

Overview of Database Management System

Learning Objectives. This chapter provides an overview of database management system which includes concepts related to data, database, and database management system. After completing this chapter the reader should be familiar with the following concepts:

- Data, information, database, database management system
- Need and evolution of DBMS
- File management vs. database management system
- ANSI/SPARK data model
- Database architecture: two-, three-, and multitier architecture

1.1 Introduction

Science, business, education, economy, law, culture, all areas of human development “work” with the constant aid of data. Databases play a crucial role within science research: the body of scientific and technical data and information in the public domain is massive and factual data are fundamental to the progress of science. But the progress of science is not the only process affected by the way people use databases. Stock exchange data are absolutely necessary to any analyst; access to comprehensive databases of large scale is an everyday activity of a teacher, an educator, an academic or a lawyer. There are databases collecting all sorts of different data: nuclear structure and radioactive decay data for isotopes (the Evaluated Nuclear Structure Data File) and genes sequences (the Human Genome Database), prisoners’ DNA data (“DNA offender database”), names of people accused for drug offenses, telephone numbers, legal materials and many others. In this chapter, the basic idea about database management system, its evolution, its advantage over conventional file system, database system structure is discussed.

1.2 Data and Information

Data are raw facts that constitute building block of information. Data are the heart of the DBMS. It is to be noted that all the data will not convey useful information. Useful information is obtained from processed data. In other words, data has to be interpreted in order to obtain information. Good, timely, relevant information is the key to decision making. Good decision making is the key to organizational survival.

Data are a representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or automatic means. The data in DBMS can be broadly classified into two types, one is the collection of information needed by the organization and the other is “metadata” which is the information about the database. The term “metadata” will be discussed in detail later in this chapter.

Data are the most stable part of an organization’s information system. A company needs to save information about employees, departments, and salaries. These pieces of information are called data. Permanent storage of data are referred to as persistent data. Generally, we perform operations on data or data items to supply some information about an entity. For example library keeps a list of members, books, due dates, and fines.

1.3 Database

A database is a well-organized collection of data that are related in a meaningful way, which can be accessed in different logical orders. Database systems are systems in which the interpretation and storage of information are of primary importance. The database should contain all the data needed by the organization as a result, a huge volume of data, the need for long-term storage of the data, and access of the data by a large number of users generally characterize database systems. The simplified view of database system is shown in Fig. 1.1. From this figure, it is clear that several users can access the data in an

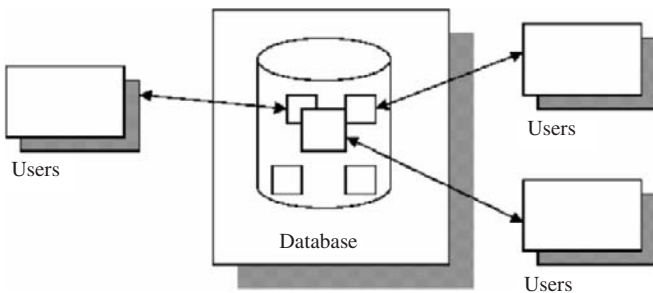


Fig. 1.1. Simplified database view

organization still the integrity of the data should be maintained. A database is integrated when same information is not recorded in two places.

1.4 Database Management System

A database management system (DBMS) consists of collection of interrelated data and a set of programs to access that data. It is software that is helpful in maintaining and utilizing a database.

A DBMS consists of:

- A collection of interrelated and persistent data. This part of DBMS is referred to as database (DB).
- A set of application programs used to access, update, and manage data. This part constitutes data management system (MS).
- A DBMS is general-purpose software i.e., not application specific. The same DBMS (e.g., Oracle, Sybase, etc.) can be used in railway reservation system, library management, university, etc.
- A DBMS takes care of storing and accessing data, leaving only application specific tasks to application programs.

DBMS is a complex system that allows a user to do many things to data as shown in Fig. 1.2. From this figure, it is evident that DBMS allows user to input data, share the data, edit the data, manipulate the data, and display the data in the database. Because a DBMS allows more than one user to share the data; the complexity extends to its design and implementation.

1.4.1 Structure of DBMS

An overview of the structure of database management system is shown in Fig. 1.3. A DBMS is a software package, which translates data from its logical representation to its physical representation and back.

The DBMS uses an application specific database description to define this translation. The database description is generated by a database designer

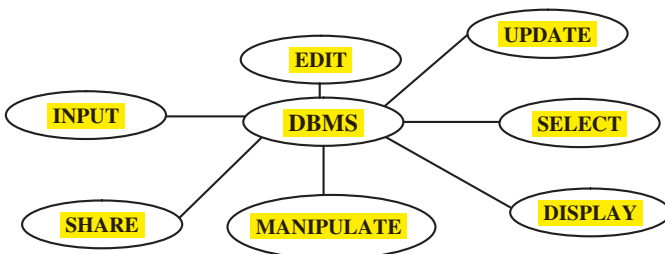


Fig. 1.2. Capabilities of database management system

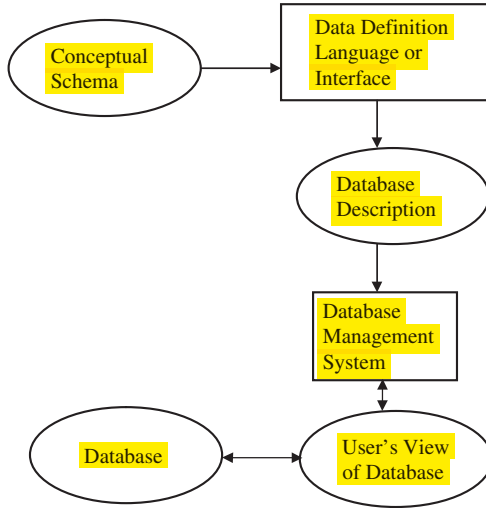


Fig. 1.3. Structure of database management system

from his or her conceptual view of the database, which is called the Conceptual Schema. The translation from the conceptual schema to the database description is performed using a data definition language (DDL) or a graphical or textual design interface.

1.5 Objectives of DBMS

The main objectives of database management system are data availability, data integrity, data security, and data independence.

1.5.1 Data Availability

Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

1.5.2 Data Integrity

Data integrity refers to the correctness of the data in the database. In other words, the data available in the database is a reliable data.

1.5.3 Data Security

Data security refers to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

1.5.4 Data Independence

DBMS allows the user to store, update, and retrieve data in an efficient manner. DBMS provides an “abstract view” of how the data is stored in the database.

In order to store the information efficiently, complex data structures are used to represent the data. The system hides certain details of how the data are stored and maintained.

1.6 Evolution of Database Management Systems

File-based system was the predecessor to the database management system. Apollo moon-landing process was started in the year 1960. At that time, there was no system available to handle and manage large amount of information. As a result, North American Aviation which is now popularly known as Rockwell International developed software known as Generalized Update Access Method (GUAM). In the mid-1960s, IBM joined North American Aviation to develop GUAM into Information Management System (IMS). IMS was based on Hierarchical data model. In the mid-1960s, General Electric released Integrated Data Store (IDS). IDS were based on network data model. Charles Bachmann was mainly responsible for the development of IDS. The network database was developed to fulfill the need to represent more complex data relationships than could be modeled with hierarchical structures. Conference on Data System Languages formed Data Base Task Group (DBTG) in 1967. DBTG specified three distinct languages for standardization. They are Data Definition Language (DDL), which would enable Database Administrator to define the schema, a subschema DDL, which would allow the application programs to define the parts of the database and Data Manipulation Language (DML) to manipulate the data.

The network and hierarchical data models developed during that time had the drawbacks of minimal data independence, minimal theoretical foundation, and complex data access. To overcome these drawbacks, in 1970, Codd of IBM published a paper titled “A Relational Model of Data for Large Shared Data Banks” in Communications of the ACM, vol. 13, No. 6, pp. 377–387, June 1970. As an impact of Codd’s paper, System R project was developed during the late 1970 by IBM San Jose Research Laboratory in California. The project was developed to prove that relational data model was implementable. The outcome of System R project was the development of Structured Query Language (SQL) which is the standard language for relational database management system. In 1980s IBM released two commercial relational database management systems known as DB2 and SQL/DS and Oracle Corporation released Oracle. In 1979, Codd himself attempted to address some of the failings in his original work with an extended version of the relational model called RM/T in 1979 and RM/V2 in 1990. The attempts to provide a data model

that represents the “real world” more closely have been loosely classified as Semantic Data Modeling.

In recent years, two approaches to DBMS are more popular, which are Object-Oriented DBMS (OODBMS) and Object Relational DBMS (ORDBMS).

The chronological order of the development of DBMS is as follows:

- Flat files – 1960s–1980s
- Hierarchical – 1970s–1990s
- Network – 1970s–1990s
- Relational – 1980s–present
- Object-oriented – 1990s–present
- Object-relational – 1990s–present
- Data warehousing – 1980s–present
- Web-enabled – 1990s–present

Early 1960s. Charles Bachman at GE created the first general purpose DBMS Integrated Data Store. It created the basis for the network model which was standardized by CODASYL (Conference on Data System Language).

Late 1960s. IBM developed the Information Management System (IMS). IMS used an alternate model, called the Hierarchical Data Model.

1970. Edgar Codd, from IBM created the Relational Data Model. In 1981 Codd received the Turing Award for his contributions to database theory. Codd Passed away in April 2003.

1976. Peter Chen presented Entity-Relationship model, which is widely used in database design.

1980. SQL developed by IBM, became the standard query language for databases. SQL was standardized by ISO.

1980s and 1990s. IBM, Oracle, Informix and others developed powerful DBMS.

1.7 Classification of Database Management System

The database management system can be broadly classified into (1) Passive Database Management System and (2) Active Database Management System:

1. *Passive Database Management System.* Passive Database Management Systems are program-driven. In passive database management system the users query the current state of database and retrieve the information currently available in the database. Traditional DBMS are passive in the sense that they are explicitly and synchronously invoked by user or application program initiated operations. Applications send requests for operations to be performed by the DBMS and wait for the DBMS to confirm and return any possible answers. The operations can be definitions and updates of the schema, as well as queries and updates of the data.

2. Active Database Management System. Active Database Management Systems are data-driven or event-driven systems. In active database management system, the users specify to the DBMS the information they need. If the information of interest is currently available, the DBMS actively monitors the arrival of the desired information and provides it to the relevant users. The scope of a query in a passive DBMS is limited to the past and present data, whereas the scope of a query in an active DBMS additionally includes future data. An active DBMS reverses the control flow between applications and the DBMS instead of only applications calling the DBMS, the DBMS may also call applications in an active DBMS.

Active databases contain a set of active rules that consider events that represent database state changes, look for TRUE or FALSE conditions as the result of a database predicate or query, and take an action via a data manipulation program embedded in the system. *Alert* is extension architecture at the IBM Almaden Research, for experimentation with active databases.

1.8 File-Based System

Prior to DBMS, file system provided by OS was used to store information. In a file-based system, we have collection of application programs that perform services for the end users. Each program defines and manages its own data.

Consider University database, the University database contains details about student, faculty, lists of courses offered, and duration of course, etc. In File-based processing for each database there is separate application program which is shown in Fig. 1.4.

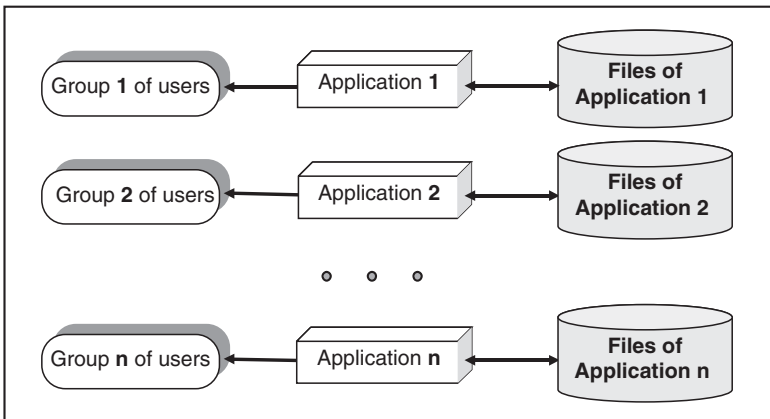


Fig. 1.4. File-based System

One group of users may be interested in knowing the courses offered by the university. One group of users may be interested in knowing the faculty information. The information is stored in separate files and separate applications programs are written.

1.9 Drawbacks of File-Based System

The limitations of file-based approach are duplication of data, data dependence, incompatible file formats, separation, and isolation of data.

1.9.1 Duplication of Data

Duplication of data means same data being stored more than once. This can also be termed as data redundancy. Data redundancy is a problem in file-based approach due to the decentralized approach. The main drawbacks of duplication of data are:

- Duplication of data leads to wastage of storage space. If the storage space is wasted it will have a direct impact on cost. The cost will increase.
- Duplication of data can lead to loss of data integrity; the data are no longer consistent. Assume that the employee detail is stored both in the department and in the main office. Now the employee changes his contact address. The changed address is stored in the department alone and not in the main office. If some important information has to be sent to his contact address from the main office then that information will be lost. This is due to the lack of decentralized approach.

1.9.2 Data Dependence

Data dependence means the application program depends on the data. If some modifications have to be made in the data, then the application program has to be rewritten. If the application program is independent of the storage structure of the data, then it is termed as data independence. Data independence is generally preferred as it is more flexible. But in file-based system there is program-data dependence.

1.9.3 Incompatible File Formats

As file-based system lacks program data independence, the structure of the file depends on the application programming language. For example, the structure of the file generated by FORTRAN program may be different from the structure of a file generated by “C” program. The incompatibility of such files makes them difficult to process jointly.

1.9.4 Separation and Isolation of Data

In file-based approach, data are isolated in separate files. Hence it is difficult to access data. The application programmer must synchronize the processing of two files to ensure that the correct data are extracted. This difficulty is more if data has to be retrieved from more than two files.

The draw backs of conventional file-based approach are summarized later:

1. We have to store the information in a secondary memory such as a disk. If the volume of information is large; it will occupy more memory space.
2. We have to depend on the addressing facilities of the system. If the database is very large, then it is difficult to address the whole set of records.
3. For each query, for example the address of the student and the list of electives that the student has chosen, we have to write separate programs.
4. While writing several programs, lot of variables will be declared and it will occupy some space.
5. It is difficult to ensure the integrity and consistency of the data when more than one program accesses some file and changes the data.
6. In case of a system crash, it becomes hard to bring back the data to a consistent state.
7. “Data redundancy” occurs when identical data are distributed over various files.
8. Data distributed in various files may be in different formats hence it is difficult to share data among different application (Data Isolation).

