-R diagram of a unary relationship *IsMarriedTo*. Whenever two people in the entity type PERSON get married, the relationship instance *IsMarriedTo* is created. The *Date* of marriage is an attribute of this relationship. Since a person can only be married to one other person, marriage is a one-to-one relationship. Furthermore, since a person can be unmarried, the minimum cardinality of the *IsMarriedTo* relationship is zero.

Figure 3.12(b) depicts several relationship instances of this relationship type. Each relationship instance (r1, r2, r3, and r4) connects two instances in PERSON. The lines allow us to read relationships between entity instances. For example, r1 suggests that person p1 is married to person p5, and so forth.

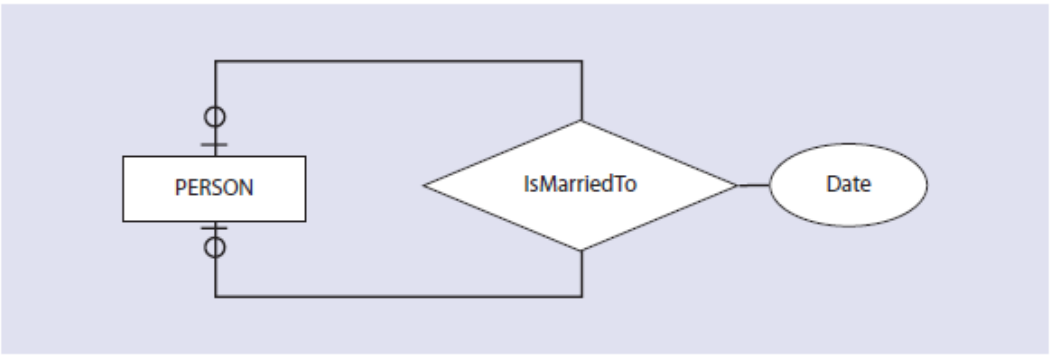


(a)

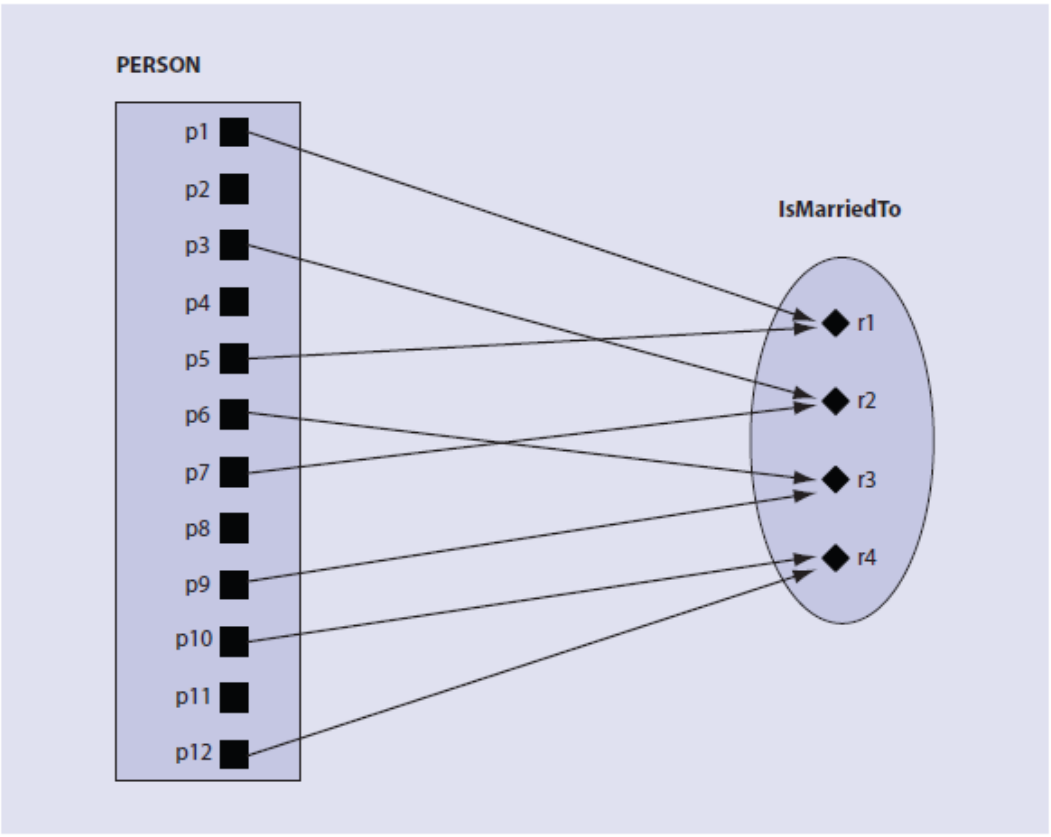


(b)

Figure 3.12 The unary one-to-one relationship.



(a)

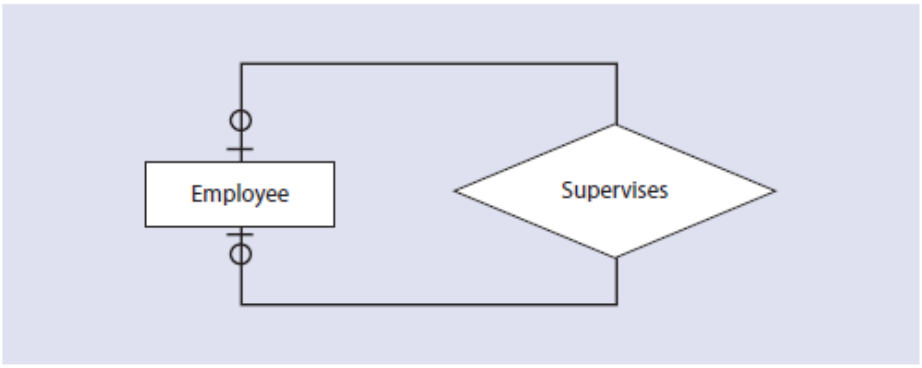


(b)

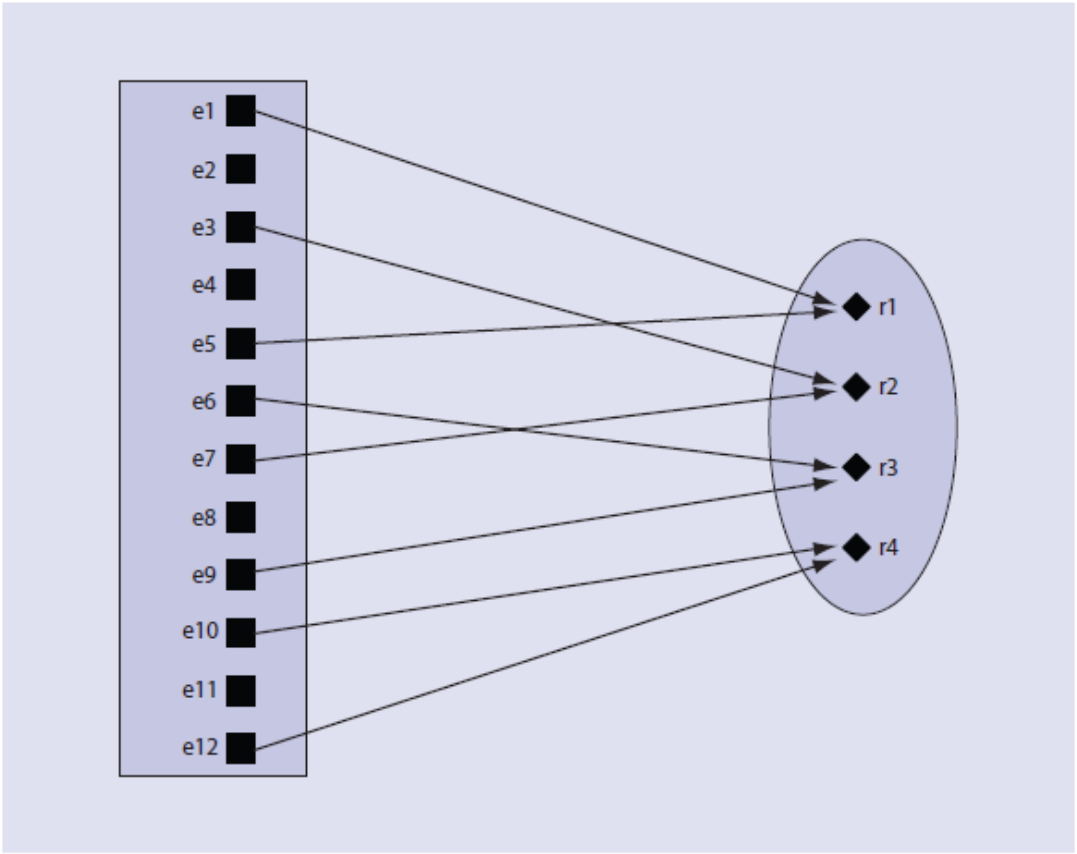
Figure 3.12 The unary one-to-one relationship.

