# Chapter-1. Introduction to Database Fundamentals

A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE, Microsoft ACCESS, or EXCEL. This is a collection of related data with an implicit meaning and hence is a database.

Databases and database technology are having a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, engineering, medicine, law, education, and library science. The word *database* is in such common use that we must begin by defining a database.

A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the Universe of Discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

In other words, a database has some source from which data are derived, some degree of interaction with events in the real world, and an audience that is actively interested in the contents of the database.

- A database can be of any size and of varying complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure.
- On the other hand, the card catalog of a large library may contain half a million cards stored under different categories—by primary author's last name, by subject, by book title—with each category organized in alphabetic order.
- A database of even greater size and complexity is maintained by the Internal Revenue Service to keep track of the tax forms filed by U.S. taxpayers.

This huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed. A database may be generated and maintained manually or it may be computerized. The library card catalog is an example of a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

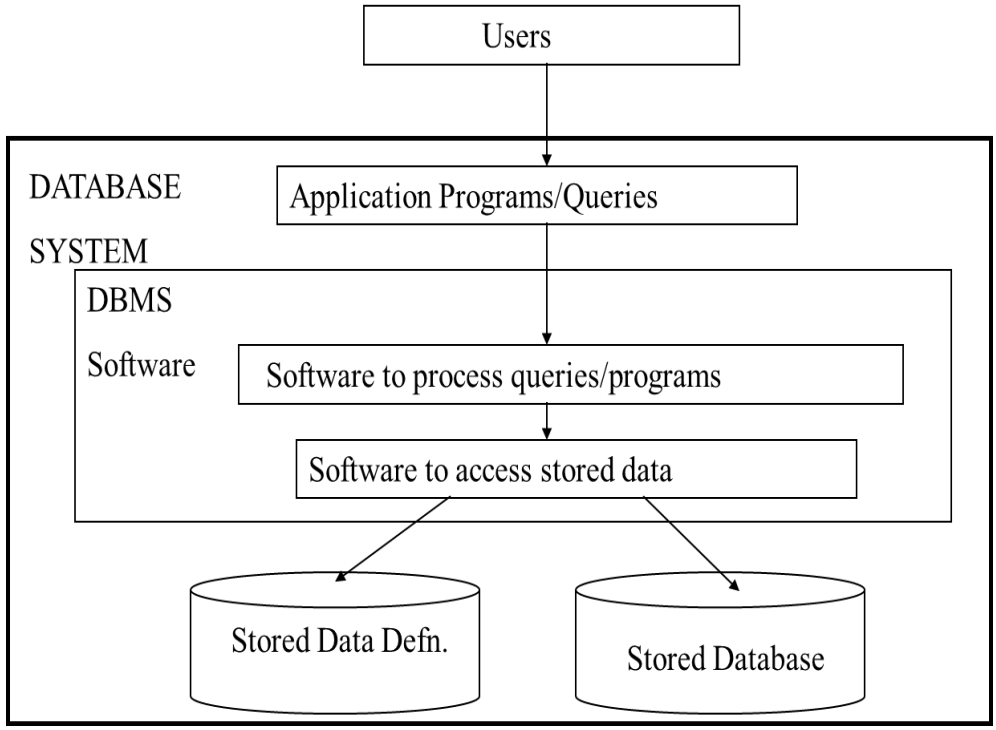
A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. Hence, the DBMS is considered as a *general-purpose software system* that facilitates the processes of *defining, constructing,* and *manipulating* databases for various applications.

Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database.

Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.

Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data.

It is not necessary to use general-purpose DBMS software to implement a computerized database. We could write our own set of programs to create and maintain the database, in effect creating our own *special-purpose* DBMS software. We will call the database and DBMS software together a database system.




A simplified database system environment :

What is a Database ?

To find out what database is, we have to start from data, which is the basic building block of any DBMS. In a database, data is organized strictly in row and column format. The rows are called Tuple or Record. The data items within one row may belong to different data types. On the

other hand, the columns are often called Domain or Attribute. All the data items within a single attribute are of the same data type.

Data: Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).

Record: Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |

Table or Relation: Collection of related records.

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

The columns of this relation are called Fields, Attributes or Domains. The rows are called Tuples or Records.

Database: Collection of related relations. Consider the following collection of tables:

T1

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

T2

| Roll | Address |
|---|---|
| 1 | KOL |
| 2 | DEL |
| 3 | MUM |

T3

| Roll | Year |
|---|---|
| 1 | I |
| 2 | II |
| 3 | I |

T4

| Year | Hostel |
|---|---|
| I | H1 |
| II | H2 |

We now have a collection of 4 tables. They can be called a “related collection”, because we can clearly find out that there are some common attributes existing in a selected pair of tables. Because of these common attributes we may combine the data of two or more tables together to find out the complete details of a student. Questions like “Which hostel does the youngest student live in?” can be answered now, although *Age* and *Hostel* attributes are in different tables.

What is Management System?