1.10 DBMS Approach

DBMS is software that provides a set of primitives for defining, accessing, and manipulating data. In DBMS approach, the same data are being shared by different application programs; as a result data redundancy is minimized. The DBMS approach of data access is shown in Fig. 1.5.

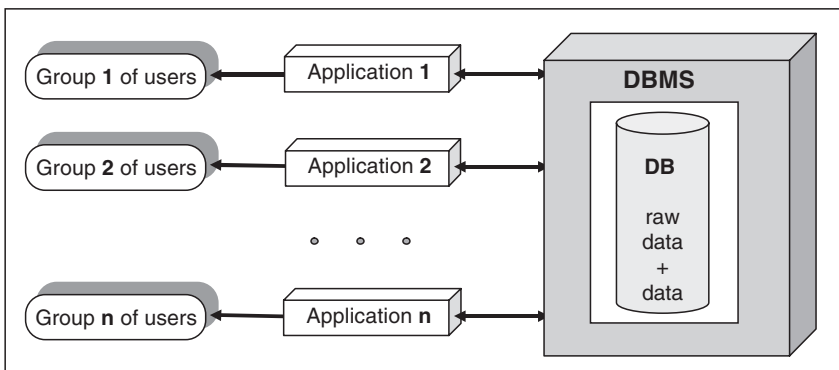


Fig. 1.5. Data access through DBMS

1.11 Advantages of DBMS

There are many advantages of database management system. Some of the advantages are listed later:

1. Centralized data management.
2. Data Independence.
3. System Integration.

1.11.1 Centralized Data Management

In DBMS all files are integrated into one system thus reducing redundancies and making data management more efficient.

1.11.2 Data Independence

Data independence means that programs are isolated from changes in the way the data are structured and stored. In a database system, the database management system provides the interface between the application programs and the data. Physical data independence means the applications need not worry about how the data are physically structured and stored. Applications should work with a logical data model and declarative query language.

If major changes were to be made to the data, the application programs may need to be rewritten. When changes are made to the data representation, the data maintained by the DBMS is changed but the DBMS continues to provide data to application programs in the previously used way.

Data independence is the immunity of application programs to changes in storage structures and access techniques. For example if we add a new attribute, change index structure then in traditional file processing system, the applications are affected. But in a DBMS environment these changes are reflected in the catalog, as a result the applications are not affected. Data independence can be physical data independence or logical data independence.

Physical data independence is the ability to modify physical schema without causing the conceptual schema or application programs to be rewritten.

Logical data independence is the ability to modify the conceptual schema without having to change the external schemas or application programs.

1.11.3 Data Inconsistency

Data inconsistency means different copies of the same data will have different values. For example, consider a person working in a branch of an organization. The details of the person will be stored both in the branch office as well as in the main office. If that particular person changes his address, then the “change of address” has to be maintained in the main as well as the branch office.

For example the “change of address” is maintained in the branch office but not in the main office, then the data about that person is inconsistent.

DBMS is designed to have data consistency. Some of the qualities achieved in DBMS are:

1. Data redundancy → Reduced in DBMS.
 2. Data independence → Activated in DBMS.
 3. Data inconsistency → Avoided in DBMS.
 4. Centralizing the data → Achieved in DBMS.
 5. Data integrity → Necessary for efficient Transaction.
 6. Support for multiple views → Necessary for security reasons.
- Data redundancy means duplication of data. Data redundancy will occupy more space hence it is not desirable.
 - Data independence means independence between application program and the data. The advantage is that when the data representation changes, it is not necessary to change the application program.
 - Data inconsistency means different copies of the same data will have different values.
 - Centralizing the data means data can be easily shared between the users but the main concern is data security.
 - The main threat to data integrity comes from several different users attempting to update the same data at the same time. For example, “The number of booking made is larger than the capacity of the aircraft/train.”
 - Support for multiple views means DBMS allows different users to see different “views” of the database, according to the perspective each one requires. This concept is used to enhance the security of the database.

1.12 Ansi/Spark Data Model (American National Standard Institute/ Standards Planning and Requirements Committee)

The distinction between the logical and physical representation of data were recognized in 1978 when ANSI/SPARK committee proposed a generalized framework for database systems. This framework provided a three-level architecture, three levels of abstraction at which the database could be viewed.

1.12.1 Need for Abstraction

The main objective of DBMS is to store and retrieve information efficiently; all the users should be able to access same data. The designers use complex data structure to represent the data, so that data can be efficiently stored and retrieved, but it is not necessary for the users to know physical database storage details. The developers hide the complexity from users through several levels of abstraction.

1.12.2 Data Independence

Data independence means the internal structure of database should be unaffected by changes to physical aspects of storage. Because of data independence, the Database administrator can change the database storage structures without affecting the users view.

The different levels of data abstraction are:

1. Physical level or internal level
2. Logical level or conceptual level
3. View level or external level

Physical Level

It is concerned with the physical storage of the information. It provides the internal view of the actual physical storage of data. The physical level describes complex low-level data structures in detail.

Logical Level

Logical level describes what data are stored in the database and what relationships exist among those data.

Logical level describes the entire database in terms of a small number of simple structures. The implementation of simple structure of the logical level may involve complex physical level structures; the user of the logical level does not need to be aware of this complexity. Database administrator use the logical level of abstraction.

View Level

View level is the highest level of abstraction. It is the view that the individual user of the database has. There can be many view level abstractions of the same data. The different levels of data abstraction are shown in Fig. 1.6.

Database Instances

Database change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database.

Database Schema

The overall design of the database is called the database schema. A schema is a collection of named objects. Schemas provide a logical classification of objects in the database. A schema can contain tables, views, triggers, functions, packages, and other objects.

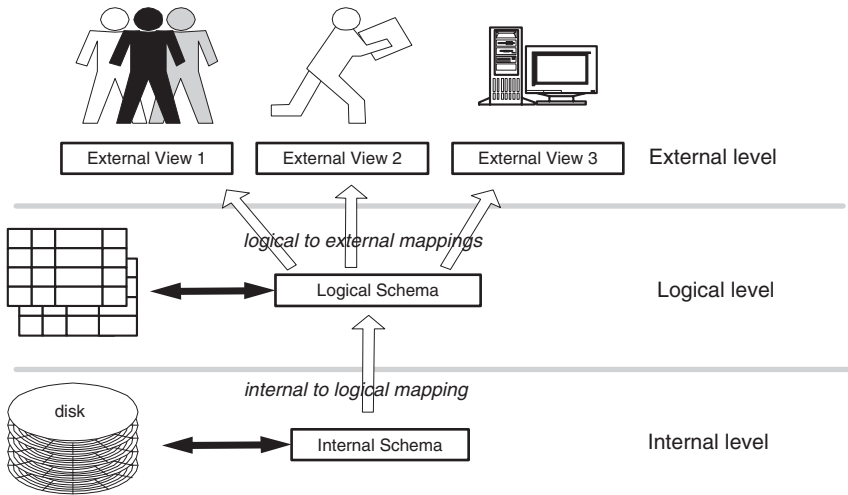
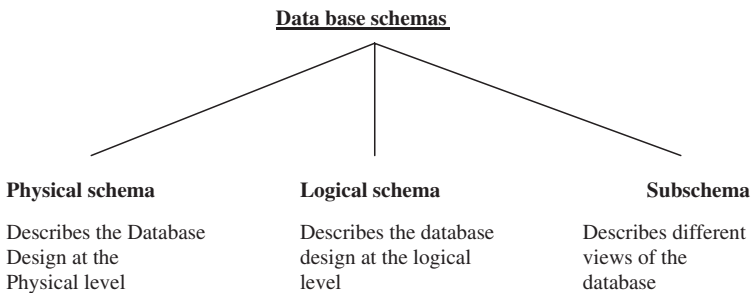


Fig. 1.6. ANSI/SPARK data model

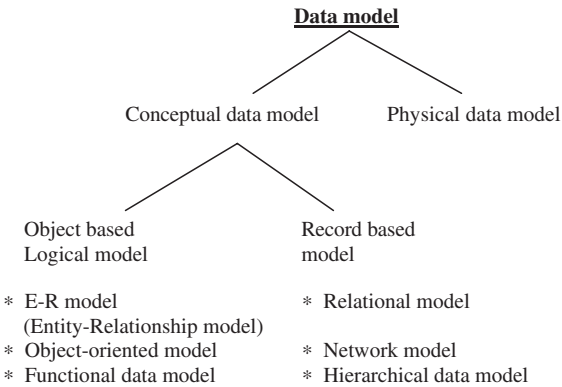
A schema is also an object in the database. It is explicitly created using the `CREATE SCHEMA` statement with the current user recorded as the schema owner. It can also be implicitly created when another object is created, provided the user has `IMPLICIT_SCHEMA` authority.



1.13 Data Models

Data model is collection of conceptual tools for describing data, relationship between data, and consistency constraints. Data models help in describing the structure of data at the logical level. Data model describe the structure of the database. A data model is the set of conceptual constructs available for defining a schema. The data model is a language for describing the data and database, it may consist of abstract concepts, which must be translated by the

designer into the constructs of the data definition interface, or it may consist of constructs, which are directly supported by the data definition interface. The constructs of the data model may be defined at many levels of abstraction.



1.13.1 Early Data Models

Three historically important data models are the hierarchical, network, and relational models. These models are relevant for their contributions in establishing the theory of data modeling and because they were all used as the basis of working and widely used database systems. Together they are often referred to as the “basic” data models. The hierarchical and network models, developed in the 1960s and 1970s, were based on organizing the primitive data structures in which the data were stored in the computer by adding connections or links between the structures. As such they were useful in presenting the user with a well-defined structure, but they were still highly coupled to the underlying physical representation of the data. Although they did much to assist in the efficient access of data, the principle of data independence was poorly supported.

1.14 Components and Interfaces of Database Management System

A database management system involves five major components: data, hardware, software, procedure, and users. These components and the interface between the components are shown in Fig. 1.7.

1.14.1 Hardware

The hardware can range from a single personal computer, to a single main-frame, to a network of computers. The particular hardware depends on the

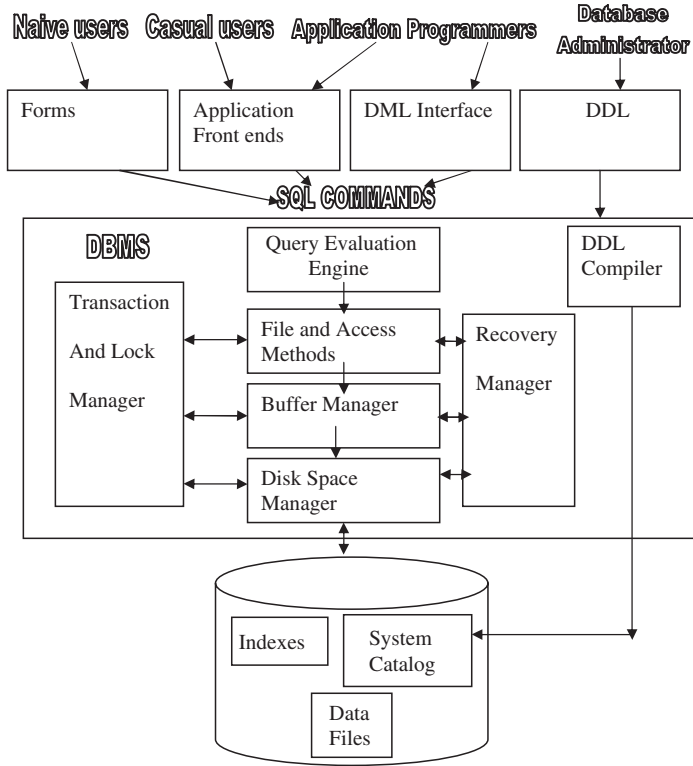


Fig. 1.7. Database management system components and interfaces

requirements of the organization and the DBMS used. Some DBMSs run only on particular operating systems, while others run on a wide variety of operating systems. A DBMS requires a minimum amount of main memory and disk space to run, but this minimum configuration may not necessarily give acceptable performance.

1.14.2 Software

The software includes the DBMS software, application programs together with the operating systems including the network software if the DBMS is being used over a network. The application programs are written in third-generation programming languages like “C,” COBOL, FORTRAN, Ada, Pascal, etc. or using fourth-generation language such as SQL, embedded in a third-generation language. The target DBMS may have its own fourth-generation tools which allow development of applications through the provision of nonprocedural query languages, report generators, graphics generators, and application generators. The use of fourth-generation tools can improve productivity significantly and produce programs that are easier to maintain.

1.14.3 Data

A database is a repository for data which, in general, is both integrated and shared. Integration means that the database may be thought of as a unification of several otherwise distinct files, with any redundancy among those files partially or wholly eliminated. The sharing of a database refers to the sharing of data by different users, in the sense that each of those users may have access to the same piece of data and may use it for different purposes. Any given user will normally be concerned with only a subset of the whole database. The main features of the data in the database are listed later:

1. The data in the database is well organized (structured)
2. The data in the database is related
3. The data are accessible in different orders without great difficulty

The data in the database is persistent, integrated, structured, and shared.

Integrated Data

A data can be considered to be a unification of several distinct data files and when any redundancy among those files is eliminated, the data are said to be integrated data.

Shared Data

A database contains data that can be shared by different users for different application simultaneously. It is important to note that in this way of sharing of data, the redundancy of data are reduced, since repetitions are avoided, the possibility of inconsistencies is reduced.

Persistent Data

Persistent data are one, which cannot be removed from the database as a side effect of some other process. Persistent data have a life span that is not limited to single execution of the programs that use them.

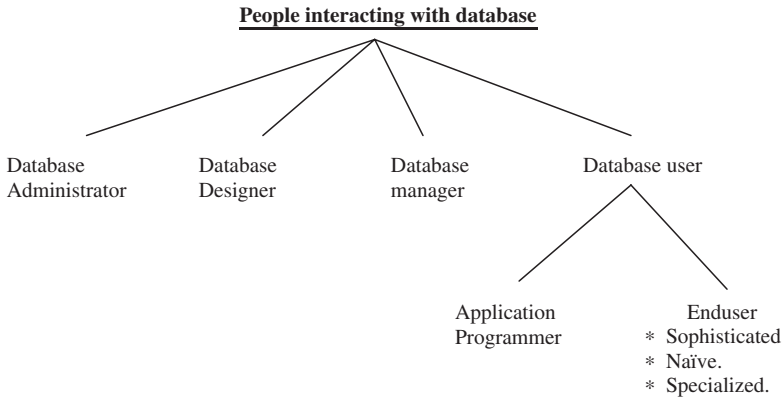
1.14.4 Procedure

Procedures are the rules that govern the design and the use of database. The procedure may contain information on how to log on to the DBMS, start and stop the DBMS, procedure on how to identify the failed component, how to recover the database, change the structure of the table, and improve the performance.