Figure 3.13(a) illustrates another example of a unary relationship, expressed by the relationship instance *Supervises*. This relationship instance exists whenever an employee supervises another employee. The relationship *Supervises* is a one-to-many relationship since an employer can supervise many employees but a supervisee can have only one supervisor. The minimum cardinality for supervising is zero (an employee may not supervise anyone) but the minimum cardinality of being supervised is one (every employee must be supervised).

Figure 3.13(b) shows several relationship instances of this relationship type. Each relationship instance (`r1`, `r2`, `r3`, and `r4`) connects two entity instances in EMPLOYEE; these are the supervisor instance and the supervisee instance. For example, `r2` suggests that employee `e3` supervises employee `e7`, and so forth.



(a)



(b)

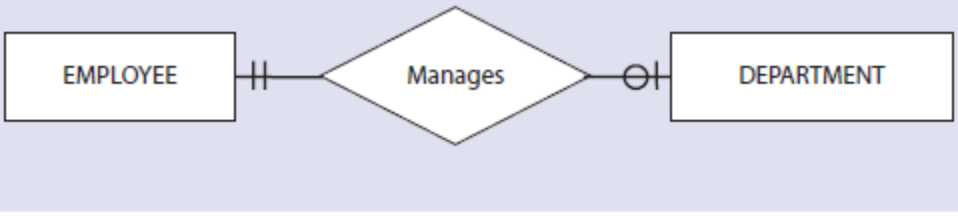
Figure 3.13 The unary one-to-many relationship.

Binary Relationship

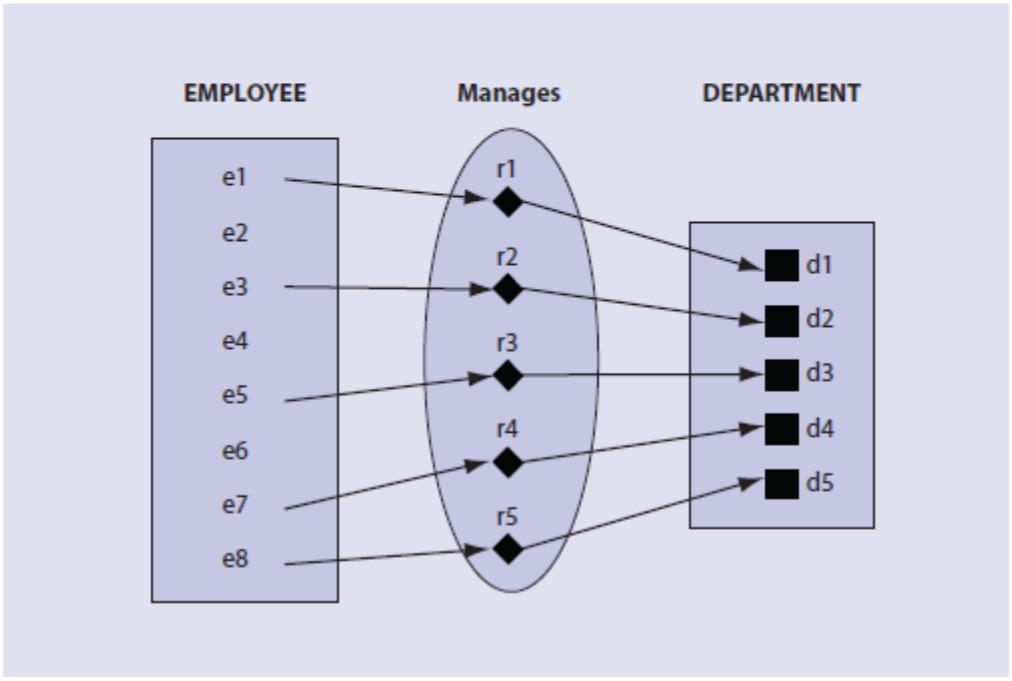
A binary relationship, or an association between two entity types, is the most common form of a relationship expressed by an E-R diagram. Recall that a binary relationship is $R \in E_1 \times E_2$ in which E_1 and E_2 are two different entity types. The examples of binary relationships with the relationship instances are presented in Figure 3.15. Notice that each relationship instance obeys a basic characteristic of the binary relationships; in other words, each relationship instance is connected to exactly two entity instances of different entity types.

By applying what we have learned earlier in this chapter, we should be able to determine the cardinalities of these relationships quite easily. The relationship in Figure 3.15(a) is a one-to-one relationship since we assume that an employee can manage at most one department and each department is managed by at most one employee. The minimum cardinality can be determined

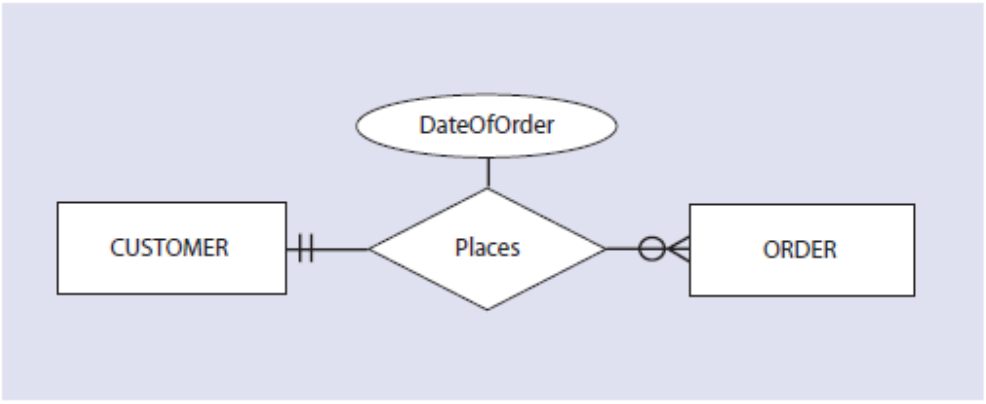
(a)



(b)

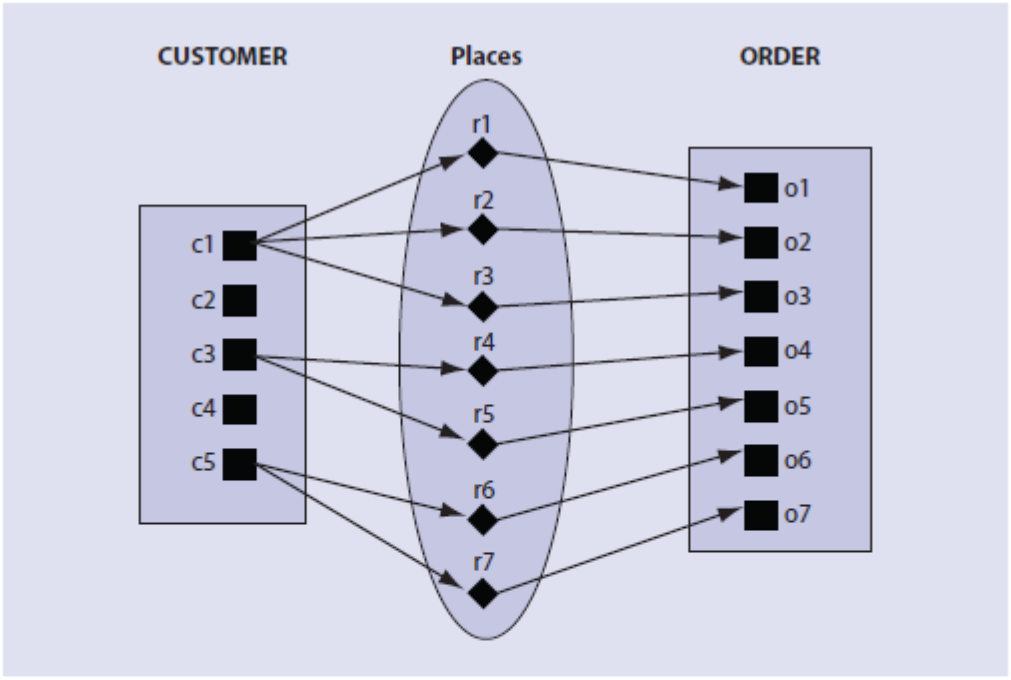


(c)

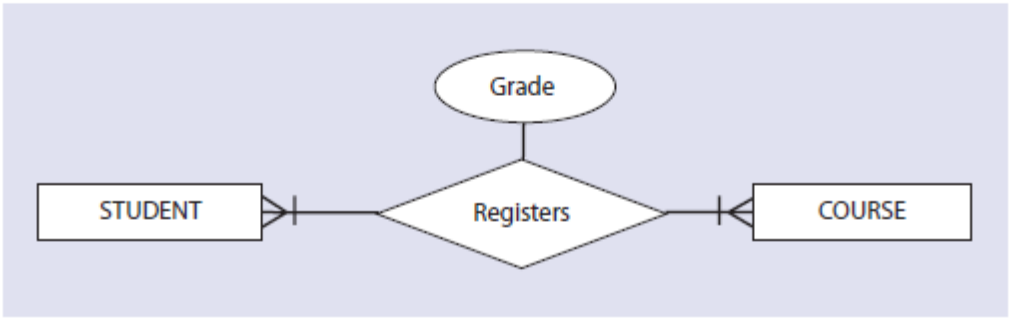


(continues)

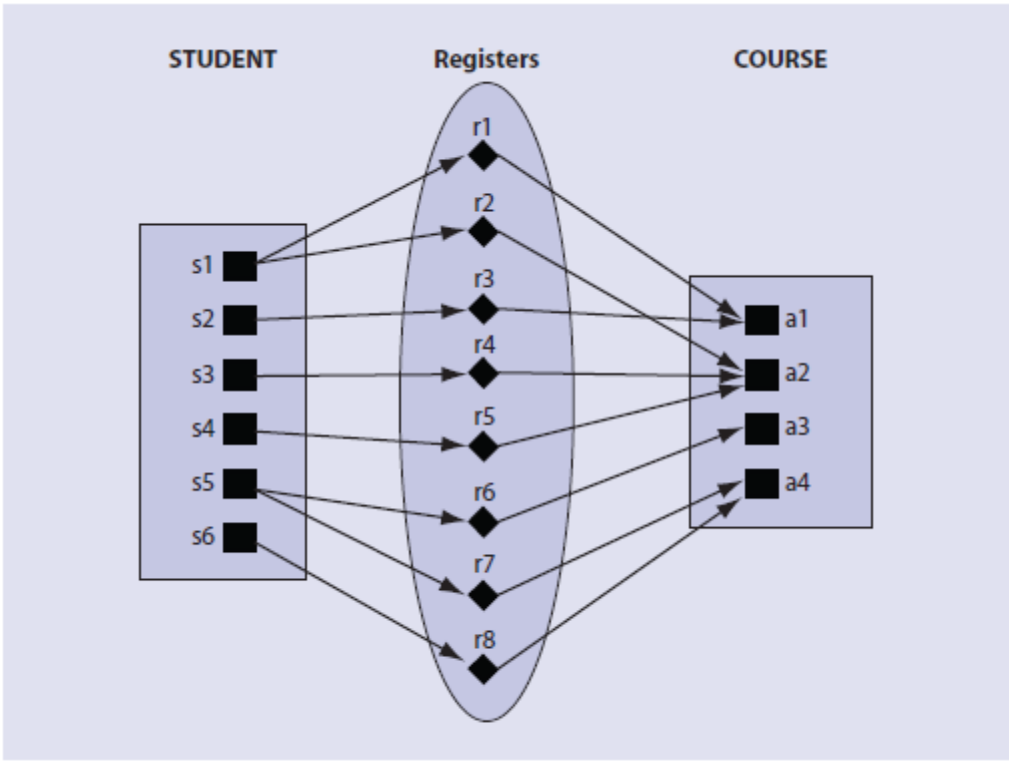
Figure 3.15 (a) and (b): Binary one-to-one relationships; (c) and (d): Binary one-to-many relationships; (e) and (f): Binary many-many relationships.


CUSTOMER
Places
ORDER
r1
r2
r3
r4
r5
r6
r7
c1
c2
c3
c4
c5
o1
o2
o3
o4
o5
o6
o7

(d)


Grade
STUDENT
Registers
COURSE

(e)



(f)

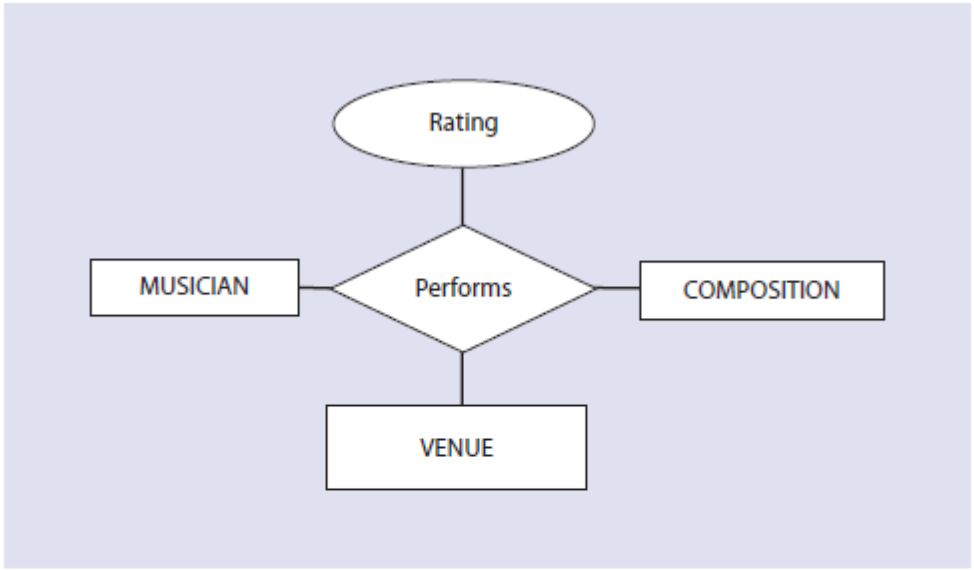
Figure 3.15 (*continued*)

since each department must be managed by an employee, but not all employees manage departments. For example, while some of the employees in Figure 3.15(b), such as e2, e4, and e6, do not manage a department, every department is managed by an employee.

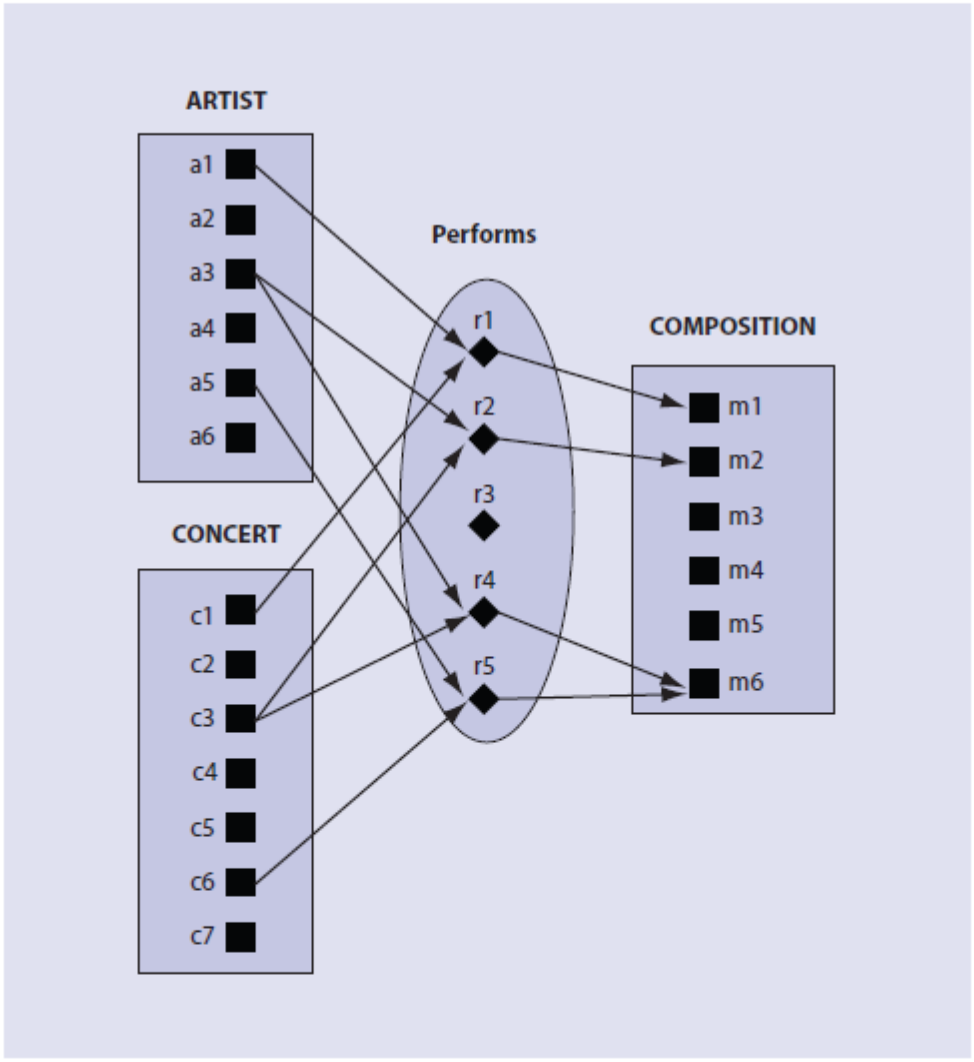
The binary one-to-many relationship represented in Figure 3.15(c) features a slightly different arrangement. While a customer can place several orders or may choose not to order at all, each order must be placed by exactly one customer. Figure 3.15(d) shows us that, while each order is made by a single customer, not all customers place orders. However, Figure 3.15(d) differs from Figure 3.15(b) in that a single customer may place any number of orders; while the first customer places three orders, and the fifth customer places only two, both transactions represent a maximum cardinality of many.