1.14.5 People Interacting with Database

Here people refers to the people who manages the database, database administrator, people who design the application program, database designer and the people who interacts with the database, database users.

A DBMS is typically run as a back-end server in a local or global network, offering services to clients directly or to Application Servers.



Database Administrator

Database Administrator is a person having central control over data and programs accessing that data. The database administrator is a manager whose responsibilities are focused on management of technical aspects of the database system. The objectives of database administrator are given as follows:

1. To control the database environment
2. To standardize the use of database and associated software
3. To support the development and maintenance of database application projects
4. To ensure all documentation related to standards and implementation is up-to-date

The summarized objectives of database administrator are shown in Fig. 1.8.

The control of the database environment should exist from the planning right through to the maintenance stage. During application development the database administrator should carry out the tasks that ensure proper control of the database when an application becomes operational. This includes review of each design stage to see if it is feasible from the database point of view. The database administrator should be responsible for developing standards to apply to development projects. In particular these standards apply to system analysis, design, and application programming for projects which are going to use the database. These standards will then be used as a basis for training systems analysts and programmers to use the database management system efficiently.

Responsibilities of Database Administrator (DBA)

The responsibility of the database administrator is to maintain the integrity, security, and availability of data. A database must be protected from

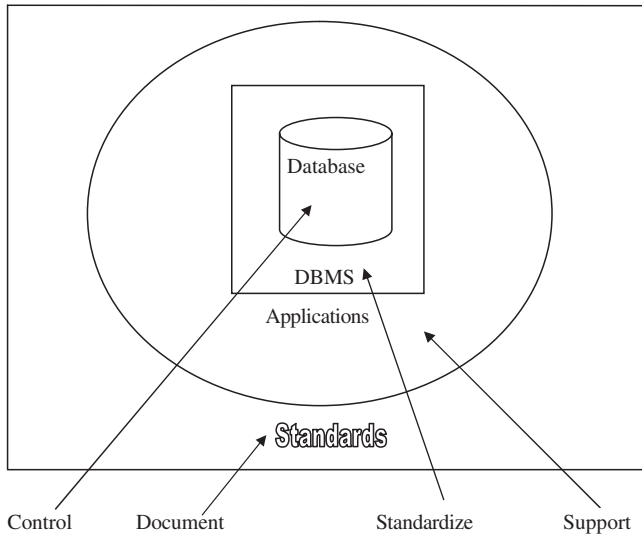


Fig. 1.8. Objectives of database administration

accidents, such as input or programming errors, from malicious use of the database and from hardware or software failures that corrupt data. Protection from accidents that cause data inaccuracy is a part of maintaining data integrity. Protecting the database from unauthorized or malicious use is termed as database security. The responsibilities of the database administrator are summarized as follows:

1. Authorizing access to the database.
2. Coordinating and monitoring its use.
3. Acquiring hardware and software resources as needed.
4. Backup and recovery. DBA has to ensure regular backup of database, in-case of damage, suitable recovery procedure are used to bring the database up with little downtime as possible.

Database Designer

Database designer can be either logical database designer or physical database designer. Logical database designer is concerned with identifying the data, the relationships between the data, and the constraints on the data that is to be stored in the database. The logical database designer must have thorough understanding of the organizations data and its business rule.

The physical database designer takes the logical data model and decides the way in which it can be physically implemented. The logical database designer is responsible for mapping the logical data model into a set of tables and integrity constraints, selecting specific storage structure, and designing

security measures required on the data. In a nutshell, the database designer is responsible for:

1. Identifying the data to be stored in the database.
2. Choosing appropriate structure to represent and store the data.

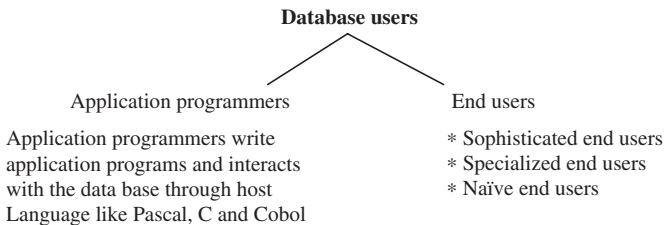
Database Manager

Database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system:

- The database manager would translate DML statement into low level file system commands for storing, retrieving, and updating data in the database.
- *Integrity enforcement.* Database manager enforces integrity by checking consistency constraints like the bank balance of customer must be maintained to a minimum of Rs. 300, etc.
- *Security enforcement.* Unauthorized users are prohibited to view the information stored in the data base.
- *Backup and recovery.* Backup and recovery of database is necessary to ensure that the database must remain consistent despite the fact of failures.

Database Users

Database users are the people who need information from the database to carry out their business responsibility. The database users can be broadly classified into two categories like application programmers and end users.



Sophisticated End Users

Sophisticated end users interact with the system without writing programs. They form requests by writing queries in a database query language. These are submitted to query processor. Analysts who submit queries to explore data in the database fall in this category.

Specialized End Users

Specialized end users write specialized database application that does not fit into data-processing frame work. Application involves knowledge base and expert system, environment modeling system, etc.

Naive End Users

Naïve end user interact with the system by using permanent application program Example: Query made by the student, namely number of books borrowed in library database.

System Analysts

System analysts determine the requirements of end user, and develop specification for canned transaction that meets this requirement.

Canned Transaction

Ready made programs through which naïve end users interact with the database is called canned transaction.

1.14.6 Data Dictionary

A data dictionary, also known as a “system catalog,” is a centralized store of information about the database. It contains information about the tables, the fields the tables contain, data types, primary keys, indexes, the joins which have been established between those tables, referential integrity, cascades update, cascade delete, etc. This information stored in the data dictionary is called the “Metadata.” Thus a data dictionary can be considered as a file that stores Metadata. Data dictionary is a tool for recording and processing information about the data that an organization uses. The data dictionary is a central catalog for Metadata. The data dictionary can be integrated within the DBMS or separate. Data dictionary may be referenced during system design, programming, and by actively-executing programs. One of the major functions of a true data dictionary is to enforce the constraints placed upon the database by the designer, such as referential integrity and cascade delete.

Metadata

The information (data) about the data in a database is called Metadata. The Metadata are available for query and manipulation, just as other data in the database.

1.14.7 Functional Components of Database System Structure

The functional components of database system structure are:

1. Storage manager.
2. Query processor.

Storage Manager

Storage manager is responsible for storing, retrieving, and updating data in the database. Storage manager components are:

1. Authorization and integrity manager.
2. Transaction manager.
3. File manager.
4. Buffer manager.

Transaction Management

- A transaction is a collection of operations that performs a single logical function in a database application.
- Transaction-management component ensures that the database remains in a consistent state despite system failures and transaction failure.
- Concurrency control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Authorization and Integrity Manager

Checks the integrity constraints and authority of users to access data.

Transaction Manager

It ensures that the database remains in a consistent state despite system failures. The transaction manager manages the execution of database manipulation requests. The transaction manager function is to ensure that concurrent access to data does not result in conflict.

File Manager

File manager manages the allocation of space on disk storage. Files are used to store collections of similar data. A file management system manages independent files, helping to enter and retrieve information records. File manager establishes and maintains the list of structure and indexes defined in the internal schema. The file manager can:

- Create a file
- Delete a file
- Update the record in the file
- Retrieve a record from a file

Buffer

The area into which a block from the file is read is termed a buffer. The management of buffers has the objective of maximizing the performance or the utilization of the secondary storage systems, while at the same time keeping the demand on CPU resources tolerably low. The use of two or more buffers for a file allows the transfer of data to be overlapped with the processing of data.

Buffer Manager

Buffer manager is responsible for fetching data from disk storage into main memory. Programs call on the buffer manager when they need a block from disk. The requesting program is given the address of the block in main memory, if it is already present in the buffer. If the block is not in the buffer, the buffer manager allocates space in the buffer for the block, replacing some other block, if required, to make space for new block. Once space is allocated in the buffer, the buffer manager reads in the block from the disk to the buffer, and passes the address of the block in main memory to the requester.

Indices

Indices provide fast access to data items that hold particular values. An index is a list of numerical values which gives the order of the records when they are sorted on a particular field or column of the table.

1.15 Database Architecture

Database architecture essentially describes the location of all the pieces of information that make up the database application. The database architecture can be broadly classified into two-, three-, and multitier architecture.

1.15.1 Two-Tier Architecture

The two-tier architecture is a client-server architecture in which the client contains the presentation code and the SQL statements for data access. The database server processes the SQL statements and sends query results back to the client. The two-tier architecture is shown in Fig. 1.9. Two-tier client/server provides a basic separation of tasks. The client, or first tier, is primarily responsible for the *presentation* of data to the user and the “server,” or second tier, is primarily responsible for supplying *data services* to the client.

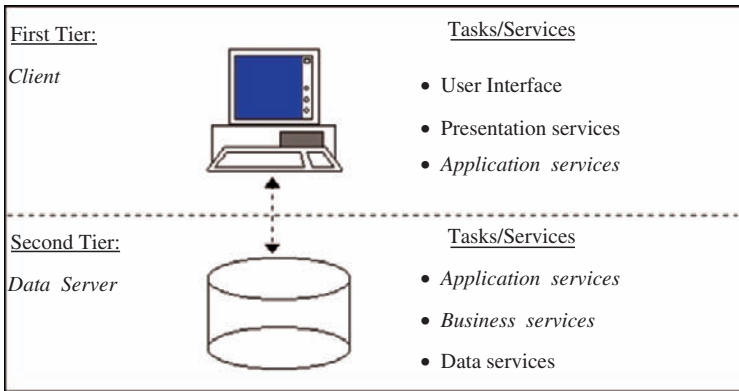


Fig. 1.9. Two-tier client-server architecture

Presentation Services

“Presentation services” refers to the portion of the application which presents data to the user. In addition, it also provides for the mechanisms in which the user will interact with the data. More simply put, presentation logic defines and interacts with the user interface. The presentation of the data should generally not contain any validation rules.

Business Services/objects

“Business services” are a category of application services. Business services encapsulate an organizations business processes and requirements. These rules are derived from the steps necessary to carry out day-today business in an organization. These rules can be validation rules, used to be sure that the incoming information is of a valid type and format, or they can be process rules, which ensure that the proper business process is followed in order to complete an operation.

Application Services

“Application services” provide other functions necessary for the application.

Data Services

“Data services” provide access to data independent of their location. The data can come from legacy mainframe, SQL RDBMS, or proprietary data access systems. Once again, the data services provide a standard interface for accessing data.

Advantages of Two-tier Architecture

The two-tier architecture is a good approach for systems with stable requirements and a moderate number of clients. The two-tier architecture is the simplest to implement, due to the number of good commercial development environments.

Drawbacks of Two-tier Architecture

Software maintenance can be difficult because PC clients contain a mixture of presentation, validation, and business logic code. To make a significant change in the business logic, code must be modified on many PC clients. Moreover the performance of two-tier architecture can be poor when a large number of clients submit requests because the database server may be overwhelmed with managing messages. With a large number of simultaneous clients, three-tier architecture may be necessary.

1.15.2 Three-tier Architecture

A “Multitier,” often referred to as “three-tier” or “ N -tier,” architecture provides greater application scalability, lower maintenance, and increased reuse of components. Three-tier architecture offers a technology neutral method of building client/server applications with vendors who employ standard interfaces which provide services for each logical “tier.” The three-tier architecture is shown in Fig. 1.10. From this figure, it is clear that in order to improve the performance a second-tier is included between the client and the server.

Through standard tiered interfaces, services are made available to the application. A single application can employ many different services which may reside on dissimilar platforms or are developed and maintained with different tools. This approach allows a developer to leverage investments in existing systems while creating new application which can utilize existing resources.

Although the three-tier architecture addresses performance degradations of the two-tier architecture, it does not address division-of-processing concerns. The PC clients and the database server still contain the same division of code although the tasks of the database server are reduced. Multiple-tier architectures provide more flexibility on division of processing.

1.15.3 Multitier Architecture

A multi-tier, three-tier, or N -tier implementation employs a three-tier logical architecture superimposed on a distributed physical model. Application Servers can access other application servers in order to supply services to the client application as well as to other Application Servers. The multiple-tier architecture is the most general client-server architecture. It can be most difficult to implement because of its generality. However, a good design and

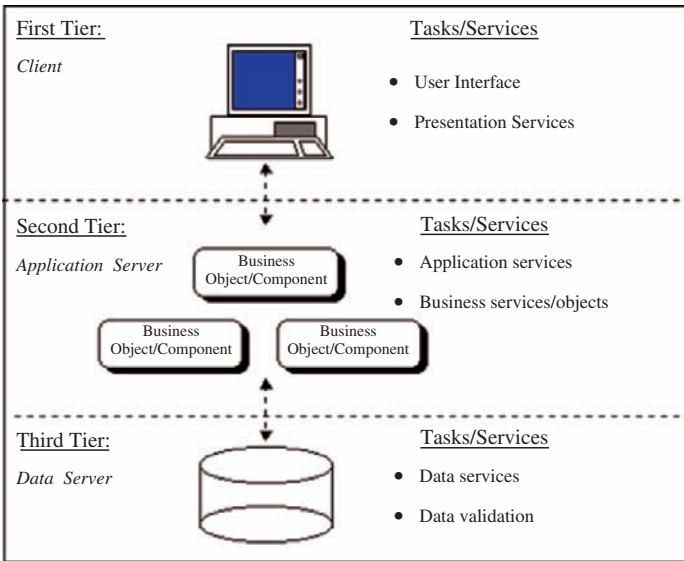


Fig. 1.10. Three-tier client-server architecture

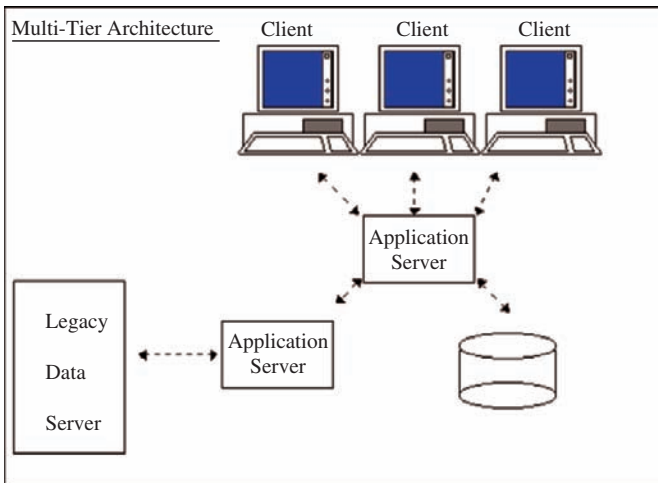


Fig. 1.11. Multiple-tier architecture

implementation of multiple-tier architecture can provide the most benefits in terms of scalability, interoperability, and flexibility.

For example, in the diagram shown in Fig.1.11, the client application looks to *Application Server #1* to supply data from a mainframe-based application. *Application Server #1* has no direct access to the mainframe application, but it does know, through the development of application services, that

Application Server #2 provides a service to access the data from the main-frame application which satisfies the client request. Application Server #1 then invokes the appropriate service on Application Server #2 and receives the requested data which is then passed on to the client.

Application Servers can take many forms. An Application Server may be anything from custom application services, Transaction Processing Monitors, Database Middleware, Message Queue to a CORBA/COM based solution.

1.16 Situations where DBMS is not Necessary

It is also necessary to specify situations where it is not necessary to use a DBMS. If traditional file processing system is working well, and if it takes more money and time to design a database, it is better not to go for the DBMS. Moreover if only one person maintains the data and that person is not skilled in designing a database as well as not comfortable in using the DBMS then it is not advisable to go for DBMS.

DBMS is undesirable under following situations:

- DBMS is undesirable if the application is simple, well-defined, and not expected to change.
- Runtime overheads are not feasible because of real-time requirements.
- Multiple accesses to data are not required.

Compared with file systems, databases have some disadvantages:

1. High cost of DBMS which includes:
 - Higher hardware costs
 - Higher programming costs
 - High conversion costs
2. Slower processing of some applications
3. Increased vulnerability
4. More difficult recovery

1.17 DBMS Vendors and their Products

Some of the popular DBMS vendors and their corresponding products are given Table 1.1.

Summary

The main objective of database management system is to store and manipulate the data in an efficient manner. A database is an organized collection of related data. All the data will not give useful information. Only processed data gives useful information, which helps an organization to take important

Table 1.1. DBMS vendors and their products

vendor	product
IBM	–DB2/MVS
	–DB2/UDB
	–DB2/400
	–Informix Dynamic Server (IDS)
Microsoft	–Access
	–SQL Server
	–DesktopEdition(MSDE)
Open Source	–MySQL
	–PostgreSQL
Oracle	–Oracle DBMS
	–RDB
Sybase	–Adaptive Server Enterprise (ASE)
	–Adaptive Server Anywhere (ASA)
	–Watcom

decisions. Before DBMS, computer file processing systems were used to store, manipulate, and retrieve large files of data. Computer file processing systems have limitations such as data duplications, limited data sharing, and no program data independence. In order to overcome these limitations database approach was developed. The main advantages of DBMS approach are program-data independence, improved data sharing, and minimal data redundancy. In this chapter we have seen the evolution of DBMS and broad introduction to DBMS. The responsibilities of Database administrator, ANSI/SPARK, two-tier, three-tier architecture were analyzed in this chapter.

Review Questions

1.1. What are the drawbacks of file processing system?

The drawbacks of file processing system are:

- Duplication of data, which leads to wastage of storage space and data inconsistency.
- Separation and isolation of data, because of which data cannot be used together.
- No program data independence.

1.2. What is meant by Metadata?

Metadata are data about data but not the actual data.

1.3. Define the term data dictionary?

Data dictionary is a file that contains Metadata.

1.4. What are the responsibilities of database administrator?**1.5. Mention three situations where it is not desirable to use DBMS?**

The situations where it is not desirable to use DBMS are:

- The database and applications are not expected to change.
- Data are not accessed by multiple users.

1.6. What is meant by data independence?

Data independence renders application programs (e.g., SQL scripts) immune to changes in the logical and physical organization of data in the system.

Logical organization refers to changes in the Schema. Example adding a column or tuples does not stop queries from working.

Physical organization refers to changes in indices, file organizations, etc.

1.7. What is meant by Physical and Logical data independence?

In logical data independence, the conceptual schema can be changed without changing the external schema. In physical data independence, the internal schema can be changed without changing the conceptual schema.

1.8. What are some disadvantages of using a DBMS over flat file system?

- DBMS initially costs more than flat file system
- DBMS requires skilled staff

1.9. What are the steps to design a good database?

- First find out the requirements of the user
- Design a view for each important application
- Integrate the views giving the conceptual schema, which is the union of all views
- Map to the data model provided by the DBMS (usually relational)
- Design external views
- Choose physical structures (indexes, etc.)

1.10. What is Database? Give an example.

A Database is a collection of related data. Here, the term “data” means that known facts that can be record. Examples of database are library information system, bus, railway, and airline reservation system, etc.

1.11. Define – DBMS.

DBMS is a collection of programs that enables users to create and maintain a database.

1.12. Mention various types of databases?

The different types of databases are:

- Multimedia database
- Spatial database (Geographical Information System Database)
- Real-time or Active Database
- Data Warehouse or On-line Analytical Processing Database

1.13. Mention the advantages of using DBMS?

The advantages of using DBMS are:

- Controlling Redundancy
- Enforcing Integrity Constraints so as to maintain the consistency of the database
- Providing Backup and recovery facilities
- Restricting unauthorized access
- Providing multiple user interfaces
- Providing persistent storage of program objects and datastructures

1.14. What is “Snapshot” or “Database State”?

The data in the database at a particular moment is known as “Database State” or “Snapshot” of the Database.

1.15. Define Data Model.

It is a collection of concepts that can be used to describe the structure of a database.

The datamodel provides necessary means to achieve the abstraction i.e., hiding the details of data storage.

1.16. Mention the various categories of Data Model.

The various categories of datamodel are:

- High Level or Conceptual Data Model (Example: ER model)
- Low Level or Physical Data Model
- Representational or Implementational Data Model
- Relational Data Model
- Network and Hierarchal Data Model
- Record-based Data Model
- Object-based Data Model

1.17. Define the concept of “database schema.” Describe the types of schemas that exist in a database complying with the three levels ANSI/SPARC architecture.

Database schema is nothing but description of the database. The types of schemas that exist in a database complying with three levels of ANSI/SPARC architecture are:

- External schema
- Conceptual schema
- Internal schema

Entity–Relationship Model

Learning Objectives. This chapter presents a top-down approach to data modeling. This chapter deals with ER and Enhanced ER (EER) model and conversion of ER model to relational model. After completing this chapter the reader should be familiar with the following concepts:

- Entity, Attribute, and Relationship.
- Entity classification – Strong entity, Weak entity, and Associative entity.
- Attribute classification – Single value, Multivalued, Derived, and Null attribute.
- Relationship – Unary, binary, and ternary relationship.
- Enhanced ER model – Generalization, Specialization.
- Mapping ER model to relation model or table.
- Connection traps.

2.1 Introduction

Peter Chen first proposed modeling databases using a graphical technique that humans can relate to easily. Humans can easily perceive entities and their characteristics in the real world and represent any relationship with one another. The objective of modeling graphically is even more profound than simply representing these entities and relationship. The database designer can use tools to model these entities and their relationships and then generate database vendor-specific schema automatically. Entity–Relationship (ER) model gives the conceptual model of the world to be represented in the database. ER Model is based on a perception of a real world that consists of collection of basic objects called entities and relationships among these objects. The main motivation for defining the ER model is to provide a high level model for conceptual database design, which acts as an intermediate stage prior to mapping the enterprise being modeled onto a conceptual level. The ER model achieves a high degree of data independence which means that the database designer do not have to worry about the physical structure of the database. A database schema in ER model can be pictorially represented by Entity–Relationship diagram.

2.2 The Building Blocks of an Entity–Relationship Diagram

ER diagram is a graphical modeling tool to standardize ER modeling. The modeling can be carried out with the help of pictorial representation of entities, attributes, and relationships. The basic building blocks of Entity–Relationship diagram are Entity, Attribute and Relationship.

2.2.1 Entity

An entity is an object that exists and is distinguishable from other objects. In other words, the entity can be uniquely identified.

The examples of entities are:

- A particular person, for example Dr. A.P.J. Abdul Kalam is an entity.
- A particular department, for example Electronics and Communication Engineering Department.
- A particular place, for example Coimbatore city can be an entity.

2.2.2 Entity Type

An entity type or entity set is a collection of similar entities. Some examples of entity types are:

- All students in PSG, say STUDENT.
- All courses in PSG, say COURSE.
- All departments in PSG, say DEPARTMENT.

An entity may belong to more than one entity type. For example, a staff working in a particular department can pursue higher education as part-time. Hence the same person is a LECTURER at one instance and STUDENT at another instance.

2.2.3 Relationship

A relationship is an association of entities where the association includes one entity from each participating entity type whereas relationship type is a meaningful association between entity types.

The examples of relationship types are:

- Teaches is the relationship type between LECTURER and STUDENT.
- Buying is the relationship between VENDOR and CUSTOMER.
- Treatment is the relationship between DOCTOR and PATIENT.

2.2.4 Attributes

Attributes are properties of entity types. In other words, entities are described in a database by a set of attributes.

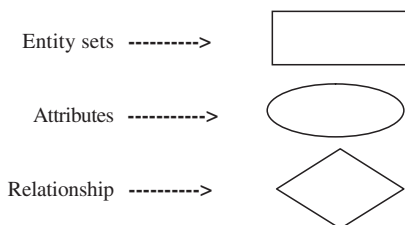
The following are example of attributes:

- Brand, cost, and weight are the attributes of CELLPHONE.
- Roll number, name, and grade are the attributes of STUDENT.
- Data bus width, address bus width, and clock speed are the attributes of MICROPROCESSOR.

2.2.5 ER Diagram

The ER diagram is used to represent database schema. In ER diagram:

- A rectangle represents an entity set.
- An ellipse represents an attribute.
- A diamond represents a relationship.
- Lines represent linking of attributes to entity sets and of entity sets to relationship sets.



Example of ER diagram

Let us consider a simple ER diagram as shown in Fig. 2.1.

In the ER diagram the two entities are STUDENT and CLASS. Two simple attributes which are associated with the STUDENT are Roll number and the name. The attributes associated with the entity CLASS are Subject Name and Hall Number. The relationship between the two entities STUDENT and CLASS is Attends.

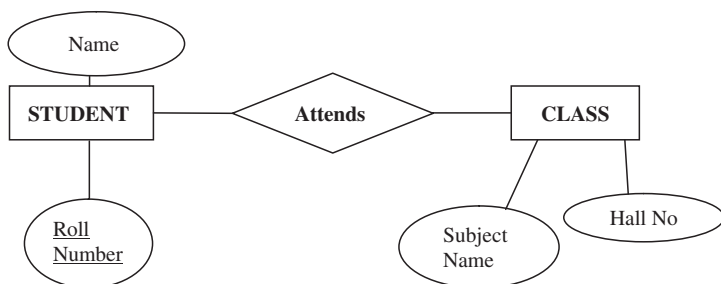
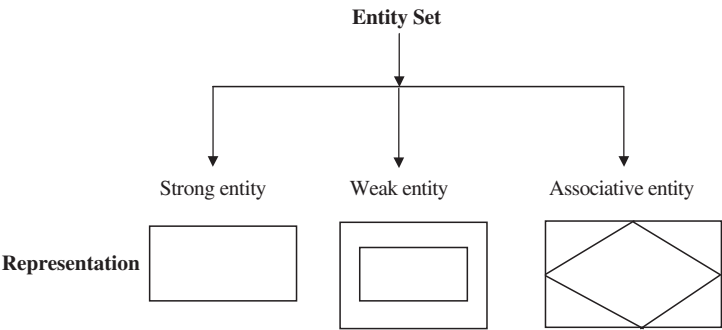


Fig. 2.1. ER diagram

2.3 Classification of Entity Sets

Entity sets can be broadly classified into:

- 1. Strong entity.
- 2. Weak entity.
- 3. Associative entity.

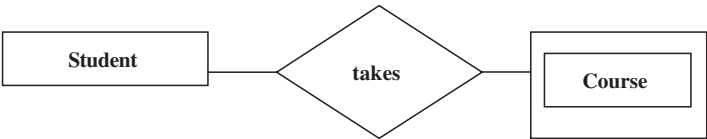


2.3.1 Strong Entity

Strong entity is one whose existence does not depend on other entity.

Example

Consider the example, student takes course. Here student is a strong entity.



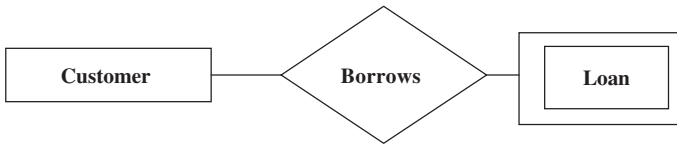
In this example, course is considered as weak entity because, if there are no students to take a particular course, then that course cannot be offered. The COURSE entity depends on the STUDENT entity.

2.3.2 Weak Entity

Weak entity is one whose existence depends on other entity. In many cases, weak entity does not have primary key.

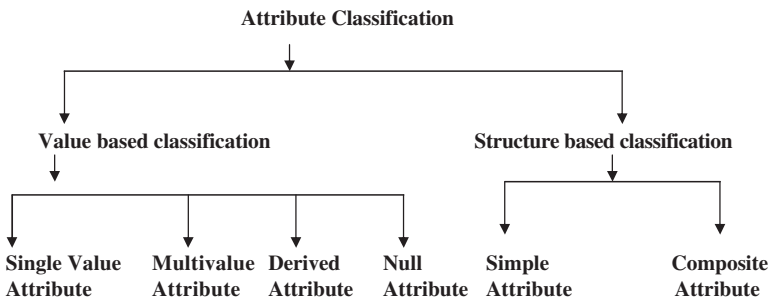
Example

Consider the example, customer borrows loan. Here loan is a weak entity. For every loan, there should be at least one customer. Here the entity loan depends on the entity customer hence loan is a weak entity.