Finally, Figure 3.15(e) and (f) illustrates a many-to-many relationship, featuring a minimum cardinality of zero and a maximum cardinality of many. In other words, each student can participate in many activities, a single activity, or no activity at all. Conversely, any number of students, from many to none, can participate in a given activity.

Ternary Relationships



(a)



(b)

Figure 3.16 Examples of ternary relationships.

Attributes of Relationships

We have already discussed examples of E-R diagrams that reveal attributes stemming off relationships. Recall Figure 3.15(c), for example. In that E-R diagram, the attribute *DateofOrder* collects data for the relationship CUSTOMER *places* ORDER. Attributes on relationships are like attributes on entity types we have seen so far. An attribute on a relationship stores information related to the relationship. In Figure 3.17, the attribute *Quantity* stores the number of components that make up an entity type ITEM. Note that the attribute *Quantity* stems from the relationship and not from the entity.

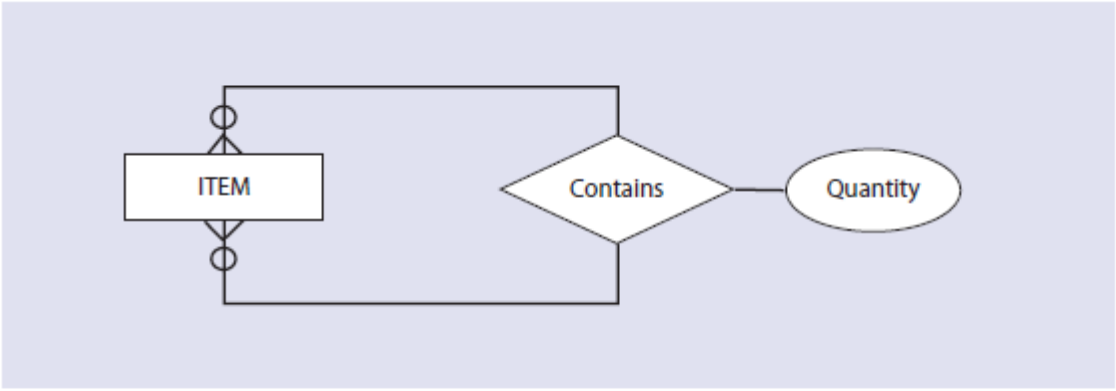


Figure 3.17 An example of attributes of relationships.

Associative Entities

An ***associative entity*** is an entity type that connects the instances of one or more entity types and contains attributes particular to this association. Basically, an associative entity is a relationship that has been turned into an entity because it meets one of the following conditions:

1. It is a many-to-many binary relationship
2. It is a ternary relationship or a relationship of an even higher degree

The associative entity in an E-R diagram is represented by an entity box enclosing the diamond relationship symbol (see Figure 3.18). This symbol demonstrates that the entity is generated from a relationship.

Consider, for example, the E-R diagram of the binary relationship illustrated in Figure 3.18(a). When we convert this relationship into an associated entity, we get the E-R diagram in Figure 3.18(b). In the example, the relationship *Participates* is converted into an associated entity, ENROLLMENT. If the relationship has attributes, they become the attributes of the corresponding associative entity. *DateJoined* is an example of an attribute of a relationship turned into an attribute of an associative entity in Figure 3.18(b).

Furthermore, recall that every entity in an E-R diagram must have an identifier. The identifiers of two original entities together serve as a composite identifier of the associative entity. In our example, the identifier of the entity ENROLLMENT is a composite attribute comprised of the identifiers of the STUDENT and ACTIVITY entities. Since a student can enroll in every activity but can only enroll once in any given activity, the composite attribute {*StudentID, ActivityID*} is a valid identifier for the associated entity ENROLLMENT.

Now that ENROLLMENT is a new entity in Figure 3.18(b), note that there is no relationship diamond on the line between the entity STUDENT and the associative entity ENROLLMENT because the associative entity represents a relationship. Furthermore, the E-R diagram in Figure 3.18 (b) depicts the cardinalities. Each instance of the STUDENT entity is related to several instances of the ENROLLMENT entity; in fact, each instance of the STUDENT entity is related to as many instances of the ENROLLMENT entity as the number of activities in which the student participates. Therefore, there is a one-to-many relationship between the STUDENT entity type and the ENROLLMENT entity type. Similarly, there is a one-to-many relationship between the entity types ACTIVITY and ENROLLMENT because several students may enroll in one activity.

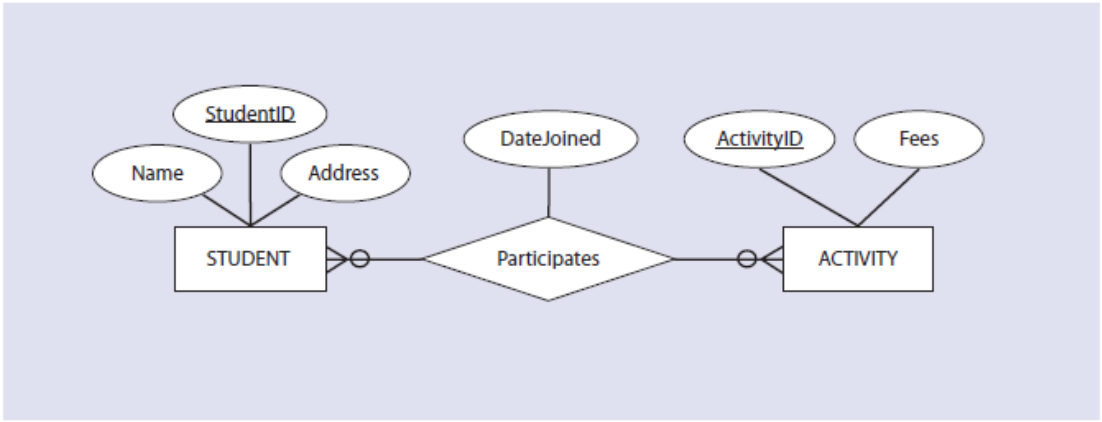


Figure 3.18 (a) A many-to-many binary relationship; (b) A many-to-many binary relationship converted to an associated entity.

Weak Entity Types

Entity types can be classified into two categories: *strong entity types* and *weak entity types*. A ***strong entity type*** exists independent of other entity types, while a ***weak entity type*** depends on another entity type. Consider a student database that includes an entity STUDENT. Suppose that we also record data about each student's dependents, such as a spouse or children, in this database. To do so, we must create the entity type DEPENDENT. The entity type DEPENDENT does not exist on its own and owes its existence to the STUDENT entity type. When students graduate and their records are removed from the STUDENT entity set, the records of their dependents are also removed from the DEPENDENT entity set. In the E-R diagram, a weak entity is indicated by a double-lined rectangle. The corresponding relationship diamond is also double-lined.

The entity type on which a weak entity type depends is called the *identifying owner* (or simply owner), and the relationship between a weak entity type and its owner is called an *identifying relationship*. In Figure 3.20, STUDENT is the owner and *Has* is the identifying relationship. For a weak entity, we define a *partial key attribute* that, when combined with the key attribute of its owner, provides the full identifier for the weak entity. In Figure 3.20, for example, *DependentName* is the partial identifying attribute. When we combine it with the *StudentID*, it uniquely identifies the dependent. Of course, in this example, we make an implicit assumption that people do not give the same name to more than one of their children.

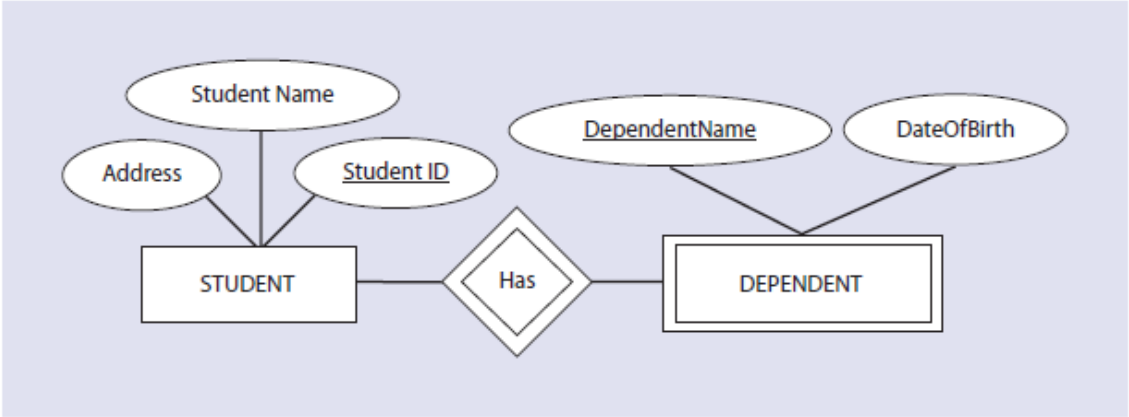


Figure 3.20 The weak entity in an E-R diagram.