2.4 Attribute Classification

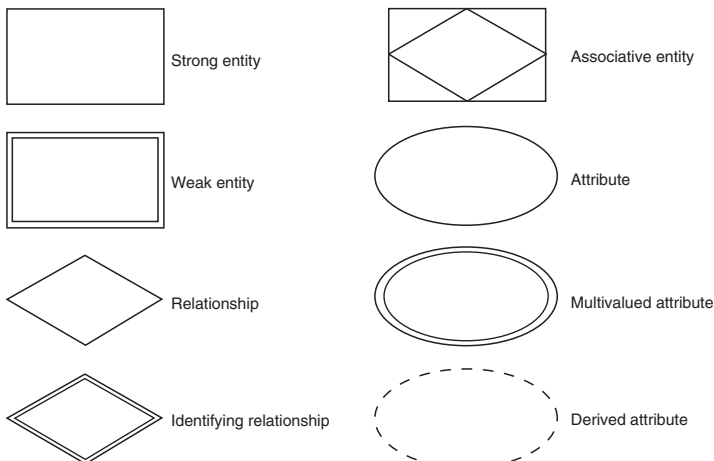
Attribute is used to describe the properties of the entity. This attribute can be broadly classified based on value and structure. Based on value the attribute can be classified into single value, multivalued, derived, and null value attribute. Based on structure, the attribute can be classified as simple and composite attribute.



2.4.1 Symbols Used in ER Diagram

The elements in ER diagram are Entity, Attribute, and Relationship. The different types of entities like strong, weak, and associative entity, different types of attributes like multivalued and derived attributes and identifying relationship and their corresponding symbols are shown later.

Basic symbols



Single Value Attribute

Single value attribute means, there is only one value associated with that attribute.

Example

The examples of single value attribute are age of a person, Roll number of the student, Registration number of a car, etc.

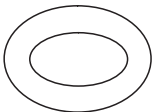
Representation of Single Value Attribute in ER Diagram



Multivalued Attribute

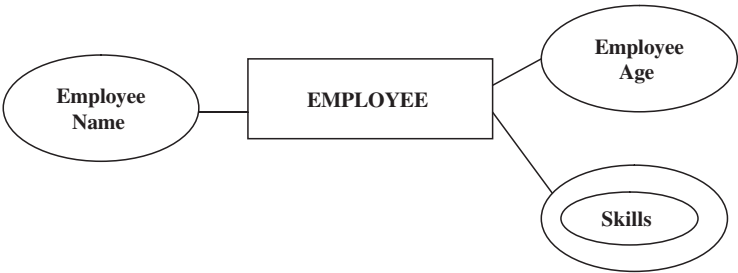
In the case of multivalued attribute, more than one value will be associated with that attribute.

Representation of Multivalued Attribute in ER Diagram

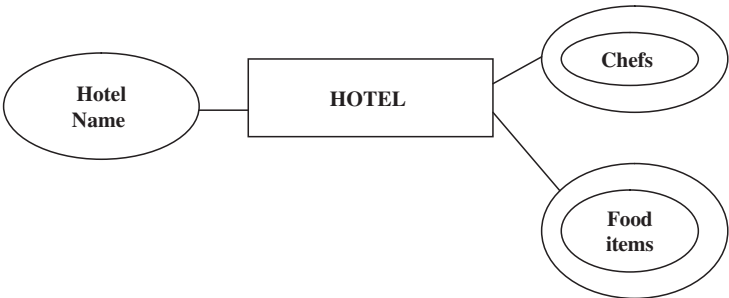


Examples of Multivalued Attribute

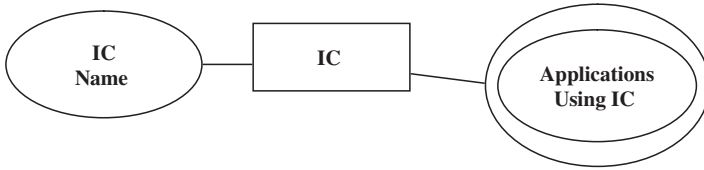
- 1. Consider an entity EMPLOYEE. An Employee can have many skills; hence skills associated to an employee are a multivalued attribute.



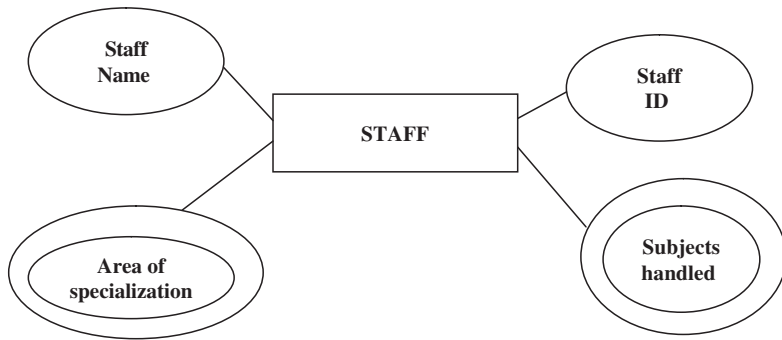
- 2. Number of chefs in a hotel is an example of multivalued attribute. Moreover, a hotel will have variety of food items. Hence food items associated with the entity HOTEL is an example of multivalued attribute.



3. Application associated with an IC (Integrated Circuit). An IC can be used for several applications. Here IC stands for Integrated Circuit.



4. Subjects handled by a staff. A staff can handle more than one subject in a particular semester; hence it is an example of multivalued attribute.



Moreover a staff can be an expert in more than one area, hence area of specialization is considered as multivalued attribute.

Derived Attribute

The value of the derived attribute can be derived from the values of other related attributes or entities.

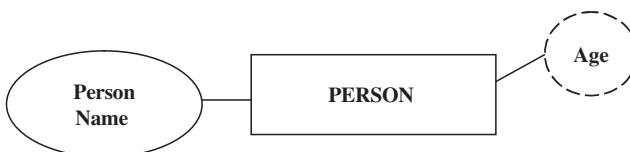
In ER diagram, the derived attribute is represented by dotted ellipse.

Representation of Derived Attribute in ER Diagram

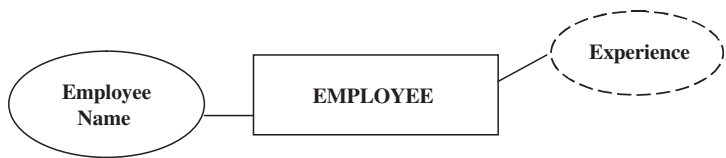


Example of Derived Attribute

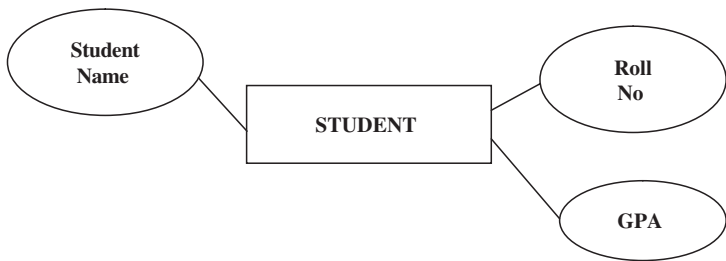
1. Age of a person can be derived from the date of birth of the person. In this example, age is the derived attribute.



- 2. Experience of an employee in an organization can be derived from date of joining of the employee.

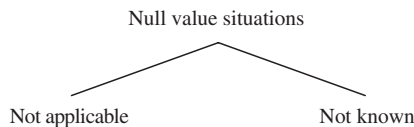


- 3. CGPA of a student can be derived from GPA (Grade Point Average).



Null Value Attribute

In some cases, a particular entity may not have any applicable value for an attribute. For such situation, a special value called null value is created.



Example

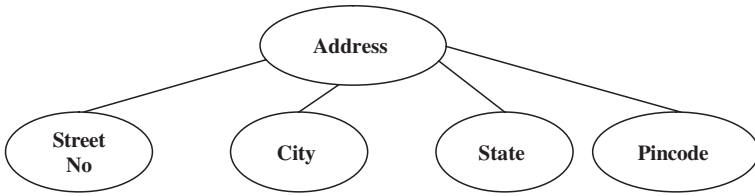
In application forms, there is one column called phone no. if a person do not have phone then a null value is entered in that column.

Composite Attribute

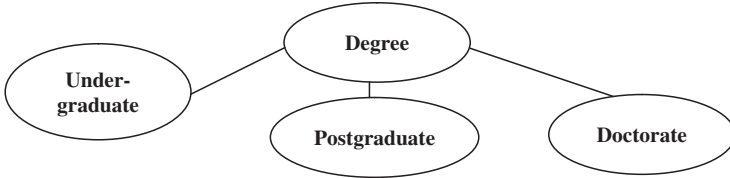
Composite attribute is one which can be further subdivided into simple attributes.

Example

Consider the attribute “address” which can be further subdivided into Street name, City, and State.



As another example of composite attribute consider the degrees earned by a particular scholar, which can range from undergraduate, postgraduate, doctorate degree, etc. Hence degree can be considered as composite attribute.

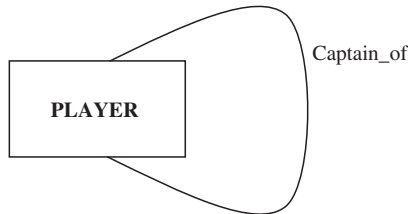


2.5 Relationship Degree

Relationship degree refers to the number of associated entities. The relationship degree can be broadly classified into unary, binary, and ternary relationship.

2.5.1 Unary Relationship

The unary relationship is otherwise known as recursive relationship. In the unary relationship the number of associated entity is one. An entity related to itself is known as recursive relationship.

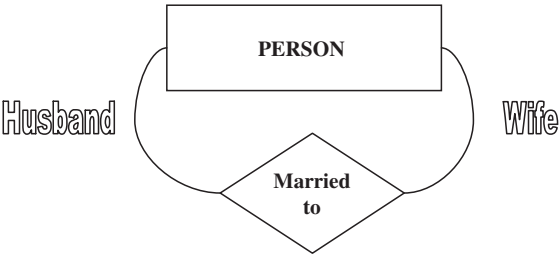


Roles and Recursive Relation

When an entity sets appear in more than one relationship, it is useful to add labels to connecting lines. These labels are called as roles.

Example

In this example, Husband and wife are referred as roles.



2.5.2 Binary Relationship

In a binary relationship, two entities are involved. Consider the example; each staff will be assigned to a particular department. Here the two entities are STAFF and DEPARTMENT.

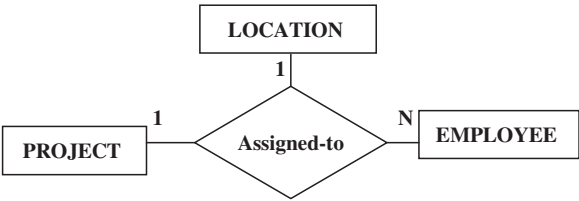


2.5.3 Ternary Relationship

In a ternary relationship, three entities are simultaneously involved. Ternary relationships are required when binary relationships are not sufficient to accurately describe the semantics of an association among three entities.

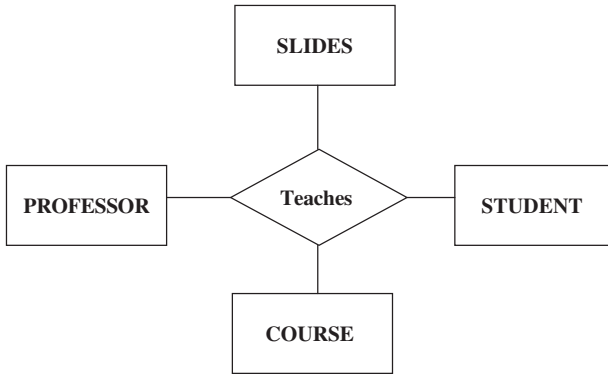
Example

Consider the example of employee assigned a project. Here we are considering three entities EMPLOYEE, PROJECT, and LOCATION. The relationship is “assigned-to.” Many employees will be assigned to one project hence it is an example of one-to-many relationship.



2.5.4 Quaternary Relationships

Quaternary relationships involve four entities. The example of quaternary relationship is “A professor teaches a course to students using slides.” Here the four entities are PROFESSOR, SLIDES, COURSE, and STUDENT. The relationships between the entities are “Teaches.”



2.6 Relationship Classification

Relationship is an association among one or more entities. This relationship can be broadly classified into one-to-one relation, one-to-many relation, many-to-many relation and recursive relation.

2.6.1 One-to-Many Relationship Type

The relationship that associates one entity to more than one entity is called one-to-many relationship. Example of one-to-many relationship is Country having states. For one country there can be more than one state hence it is an example of one-to-many relationship. Another example of one-to-many relationship is parent-child relationship. For one parent there can be more than one child. Hence it is an example of one-to-many relationship.




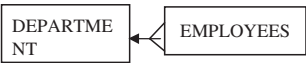
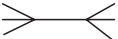
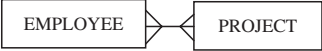

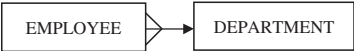
2.6.2 One-to-One Relationship Type

One-to-one relationship is a special case of one-to-many relationship. True one-to-one relationship is rare. The relationship between the President and the country is an example of one-to-one relationship. For a particular country there will be only one President. In general, a country will not have more than one President hence the relationship between the country and the President is an example of one-to-one relationship. Another example of one-to-one relationship is House to Location. A house is obviously in only one location.

2.6.3 Many-to-Many Relationship Type

The relationship between EMPLOYEE entity and PROJECT entity is an example of many-to-many relationship. Many employees will be working in many projects hence the relationship between employee and project is many-to-many relationship.

Table 2.1. Relationship types

Relationship type	Representation	Example
One-to-one		
One-to-many		
Many-to-many		
Many-to-one		

2.6.4 Many-to-One Relationship Type

The relationship between EMPLOYEE and DEPARTMENT is an example of many-to-one relationship. There may be many EMPLOYEES working in one DEPARTMENT. Hence relationship between EMPLOYEE and DEPARTMENT is many-to-one relationship. The four relationship types are summarized and shown in Table 2.1.

2.7 Reducing ER Diagram to Tables

To implement the database, it is necessary to use the relational model. There is a simple way of mapping from ER model to the relational model. There is almost one-to-one correspondence between ER constructs and the relational ones.