Attribute Inheritance and Subclass Relationships

Attribute Inheritance is the property by which subclass entities inherit attributes of the superclass. The subclass entity type is an entity type in itself with a set of attributes and relationships. In addition to its own attributes, a subclass entity inherits all the attributes of its superclass. For example, the STUDENT subclass has attributes such as *MajorDept*, *Class*, and *GPA*. It also in-

herits all the attributes of the PERSON entity type. In other words, if Chris Alto is an instance of the STUDENT subclass, then he is necessarily an instance of the PERSON superclass as well. Chris Alto is a value for the *Name* attribute for the entity PERSON, and the STUDENT subclass inherits it from the PERSON. Note that Chris's 4.0 *GPA* is an attribute of the STUDENT subclass only. Thus, the identifier of a superclass is also inherited by all its subclasses and serves as an identifier for subclasses. Subclasses don't need their own identifier. Thus, the relationship between a superclass and subclass is always a one-to-one relationship (notations are usually ignored in an E-R diagram).

Although, each subclass also possesses attributes of its superclass, each subclass entity expresses a distinct role in an E-R diagram. It can have its own attributes and relationships that distinguish one subclass from another. Figure 3.24 depicts relationships for the STUDENT and FACULTY subclasses.

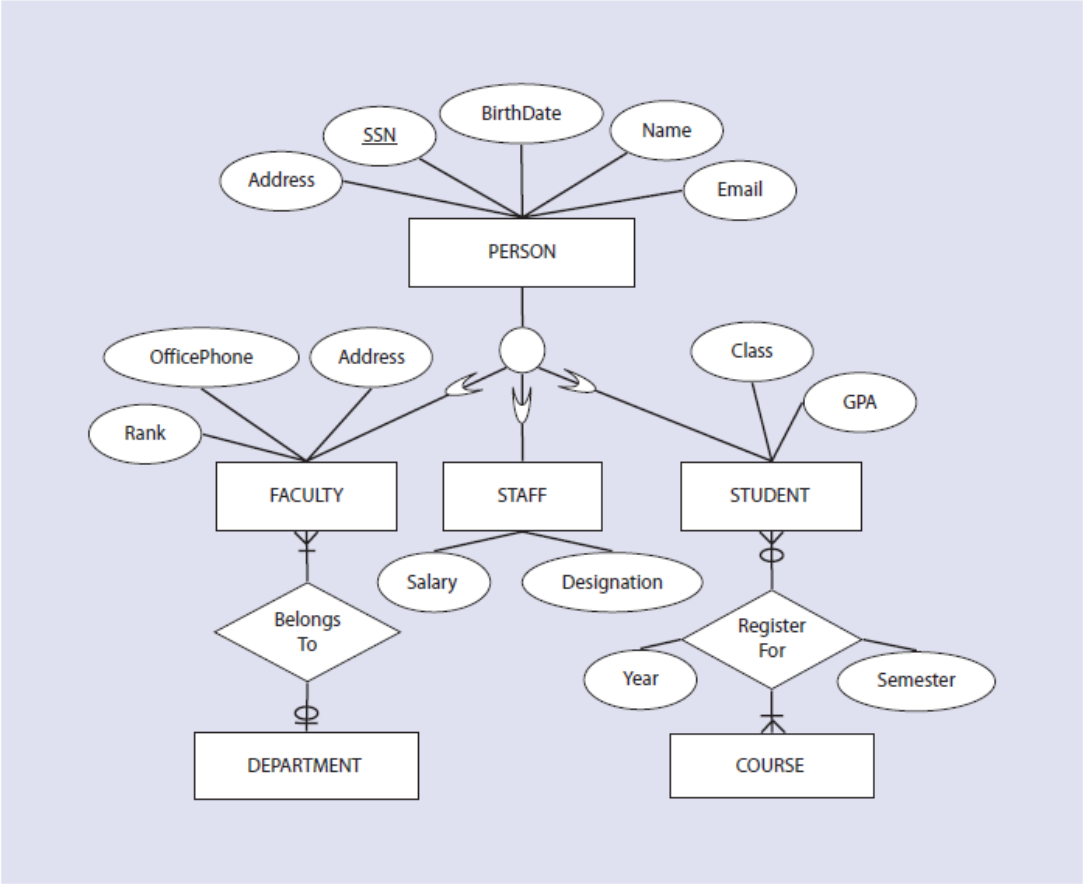


Figure 3.24 An example of the superclass/subclass relationships.

Generalization and Specialization Process

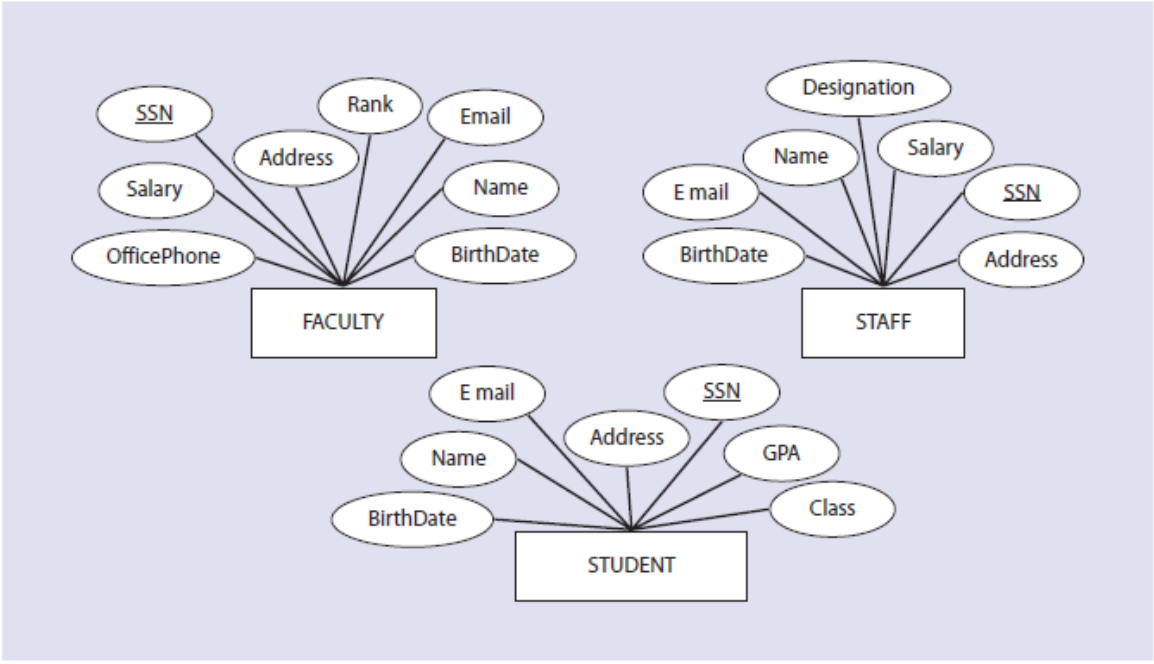
Generalization

Generalization is the process of defining general entity types from a set of specialized entity types by identifying their common characteristics. In other words, this process minimizes the differences between entities by *identifying a general entity type* that features the common attributes of specialized entities. Generalization is a bottom-up approach as it starts with the specialized entity types (subclasses) and forms a generalized entity type (superclass).

For example, suppose that someone has given us the specialized entity types FACULTY, STAFF, and STUDENT, and we want to represent these entity types separately in the E-R model as depicted in Figure 3.25(a). However, if we examine them closely, we can observe that a number of attributes are common to all entity types, while others are specific to a particular entity. For example, FACULTY, STAFF, and STUDENT all share the attributes *Name*, *SSN*, *Birth Date*, *Address*, and *Email*. On the other hand, attributes such as *GPA*, *Class*, and *MajorDept* are specific to the STUDENTS; *OfficePhone* is specific to FACULTY, and *Designation* is specific to STAFF. Common attributes suggest that each of these three entity types is a form of a more general entity type. This general entity type is simply a PERSON superclass entity with common attributes of three subclasses (see Figure 3.25(b)).

Thus, in the generalization process, we group specialized entity types to form one general entity type and identify common attributes of specialized entities as attributes of a general entity type. The general entity type is a superclass of specialized entity types or subclasses.

(a)



(*continues*)

Figure 3.25 (a) STAFF, FACULTY, and STUDENT entities before generalization; (b) PERSON superclass and FACULTY, STAFF, and STUDENT subclasses after generalization.