2.7.1 Mapping Algorithm

The mapping algorithm gives the procedure to map ER diagram to tables. The rules in mapping algorithm are given as:

- For each strong entity type say E, create a new table. The columns of the table are the attribute of the entity type E.
- For each weak entity W that is associated with only one 1–1 identifying owner relationship, identify the table T of the owner entity type. Include as columns of T, all the simple attributes and simple components of the composite attributes of W.
- For each weak entity W that is associated with a 1–N or M–N identifying relationship, or participates in more than one relationship, create a new table T and include as its columns, all the simple attributes and simple components of the composite attributes of W. Also form its primary key by including as a foreign key in R, the primary key of its owner entity.

- For each binary 1–1 relationship type R, identify the tables S and T of the participating entity types. Choose S, preferably the one with total participation. Include as foreign key in S, the primary key of T. Include as columns of S, all the simple attributes and simple components of the composite attributes of R.
- For each binary 1–N relationship type R, identify the table S, which is at N side and T of the participating entities. Include as a foreign key in S, the primary key of T. Also include as columns of S, all the simple attributes and simple components of composite attributes of R.
- For each M–N relationship type R, create a new table T and include as columns of T, all the simple attributes and simple components of composite attributes of R. Include as foreign keys, the primary keys of the participating entity types. Specify as the primary key of T, the list of foreign keys.
- For each multivalued attribute, create a new table T and include as columns of T, the simple attribute or simple components of the attribute A. Include as foreign key, the primary key of the entity or relationship type that has A. Specify as the primary key of T, the foreign key and the columns corresponding to A.

Regular Entity

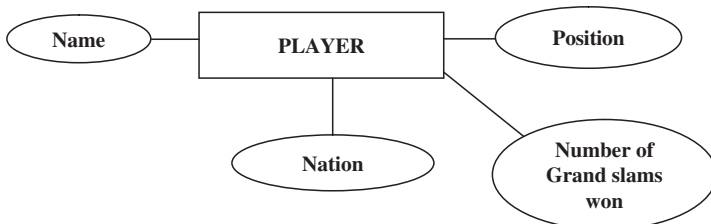
Regular entities are entities that have an independent existence and generally represent real-world objects such as persons and products. Regular entities are represented by rectangles with a single line.

2.7.2 Mapping Regular Entities

- Each regular entity type in an ER diagram is transformed into a relation. The name given to the relation is generally the same as the entity type.
- Each simple attribute of the entity type becomes an attribute of the relation.
- The identifier of the entity type becomes the primary key of the corresponding relation.

Example 1

Mapping regular entity type tennis player



This diagram is converted into corresponding table as

Player Name	Nation	Position	Number of Grand slams won
Roger Federer	Switzerland	1	5
Roddick	USA	2	4

Here,

- **Entity name = Name of the relation or table.**

In our example, the entity name is PLAYER which is the name of the table

- **Attributes of ER diagram = Column name of the table.**

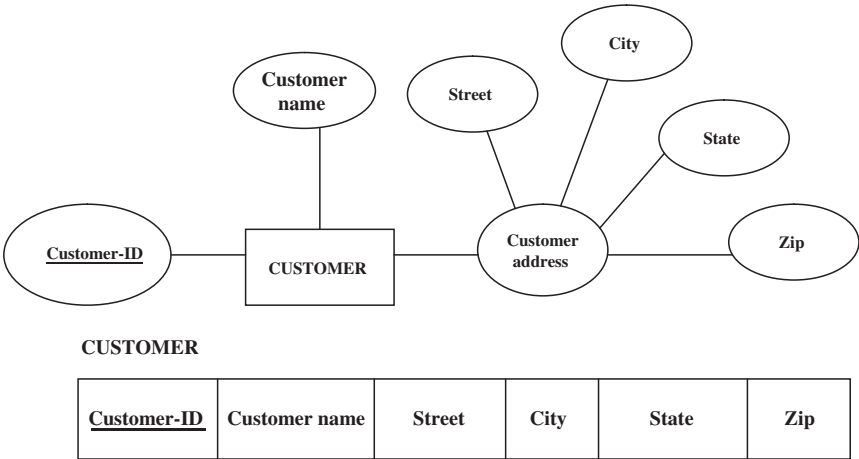
In our example the Name, Nation, Position, and Number of Grand slams won which forms the column of the table.

2.7.3 Converting Composite Attribute in an ER Diagram to Tables

When a regular entity type has a composite attribute, only the simple component attributes of the composite attribute are included in the relation.

Example

In this example the composite attribute is the Customer address, which consists of Street, City, State, and Zip.



When the regular entity type contains a multivalued attribute, two new relations are created.

The first relation contains all of the attributes of the entity type except the multivalued attribute.

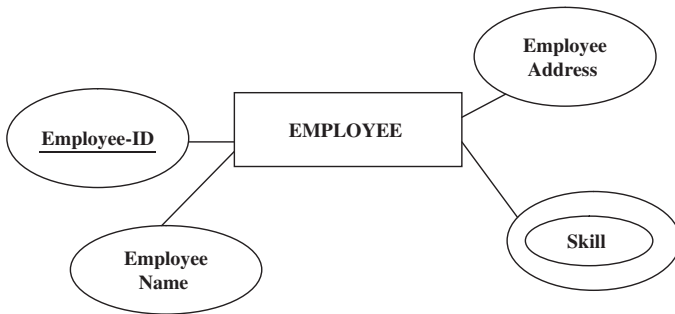
The second relation contains two attributes that form the primary key of the second relation. The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation. The second is the multivalued attribute.

2.7.4 Mapping Multivalued Attributes in ER Diagram to Tables

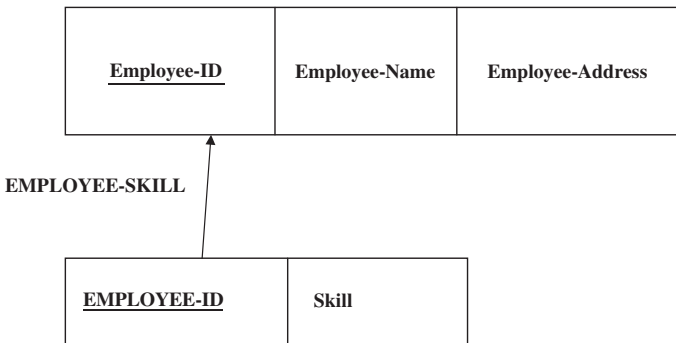
A multivalued attribute is having more than one value. One way to map a multivalued attribute is to create two tables.

Example

In this example, the skill associated with the EMPLOYEE is a multivalued attribute, since an EMPLOYEE can have more than one skill as fitter, electrician, turner, etc.



EMPLOYEE

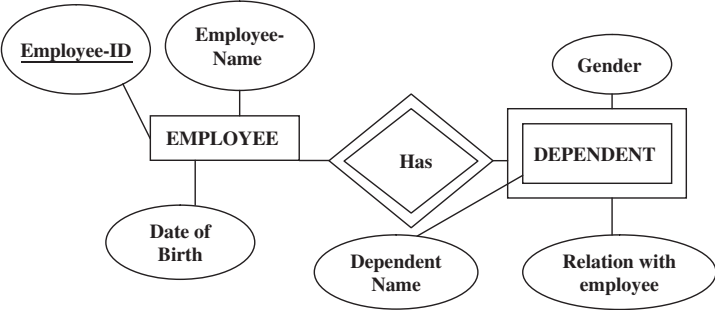


2.7.5 Converting “Weak Entities” in ER Diagram to Tables

Weak entity type does not have an independent existence and it exists only through an identifying relationship with another entity type called the owner.

For each weak entity type, create a new relation and include all of the simple attributes as attributes of the relation. Then include the primary key of the identifying relation as a foreign key attribute to this new relation.

The primary key of the new relation is the combination of the primary key of the identifying and the partial identifier of the weak entity type. In this example DEPENDENT is weak entity.



The corresponding table is given by

EMPLOYEE			
<u>Employee-ID</u>	Employee-Name	Date of Birth	

DEPENDENT			
Dependent-Name	Gender	<u>Employee-ID</u>	Relation with Employee

2.7.6 Converting Binary Relationship to Table

A relationship which involves two entities can be termed as binary relationship. This binary relationship can be one-to-one, one-to-many, many-to-one, and many-to-many.

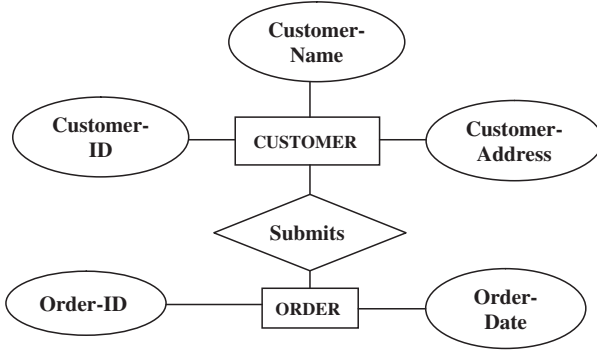
Mapping one-to-Many Relationship

For each 1–M relationship, first create a relation for each of the two entity type’s participation in the relationship.

Example

One customer can give many orders. Hence the relationship between the two entities CUSTOMER and ORDER is one-to-many relationship. In one-to-many relationship, include the primary key attribute of the entity on the

one-side of the relationship as a foreign key in the relation that is on the many side of the relationship.



Here we have two entities CUSTOMER and ORDER. The relationship between CUSTOMER and ORDER is one-to-many. For two entities CUSTOMER and ORDER, two tables namely CUSTOMER and ORDER are created as shown later. The primary key CUSTOMER.ID in the CUSTOMER relation becomes the foreign key in the ORDER relation.

CUSTOMER

<u>Customer-ID</u>	Customer-Name	Customer-Address
--------------------	---------------	------------------

ORDER

<u>Order-ID</u>	Order-Date	Customer-ID
-----------------	------------	-------------

Binary one-to-one relationship can be viewed as a special case of one-to-many relationships.

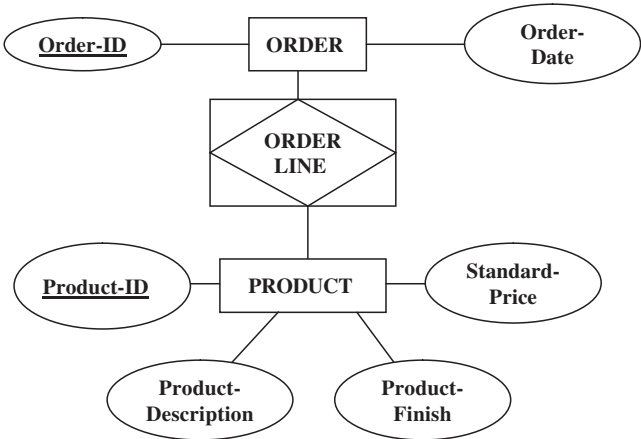
The process of mapping one-to-one relationship requires two steps. First, two relations are created, one for each of the participating entity types. Second, the primary key of one of the relations is included as a foreign key in the other relation.

2.7.7 Mapping Associative Entity to Tables

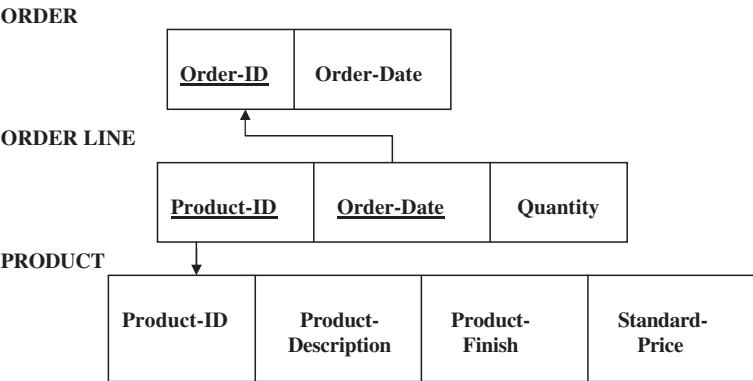
Many-to-many relationship can be modeled as an associative entity in the ER diagram.

Example 1. (Without Identifier)

Here the associative entity is ORDERLINE, which is without an identifier. That is the associative entity ORDERLINE is without any key attribute.



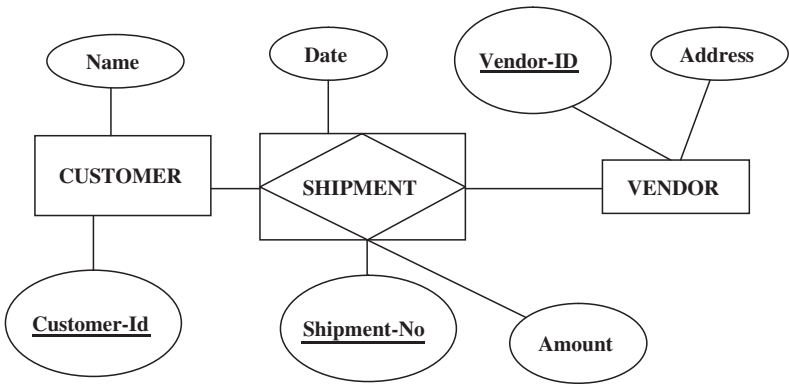
The first step is to create three relations, one for each of the two participating entity types and the third for the associative entity. The relation formed from the associative entity is associative relation.



Example 2. (With Identifier)

Sometimes data models will assign an identifier (surrogate identifier) to the associative entity type on the ER diagram. There are two reasons to motivate this approach:

- 1. The associative entity type has a natural identifier that is familiar to end user.
- 2. The default identifier may not uniquely identify instances of the associative entity.



- Shipment-No is a natural identifier to end user.
- The default identifier consisting of the combination of Customer-ID and Vendor-ID does not uniquely identify the instances of SHIPMENT.

CUSTOMER

Customer-ID	Name	Other Attributes
-------------	------	------------------

SHIPMENT

Shipment-No	Customer-ID	Vendor-ID	Date	Amount
-------------	-------------	-----------	------	--------

VENDOR

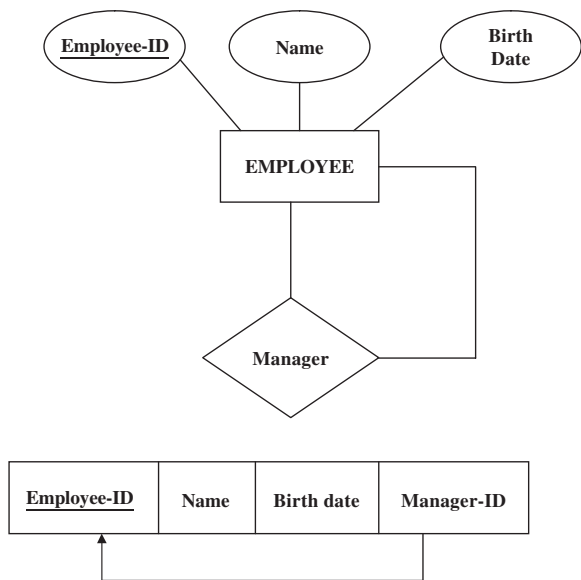
Vendor-ID	Address	Other Attributes
-----------	---------	------------------

2.7.8 Converting Unary Relationship to Tables

Unary relationships are also called recursive relationships. The two most important cases of unary relationship are one-to-many and many-to-many.