(b)

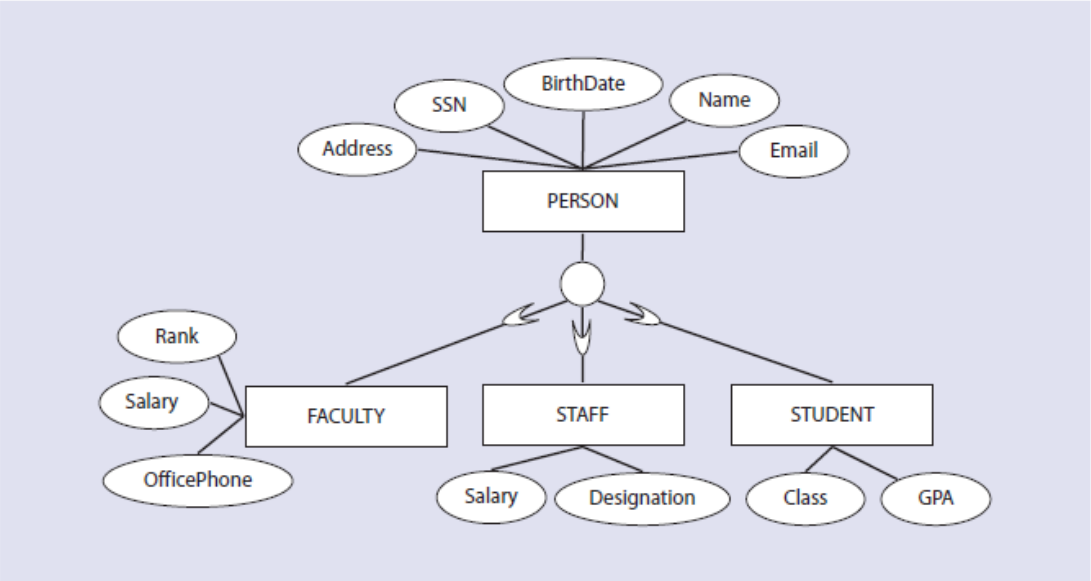


Figure 3.25 (*continued*)

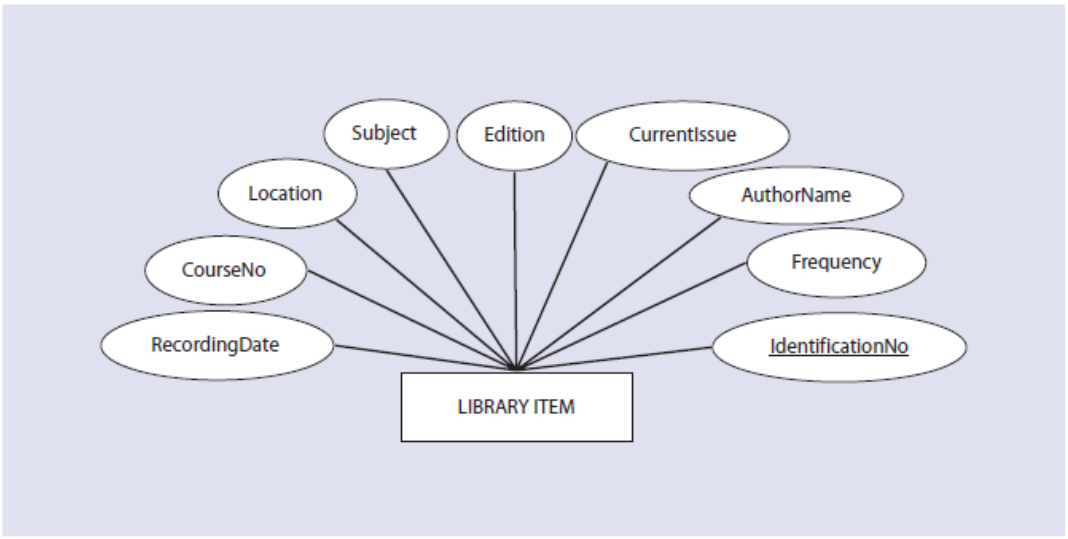
Specialization

Specialization is the process of defining one or more subclasses of a superclass by identifying its distinguishing characteristics. Unlike generalization, specialization is thus a top-down approach. It starts with the general entity (superclass) and forms specialized entity types (subclasses) based on specialized attributes or relationships specific to a subclass.

For example, consider Figure 3.26(a). LIBRARY ITEM is an entity type with several attributes such as *IdentificationNo*, *RecordingDate*, *Frequency*, and *Edition*. After careful review of these items, it should become clear that some items such as books do not have values for attributes such as *Frequency*, *RecordingDate*, and *CourseNo*, while Video CDs do not have an *Author* or an *Edition*.

In addition, all items have common attributes such as *IdentificationNo*, *Location*, and *Subject*. Someone creating a library database, then, could use the specialization process to identify superclass and subclass relationships. In this case, the original entity LIBRARY ITEM forms a superclass entity type made up of attributes shared by all items, while specialized items with distinguishing attributes, such as BOOK, JOURNALS, and VIDEOCD, form subclasses.

(a)



(b)

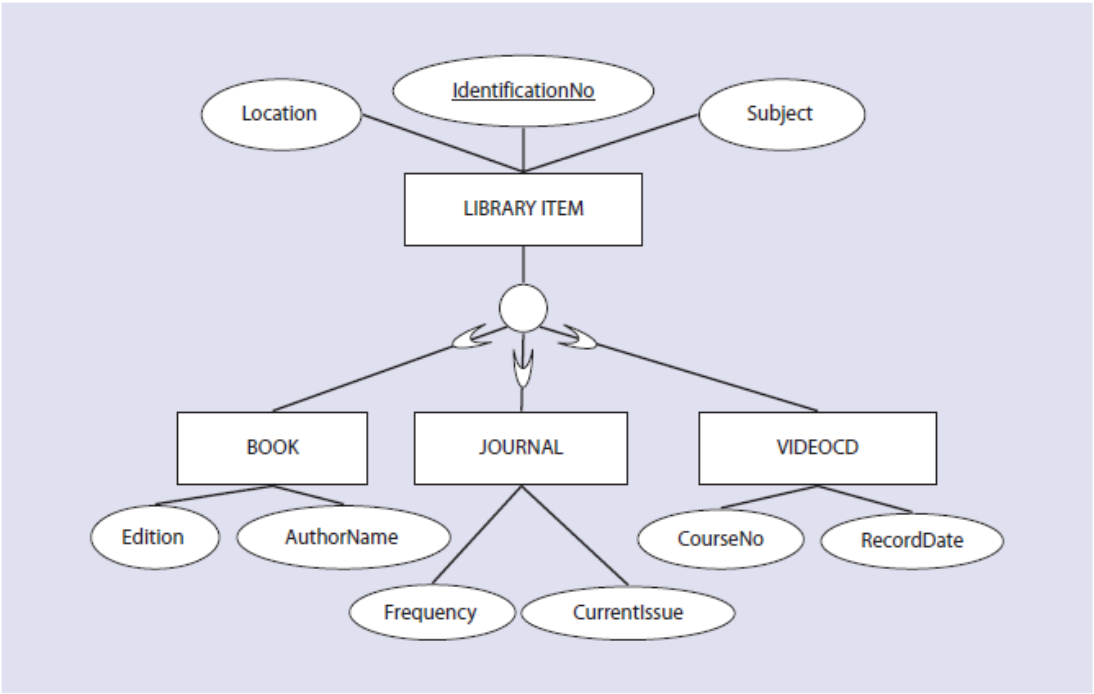


Figure 3.26 (a) LIBRARY ITEM entity before specialization; (b) LIBRARY ITEM superclass and BOOK, JOURNAL, and VIDEOCD subclasses after specialization.

Total Participation Rule In total participation, membership is mandatory. Each instance of a superclass must be an instance of at least one subclass. For example, consider Figure 3.27.

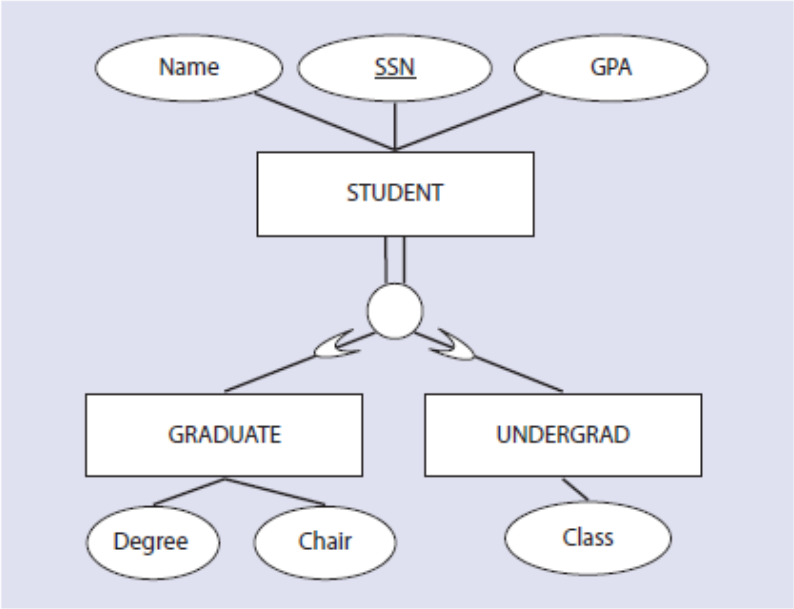


Figure 3.27 An example of the total participation rule.

Every instance of the superclass STUDENT must be an instance of either the GRADUATE student or UNDERGRAD student subclass. That is, if John Doe is an instance of a STUDENT, then John must be a graduate or undergraduate student. However, whether John Doe belongs to the graduate, undergraduate, or both entity types is answered by the disjoint rule, which we will define in a moment. We use a double line between the superclass entity type and the circle to represent the total participation.

Partial Participation Rule Membership is optional in a partial participation. An instance of a superclass does not have to be an instance of any of the subclasses. For example, consider Figure 3.26. An instance of the LIBRARY ITEM superclass can be a member of BOOK, VIDEO CD, or JOURNALS; however it is not mandatory for an instance to belong to any of these subclasses. If the library item *Newspaper* is an instance of a superclass, it does not have to be included in one of the subclasses; it can stay at the superclass level without having values for any subclass attributes. We use a single line between the superclass entity type and the circle (the default notation) to represent partial participation.

Definition

An **entity** is an object that exists and that is distinguishable from other objects.

An **attribute** is a property or characteristic of an entity type that is of interest to an organization.

A **relationship** is an association among several entities.

Definition

The number (unary, binary, or ternary) of entity sets that participate in a relationship is called the **degree of relationship**.

The **cardinality of relationship** represents the minimum/maximum number of instances of entity B that must/can be associated with any instance of entity A.

Definition

An **associative entity** is an entity type that connects the instances of one or more entity types and contains attributes particular to this association.

A **strong entity type** exists independent of other entity types, while a **weak entity type** depends on another entity type.

Definition

Attribute inheritance is the property by which subclass entities inherit values for all attributes of the superclass.

Generalization is the process of defining a general entity type from a set of specialized entity types by identifying their common characteristics.

Specialization is a process of defining one or more subclasses of a superclass by identifying their distinguishing characteristics.

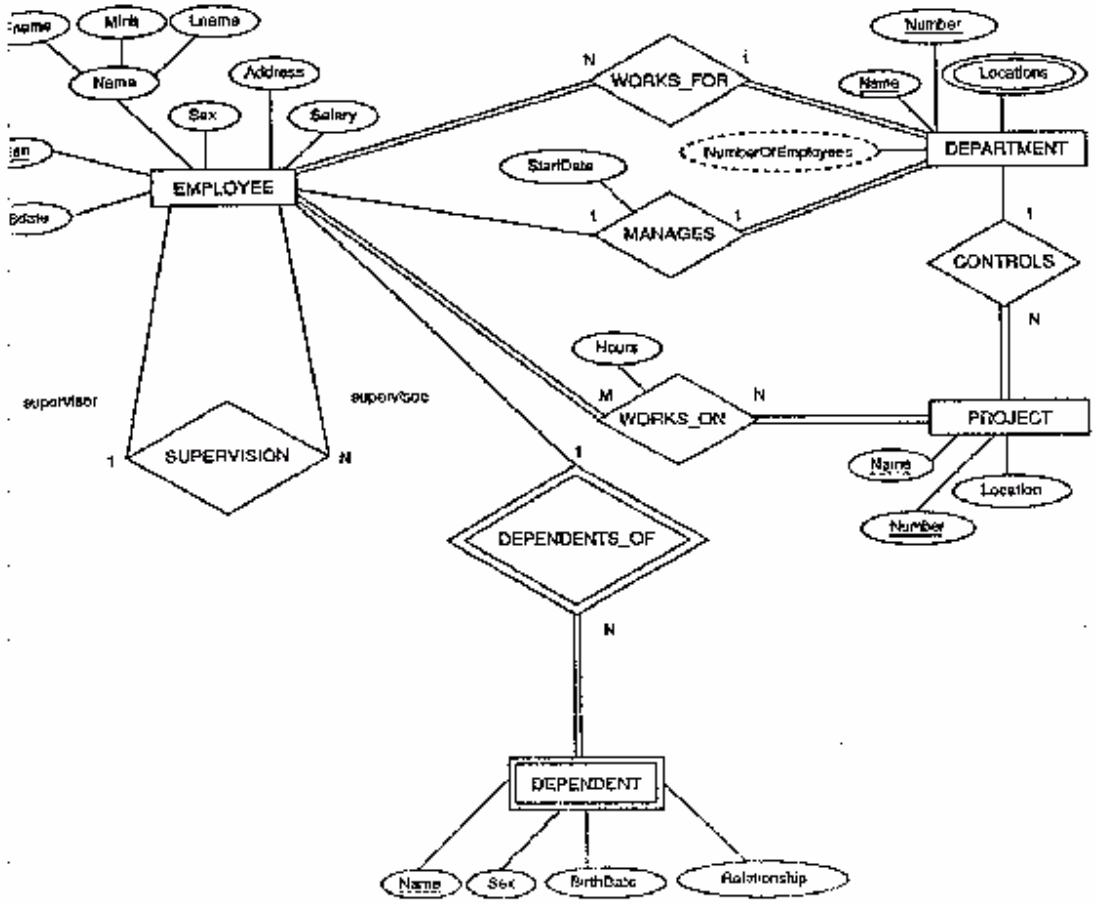
Definition

Participation constraints dictate whether each instance (member) of a superclass must participate as an instance (member) of a subclass.

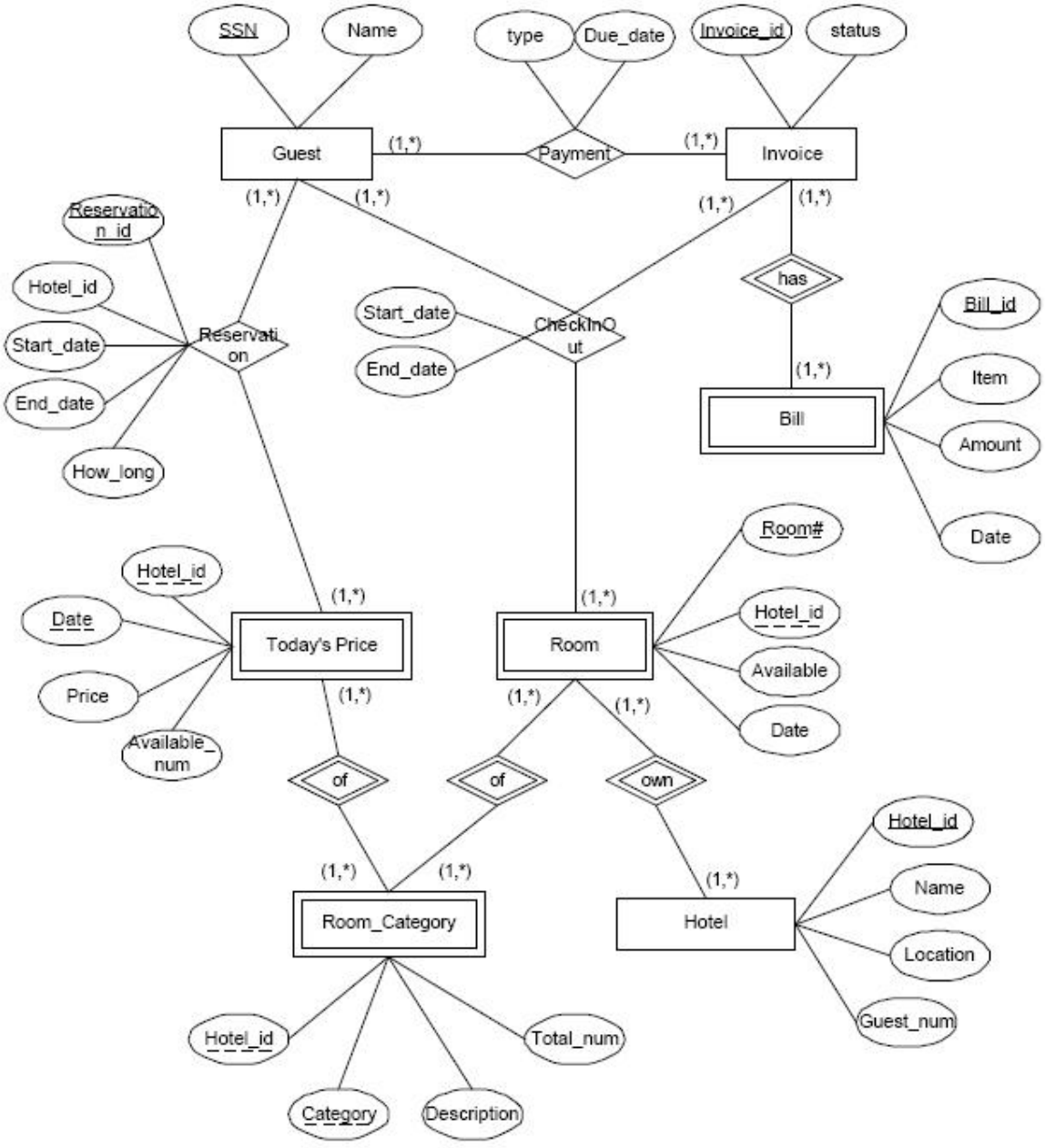
Disjoint constraints define whether it is possible for an instance of a superclass to simultaneously be a member of one or more subclasses.