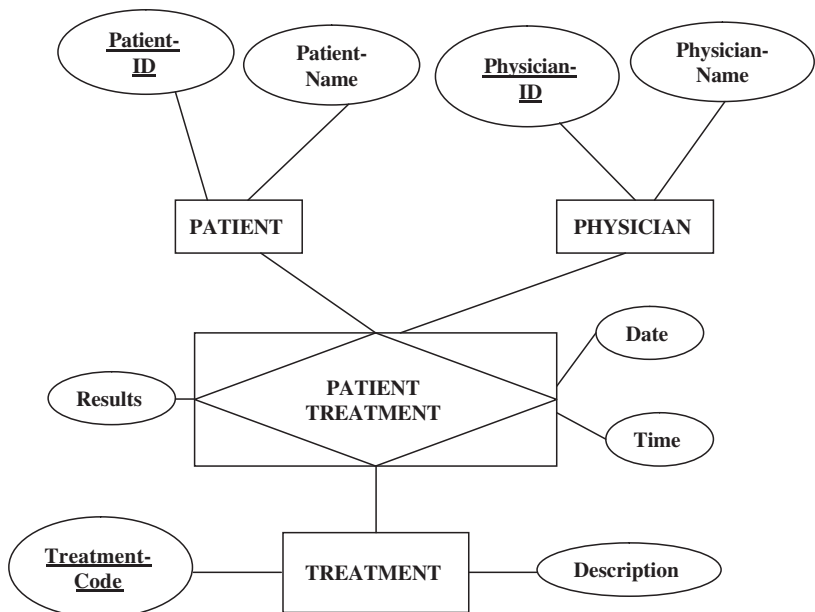
One-to-many Unary Relationship

Each employee has exactly one manager. A given employee may manage zero to many employees. The foreign key in the relation is named Manager-ID. This attribute has the same domain as the primary key Employee-ID.



2.7.9 Converting Ternary Relationship to Tables

A ternary relationship is a relationship among three entity types. The three entities given in this example are **PATIENT**, **PHYSICIAN**, and **TREATMENT**. The **PATIENT–TREATMENT** is an associative entity.



The primary key attributes – Patient ID, Physician ID, and Treatment Code – become foreign keys in PATIENT TREATMENT. These attributes are components of the primary key of PATIENT TREATMENT.

PATIENT TREATMENT

<u>Patient-ID</u>	Patient-Name
-------------------	--------------

PHYSICIAN

<u>Physician-ID</u>	Physician-Name
---------------------	----------------

PATIENT TREATMENT

<u>Patient-ID</u>	<u>Physician-ID</u>	<u>Treatment-Code</u>	<u>Date</u>	<u>Time</u>	Results
-------------------	---------------------	-----------------------	-------------	-------------	---------

TREATMENT

<u>Treatment-Code</u>	Description
-----------------------	-------------

2.8 Enhanced Entity–Relationship Model (EER Model)

The basic concepts of ER modeling are not powerful enough for some complex applications. Hence some additional semantic modeling concepts are required, which are being provided by Enhanced ER model. The Enhanced ER model is the extension of the original ER model with new modeling constructs. The new modeling constructs introduced in the EER model are supertype (superclass)/subtype (subclass) relationships. The supertype allows us to model general entity type whereas the subtype allows us to model specialized entity types.

Enhanced ER model = ER model + hierarchical relationships.

EER modeling is especially useful when the domain being modeled is object-oriented in nature and the use of inheritance reduces the complexity of the design. The extended ER model extends the ER model to allow various types of abstraction to be included and to express constraints more clearly.

2.8.1 Supertype or Superclass

Supertype or superclass is a generic entity type that has a relationship with one or more subtypes. For example PLAYER is a generic entity type which has

a relationship with one or more subtypes like CRICKET PLAYER, FOOTBALL PLAYER, HOCKEY PLAYER, TENNIS PLAYER, etc.

2.8.2 Subtype or Subclass

A subtype or subclass is a subgrouping of the entities in an entity type that is meaningful to the organization. A subclass entity type is a specialized type of superclass entity type. A subclass entity type represents a subset or subgrouping of superclass entity type’s instances. Subtypes inherit the attributes and relationships associated with their supertype.

Consider the entity type ENGINE, which has two subtypes PETROL ENGINE and DIESEL ENGINE.

Consider the entity type STUDENT, which has two subtypes UNDERGRADUATE and POSTGRADUATE.

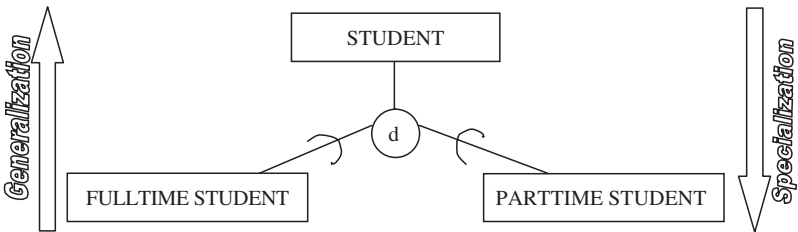
2.9 Generalization and Specialization

Generalization and specialization are two words for the same concept, viewed from two opposite directions. Generalization is the bottom-up process of defining a generalized entity type from a set of more specialized entity types. Specialization is the top-down process of defining one or more subtypes of a supertype.

Generalization is the process of minimizing the differences between entities by identifying common features. It can also be defined as the process of defining a generalized entity type from a set of entity types.

Specialization is a process of identifying subsets of an entity set (the superset) that share some distinguishing characteristics. In specialization the superclass is defined first and the subclasses are defined next. Specialization is the process of viewing an object as a more refined, specialized object. Specialization emphasizes the differences between objects.

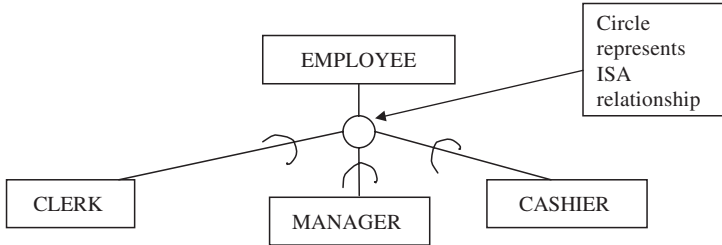
For example consider the entity type STUDENT, which can be further classified into FULLTIME STUDENT and PARTTIME STUDENT. The classification of STUDENT into FULLTIME STUDENT and PARTTIME STUDENT is called Specialization.



2.10 ISA Relationship and Attribute Inheritance

IS_A relationship supports attribute inheritance and relationship participation. In the EER diagram, the subclass relationship is represented by ISA relationship. Attribute inheritance is the property by which subclass entities inherit values for all attributes of the superclass.

Consider the example of EMPLOYEE entity set in a bank. The EMPLOYEE in a bank can be CLERK, MANAGER, CASHIER, ACCOUNTANT, etc. It is to be observed that the CLERK, MANAGER, CASHIER, ACCOUNTANT inherit some of the attributes of the EMPLOYEE.



In this example the superclass is EMPLOYEE and the subclasses are CLERK, MANAGER, and CASHIER. The subclasses inherit the attributes of the superclass. Since each member of the subclass is an ISA member of the superclass, the circle below the EMPLOYEE entity set represents ISA relationship.

2.11 Multiple Inheritance

A subclass with more than one superclass is called a shared subclass. A subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclass, that is it has multiple inheritance from its superclasses. In multiple inheritance a subclass can be subclass of more than one superclass.

Example of Multiple Inheritance

Consider a person in an educational institution. The person can be employee, alumnus, and student. The employee entity can be staff or faculty. The student can be a graduate student or a postgraduate student. The postgraduate student can be a teaching assistant. If the postgraduate student is a teaching assistant, then he/she inherits the characteristics of the faculty as well as student class. That is the teaching assistant subclass is a subclass of more than one superclass (faculty, student). This phenomenon is called multiple inheritance and is shown in the Fig. 2.2.

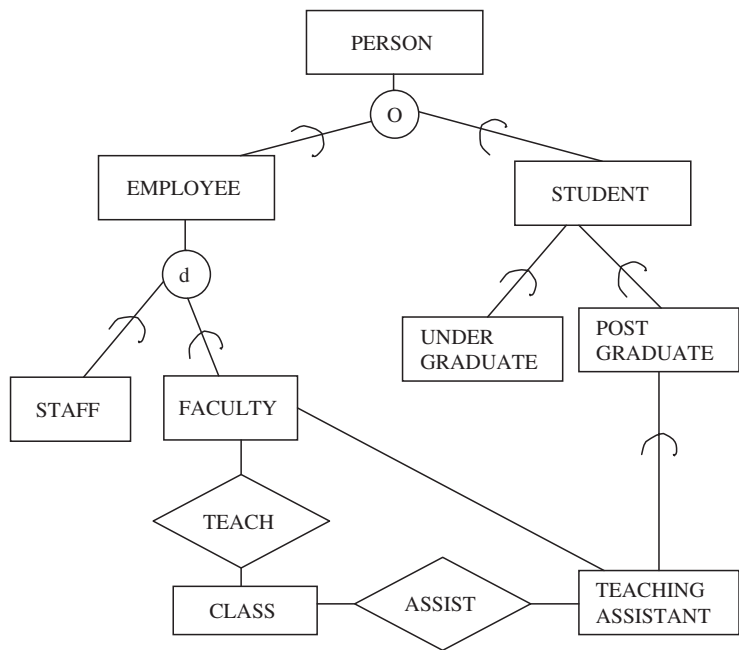


Fig. 2.2. Multiple inheritance

2.12 Constraints on Specialization and Generalization

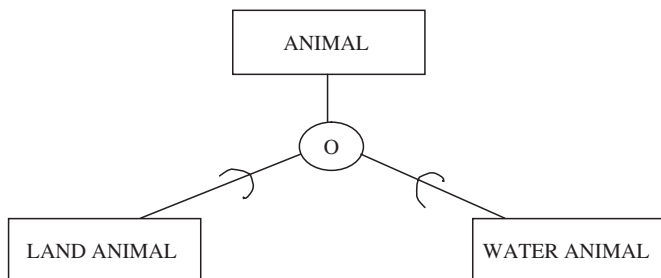
The constraints on specialization and generalization can be broadly classified into disjointness and completeness. The disjointness constraint allows us to specify whether an instance of a supertype may simultaneously be a member of two or more subtypes. In disjointness we have two categories (1) Overlap and (2) Disjoint. In completeness we have two categories (1) Total and (2) Partial. The completeness constraint addresses the question whether an instance of a supertype must also be a member of at least one subtype.

2.12.1 Overlap Constraint

Overlap refers to the fact that the same entity instance may be a member of more than one subclass of the specialization.

Example of Overlap Constraint

Consider the example of ANIMAL entity, which can be further subdivided into LAND ANIMAL and WATER ANIMAL. Consider the example of Frog and Crocodile which can live in both land and water hence the division of ANIMAL into LAND and WATER animals is an example of overlap constraint.

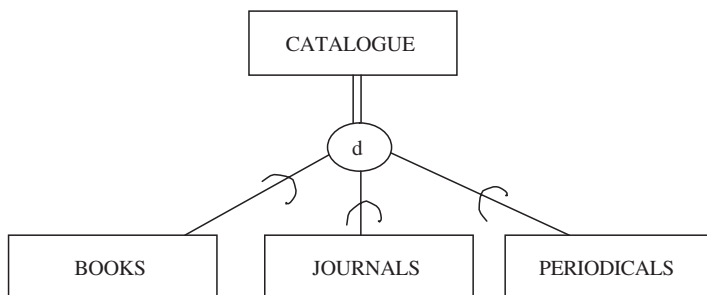


2.12.2 Disjoint Constraint

Disjoint refers to the fact that the same entity instance may be a member of only one subclass of the specialization.

Example of Disjointness Constraint

Consider the example of CATALOGUE. The CATALOGUE is a superclass, which can be further subdivided into BOOKS, JOURNALS, and PERIODICALS. This falls under disjointness because a BOOK entity can be neither JOURNAL nor PERIODICAL.

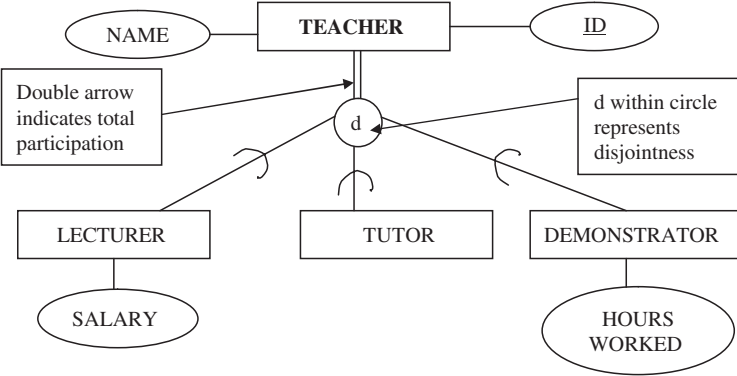


2.12.3 Total Specialization

Total completeness refers to the fact that every entity instance in the super-class must be a member of some subclass in the specialization. With total specialization, an instance of the supertype must be a member of at least one subtype.

Example of Total Specialization

Consider the example of TEACHER; the teacher is a general term, which can be further specialized into LECTURER, TUTOR, and DEMONSTRATOR. Here every member in the superclass participates as a member of a subclass, hence it is an example of total participation.

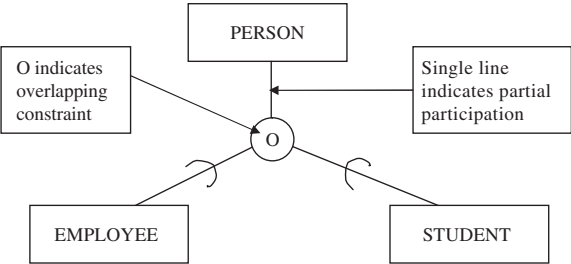


2.12.4 Partial Specialization

Partial completeness refers to the fact that an entity instance in the superclass need not be a member of any subclass in the specialization. With partial specialization, an instance of a supertype may or may not be a member of any subtype.

Example of Partial Specialization

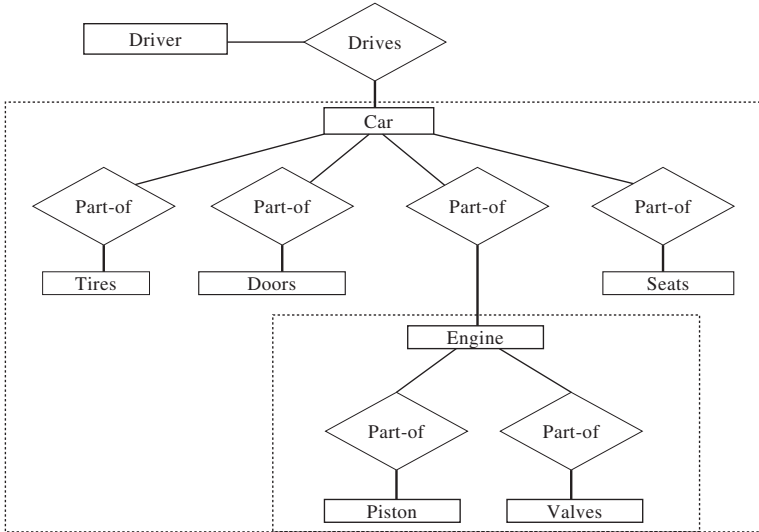
Consider the **PERSON** specialization into **EMPLOYEE** and **STUDENT**. This is an example of partial specialization because there can be a person who is unemployed and does not study.



2.13 Aggregation and Composition

Relationships among relationships are not supported by the ER model. Groups of entities and relationships can be abstracted into higher level entities using aggregation. Aggregation represents a “HAS-A” or “IS-PART-OF” relationship between entity types. One entity type is the whole, the other is the part. Aggregation allows us to indicate that a relationship set participates in another relationship set.

Consider the example of a driver driving a car. The car has various components like tires, doors, engine, seat, etc., which varies from one car to another. Relationship drives is insufficient to model the complexity of this system. *Part-of* relationships allow abstraction into higher level entities. In this example engine, tires, doors, and seats are aggregated into car.



Composition is a stronger form of aggregation where the part cannot exist without its containing whole entity type and the part can only be part of one entity type.

Consider the example of DEPARTMENT has PROJECT. Each project is associated with a particular DEPARTMENT. There cannot be a PROJECT without DEPARTMENT. Hence DEPARTMENT has PROJECT is an example of composition.

2.14 Entity Clusters

EER diagrams are difficult to read when there are many entities and relationships. One possible solution is to group entities and relationships into entity clusters. Entity cluster is a set of one or more entity types and associated relationships grouped into a single abstract entity type. Entity cluster behaves like an entity type; hence entity clusters and entity types can be further grouped to form a higher level entity cluster. Entity clustering is a hierarchical decomposition of a macrolevel view of the data model into finer and finer views, eventually resulting in the full detailed data model.

To understand entity cluster, consider the example of Hospital Management. In hospital, the DOCTORS treat the PATIENT. The DOCTORS are paid by the MANAGEMENT which builds buildings. The DOCTORS can

be either general physician or specialist like those with MS or MD. The patient can be either inpatient or outpatient. It is to be noted that only outpatient will be allotted bed. If we have to represent the earlier ideas, it can be done using EER diagram as shown in Fig. 2.3. The EER diagram is found to be complex; the same idea is represented using Entity Clusters as shown in Fig. 2.4. Here the DOCTOR specialization is clustered into DOCTORS entity and the PATIENT specialization is clustered into simply PATIENT. At the first glance, it may look like reduction of EER model to ER model, but it is not so. Here the entities as well as relationships are clustered into simply entity set.

2.15 Connection Traps

Connection trap is the misinterpretation of the meaning of certain relationships. This connection traps can be broadly classified into fan and chasm trap. Any conceptual model will contain potential connection traps. An error in the interpretation of the meaning of the relationship may cause the database to be incapable of storing certain information. Both the fan and chasm trap arise when the relationships appear to exist between entity types, but the links between occurrences may be ambiguous or not exist. Related groups of entities could become clusters.

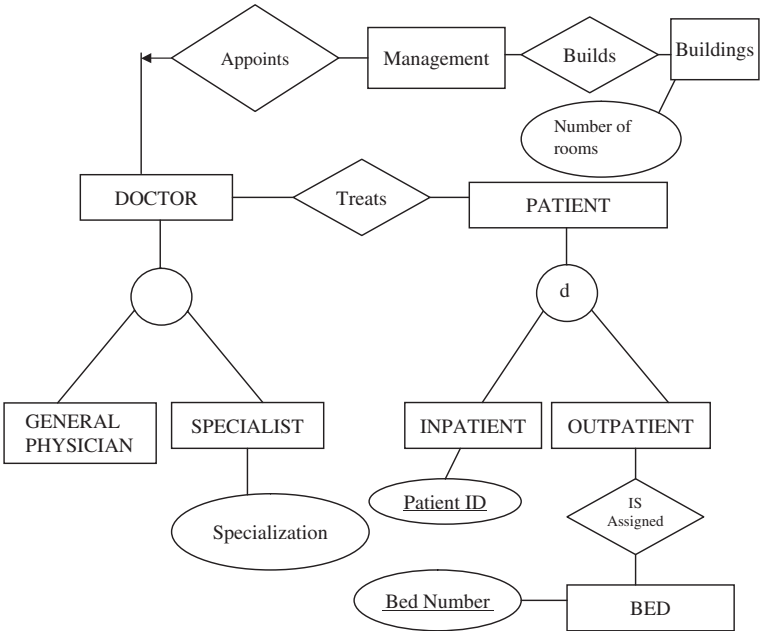


Fig. 2.3. EER diagram of Hospital Management

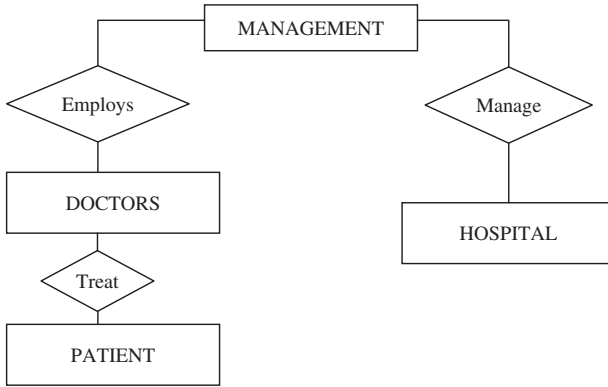


Fig. 2.4. Entity Cluster

2.15.1 Fan Trap

Fan trap occurs when the model represents a relationship between entity types but the pathway between certain entity occurrences is ambiguous. Fan trap occurs when 1–M relationships fan out from a single entity. In order to understand the concept of Fan trap, consider the following example

Contractor works in a team. Statement (1)
 Team develops projects. Statement (2)

Statement (1) represents M–1 relationship. Statement (2) represents 1–M relationship. But the information about which contractors are involved in developing which projects is not clear.

Consider another example of Fan trap.

Department is on Site. Statement (1)
 Site employs Staff. Statement (2)

Statement (1) represents M–1 relationship, because many departments may be in a single site. Statement (2) represents 1–M relationships. However which staff works in a particular department is ambiguous. The fan trap is resolved by reconstructing the original ER model to represent the correct association.



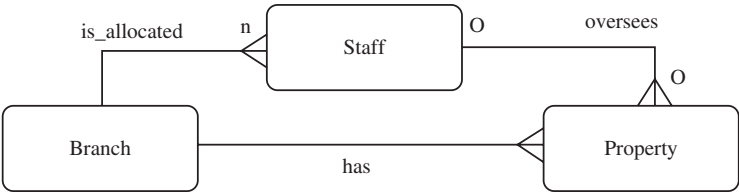
2.15.2 Chasm Trap

A chasm trap occurs when a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity

occurrences. It occurs where there is a relationship with partial participation, which forms part of the pathway between entities that are related. Consider the relationship shown later.



A single branch may be allocated to many staff who oversees the management of properties for rent. It should be noted that not all staff oversee property and not all property is managed by a member of staff. Hence there exist a partial participation of Staff and Property in the relation “oversees,” which means that some properties cannot be associated with a branch office through a member of staff. Hence the model has to modified as shown later.



2.16 Advantages of ER Modeling

An ER model is derived from business specifications. ER models separate the information required by a business from the activities performed within a business. Although business can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant. The advantages of ER modeling are summarized later:

1. The ER modeling provides an easily understood pictorial map for the database design.
2. It is possible to represent the real world problems in a better manner in ER modeling.
3. The conversion of ER model to relational model is straightforward.
4. The enhanced ER model provides more flexibility in modeling real world problems.
5. The symbols used to represent entity and relationships between entities are simple and easy to follow.

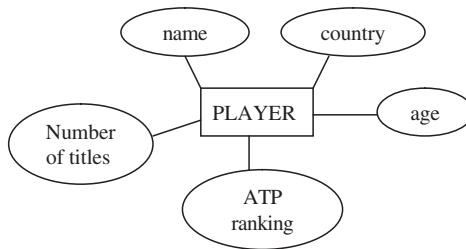
Summary

This chapter has described the fundamentals of ER modeling of data. An ER model is a logical representation of data. The ER model was introduced

by Peter Chen in 1976. An ER model is usually expressed in the form of ER diagram. The basic constructs of ER model are entity types, relationships, and attributes. This chapter also described the types of entities like strong and weak entity, types of relationships like one-to-one, one-to-many, and many-to-many relationship. Attributes can also be classified as single valued, multivalued and derived attribute. In this chapter different types of entities, attributes, and relationship were explained with simple examples.

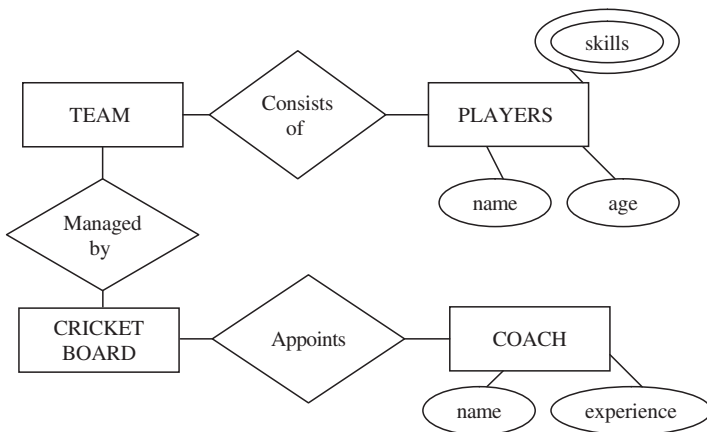
Review Questions

2.1. Construct an ER diagram of tennis player.



2.2. Construct an ER diagram of Indian cricket team.

One way of constructing ER diagram for Indian cricket team is shown later.



Here skills refers to player's skill which may be batting, bowling, and fielding. All-rounders can have many skills.

2.3. What is Weak entity type?

Entity types that do not have key attribute of their own are called Weak entity type.

2.4. Define entity with example?

An entity is an object with a physical existence.
Examples of entity is a person, a car, an organization, a house, etc.

2.5. Define Entity type, Entity set?

An entity type defines a collection of entities that have same attribute
Entity Set
Entity set is the collection of a particular entity type that are grouped into an “Entity Set.”

2.6. Should a real world object be modeled as an entity or as an attribute?

Object should be an entity if a number of attributes could be associated with it for proper identification and description, either now or later. Object should be an attribute, if it has an atomic nature. For example, Color should be an attribute, unless we identify Color either as a process (e.g., painting) where a number of attributes codes are to be recorded (e.g., type, shade, gray-scale, manufacturer, or as an object with properties (e.g., car-color with details).

2.7. When composite attribute usage is preferred than set of attributes?

Composite attribute is chosen when a meaningful name can be assigned to the set of attributes, e.g., data, address. Otherwise a set of simple attributes should be chosen.

2.8. Distinguish between strong and weak entity?

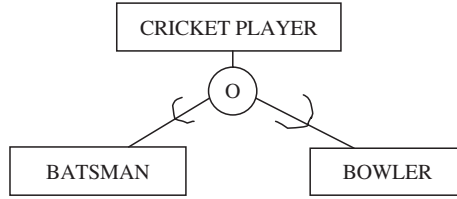
Strong entity	Weak entity
Exists independently of other entities	Dependent on a strong entity, cannot exist on its own
Strong entity has its own unique identifier	Does not have a unique identifier
Represented by a single line rectangle in ER diagram	Represented with a double-line rectangle in ER diagram

2.9. What is inheritance in generalization hierarchies?

Inheritance is a data modeling feature that supports sharing of attributes between a supertype and a subtype. Subtype inherits attributes from their supertype.

2.10. Give an example of supertype/subtype relationship where the overlap rule applies?

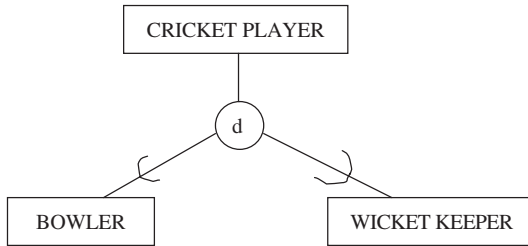
Overlap refers to the fact that the same entity instance may be a member of more than one subclass of the specialization. Consider the example of CRICKET PLAYER. Here CRICKET PLAYER is the supertype. The subtype can be BOWLER, BATSMAN.



Same player can be both batsman and bowler. Hence overlap rule holds good in this example.

2.11. Give an example of supertype/subtype relationship where the disjoint rule applies?

Let us consider the example of CRICKET PLAYER again. Here the super type is CRICKET PLAYER. The subtypes are BOWLER and WICKETKEEPER. We know that the same cricket player cannot be both bowler and wicket keeper hence disjoint rule applies for this example.



II. Match the following

- | | | |
|-----------------|-------|--|
| (1) Relation | _____ | (a) Rows |
| (2) Tuples | _____ | (b) Number of Rows of a Relation |
| (3) Cardinality | _____ | (c) Number of Columns of a Relation |
| (4) Degree | _____ | (d) Columns or Range of values a column may have |
| (5) Domain | _____ | (e) Table |

Answer

- (1) → (e)
 (2) → (a)
 (3) → (b)
 (4) → (c)
 (5) → (d)