ER Diagrams

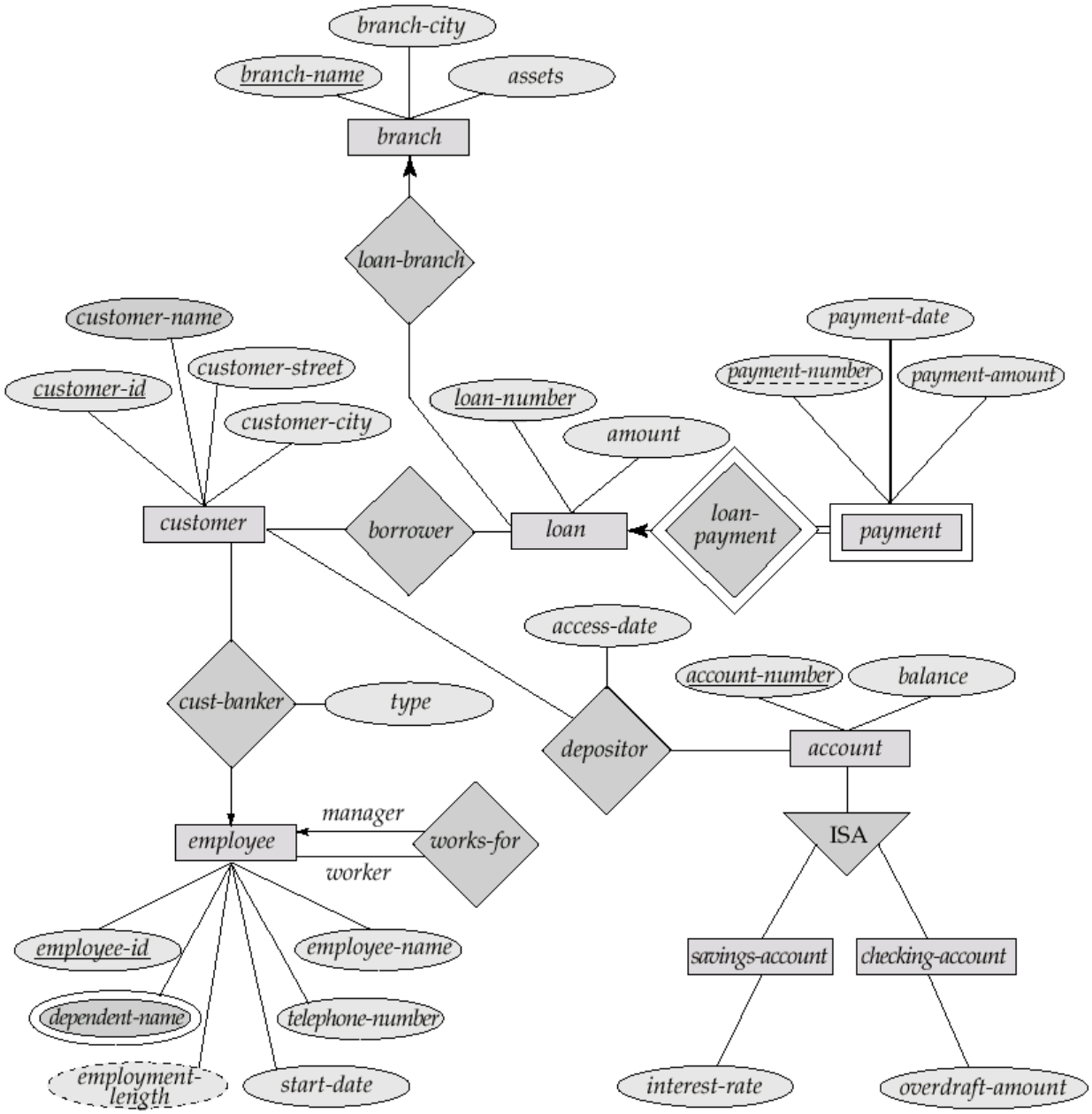
Example 1: Company system



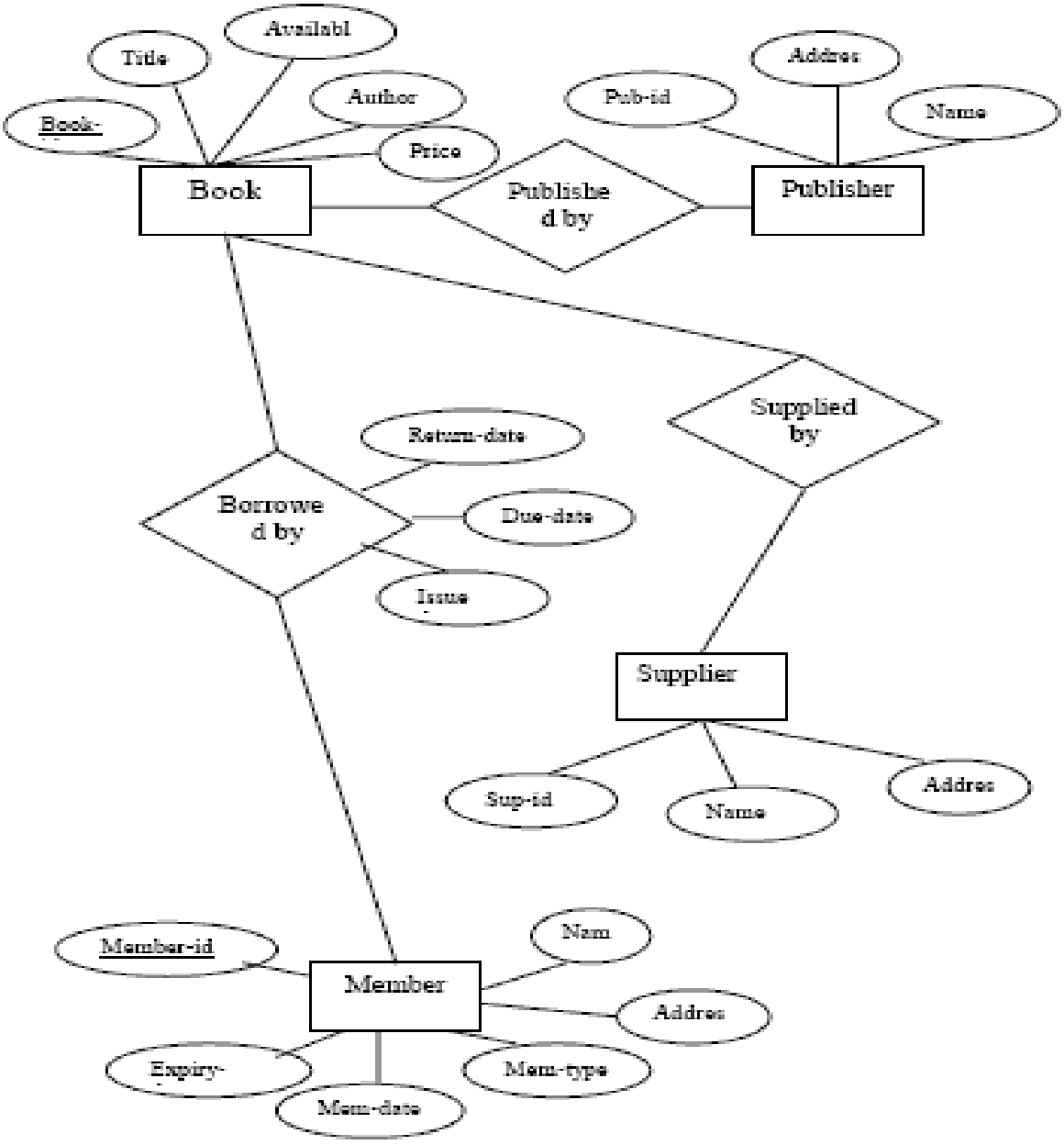
Hotel Management



ER diagram for Banking system



Library Management



College management