A management system is a set of rules and procedures which help us to create organize and manipulate the database. It also helps us to add, modify delete data items in the database. The management system can be either manual or computerized. It is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

## 1.1 Data Processing Vs. Data Management Systems

Although Data Processing and Data Management Systems both refer to functions that take raw data and transform it into usable information, the usage of the terms is very different.

Data Processing is the term generally used to describe what was done by large mainframe computers from the late 1940's until the early 1980's (and which continues to be done in most large organizations to a greater or lesser extent even today): large volumes of raw transaction data fed into programs that update a master file, with fixed-format reports written to paper.

The term Data Management Systems refers to an expansion of this concept, where the raw data, previously copied manually from paper to punched cards, and later into data-entry terminals, is now fed into the system from a variety of sources, including ATMs, EFT, and direct customer entry through the Internet. The master file concept has been largely displaced by database management systems, and static reporting replaced or augmented by ad-hoc reporting and direct inquiry, including downloading of data by customers. The ubiquity of the Internet and the Personal Computer have been the driving force in the transformation of Data Processing to the more global concept of Data Management Systems.

## 1.2 File Oriented Approach

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called file processing systems. In a typical file processing systems, each department has its own files, designed specifically for those applications. The department itself working with the data processing staff, sets policies or standards for the format and maintenance of its files.

Programs are dependent on the files and vice-versa; that is, when the physical format of the file is changed, the program has also to be changed. Although the traditional file oriented approach to information processing is still widely used, it does have some very important disadvantages.

This typical file-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) came along, organizations usually stored information in such systems.

Keeping organizational information in a file-processing system has a number of major disadvantages:

### 1.2.1 Disadvantages of File – Processing Systems :

### i) Data redundancy and inconsistency.

Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places

(files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

## ii) Difficulty in accessing data.

Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* customers. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. As expected, a program to generate such a list does not exist. Again, the officer has the preceding two options, neither of which is satisfactory.

The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

## iii) Data isolation.

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

## iv) Integrity problems.

The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

## v) Atomicity problems.

A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

### vi) Concurrent access anomalies.

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

### vii) Security problems.

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult.

These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems.

## 1.3 Characteristics of the Database Approach

i) Self-Describing Nature of a Database System

ii) Insulation between Programs and Data, and Data Abstraction

iii) Support of Multiple Views of the Data

iv) Sharing of Data and Multiuser Transaction Processing

### i) Self-Describing Nature of a Database System

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

The catalog is used by the DBMS software and also by database users who need information about the database structure. A general purpose DBMS software package is not

written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with *any number of database applications*—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

## ii) Insulation between Programs and Data, and Data Abstraction

The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence.

For example, a file access program may be written in such a way that it can access only STUDENT records of the structure. If we want to add another piece of data to each STUDENT record, say the Birthdate, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we just need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birthdate; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.

In object-oriented and object-relational databases, users can define operations on data as part of the database definitions. An operation (also called a *function*) is specified in two parts. The *interface* of an operation includes the operation name and the data types of its arguments (or parameters). The *implementation* of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

The characteristic that allows program-data independence and program-operation independence is called data abstraction. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model *hides* storage and implementation details that are not of interest to most database users.

## iii) Support of Multiple Views of the Data.

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

For example, one user of the database may be interested only in the transcript of each student. A second user, who is interested only in checking that students have taken all the prerequisites of each course they register for, may require the view.

## iv) Sharing of Data and Multiuser Transaction Processing.

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

For example, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger. These types of applications are generally called on-line transaction processing (OLTP) applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly.

## 1.4 Actors on the Scene (The People who works with the database):

Many persons are involved in the design, use, and maintenance of a large database with a few hundred users. Here, we identify the people whose jobs involve the day-to-day use of a large database; we call them the "actors on the scene." We consider people who may be called "workers behind the scene"—those who work to maintain the database system environment, but who are not actively interested in the database itself.

### *1.4.1 Database Administrators*

In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA).

The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.

### *1.4.2 Database Designers*

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users, in order to understand their requirements, and to come up with a design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed.

Database designers typically interact with each potential group of users and develop a view of the database that meets the data and processing requirements of this group. These views are then analyzed and *integrated* with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

*1.4.3 End Users*

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

i) *Casual end users* occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.

ii) *Naive or parametric end users* make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called canned transactions—that have been carefully programmed and tested. The tasks that such users perform are varied:

- Bank tellers check account balances and post withdrawals and deposits.
- Reservation clerks for airlines, hotels, and car rental companies check availability for a given request and make reservations.
- Clerks at receiving stations for courier mail enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.

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

iv) *Stand-alone users* maintain personal databases by using ready-made program packages that provide easy-to-use menu- or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

A typical DBMS provides multiple facilities to access a database.

- Naive end users need to learn very little about the facilities provided by the DBMS; they have to understand only the types of standard transactions designed and implemented for their use.
- Casual users learn only a few facilities that they may use repeatedly.
- Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements.

Stand-alone users typically become very proficient in using a specific software package.

*1.4.4 System Analysts and Application Programmers (Software Engineers)*

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements.

Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers (nowadays called software engineers) should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

## 1.5 Workers behind the Scene

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

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

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

- *Operators and maintenance personnel* are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

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

## 1.6 Advantages of Using a DBMS:

*1.6.1 Controlling Redundancy*

Redundancy means storing same data more than one. This redundancy in storing the same data multiple times leads to several problems.

First, there is the need to perform a single logical update—such as entering data on a new student—multiple times: once for each file where student data is recorded. This leads to *duplication of effort.*

Second, *storage space is wasted* when the same data is stored repeatedly, and this problem may be serious for large databases.

Third, files that represent the same data may become *inconsistent.* This may happen because an update is applied to some of the files but not to others. Even if an update—such as adding a new student—is applied to all the appropriate files, the data concerning the student may still be *inconsistent* since the updates are applied independently by each user group. For example, one user group may enter a student's birthdate erroneously as JAN-19-1974, whereas the other user groups may enter the correct value of JAN-29-1974.

A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated.

### *1.6.2 Restricting Unauthorized Access*

When multiple users share a database, it is likely that some users will not be authorized to access all information in the database. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically. Notice that we can apply similar controls to the DBMS software. For example, only the DBA's staff may be allowed to use certain privileged software, such as the software for creating new accounts. Similarly, parametric users may be allowed to access the database only through the canned transactions developed for their use.

### *1.6.3 Providing Persistent Storage for Program Objects and Data Structures*

Databases can be used to provide persistent storage for program objects and data structures. This is one of the main reasons for the emergence of the object-oriented database systems. Programming languages typically have complex data structures, such as record types in PASCAL or class definitions in C++.

The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so-called impedance mismatch problem, since the data structures provided by the DBMS were incompatible with the programming language's data structures. Object-oriented database systems typically offer data structure compatibility with one or more object-oriented programming languages.

### *1.6.4 Permitting Inferencing and Actions Using Rules*

Some database systems provide capabilities for defining *deduction rules* for *inferencing* new information from the stored database facts. Such systems are called deductive database systems. For example, there may be complex rules in the miniworld application for determining when a student is on probation. These can be specified *declaratively* as rules, which when compiled and maintained by the DBMS can determine all students on probation. In a traditional DBMS, an explicit *procedural program code* would have to be written to support such applications. But if the miniworld rules change, it is generally more convenient to change the declared

deduction rules than to recode procedural programs. More powerful functionality is provided by active database systems, which provide active rules that can automatically initiate actions.

### *1.6.5 Providing Multiple User Interfaces*

Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.

These include

- query languages for casual users;
- programming language interfaces for application programmers;
- forms and command codes for parametric users; and
- menu-driven interfaces and natural language interfaces for stand-alone users.

Both forms-style interfaces and menu-driven interfaces are commonly known as graphical user interfaces (GUIs). Many specialized languages and environments exist for specifying GUIs. Capabilities for providing World Wide Web access to a database—or web-enabling a database—are also becoming increasingly common.

### *1.6.6 Representing Complex Relationships Among Data*

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

### *1.6.7 Enforcing Integrity Constraints*

Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints.

- The simplest type of integrity constraint involves specifying a data type for each data item.
- A more complex type of constraint that occurs frequently involves specifying that a record in one file must be related to records in other files.
- Another type of constraint specifies uniqueness on data item values, such as "every course record must have a unique value for CourseNumber".

These constraints are derived from the meaning or semantics of the data and of the miniworld it represents. It is the database designers' responsibility to identify integrity constraints during database design. Some constraints can be specified to the DBMS and automatically enforced. Other constraints may have to be checked by update programs or at the time of data entry.

### *1.6.8 Providing Backup and Recovery*

A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the program

started executing. Alternatively, the recovery subsystem could ensure that the program is resumed from the point at which it was interrupted so that its full effect is recorded in the database.

## 1.7 Additional Implications of using the Database Approach

In addition to the above issues, there are other implications of using the database approach that can benefit most organizations.

### *1.7.1 Potential for Enforcing Standards*

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

### *1.7.2 Reduced Application Development Time*

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

### *1.7.3 Flexibility*

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

### *1.7.4 Availability of Up-to-Date Information*

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

### *1.7.5 Economies of Scale*

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

## 1.8 When Not to Use a DBMS

In spite of the advantages of using a DBMS, there are a few situations in which such a system may involve unnecessary overhead costs as that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training.
- Generality that a DBMS provides for defining and processing data.
- Overhead for providing security, concurrency control, recovery, and integrity functions.

Additional problems may arise if the database designers and DBA do not properly design the database or if the database systems applications are not implemented properly. Hence, it may be more desirable to use regular files under the following circumstances:

- The database and applications are simple, well defined, and not expected to change.
- There are stringent real-time requirements for some programs that may not be met because of DBMS overhead.
- Multiple-user access to data is not required.

## Review Questions

a) Define the following terms: *data, database, DBMS, database system, database catalog, program-data independence, user view, DBA, end user, canned transaction, deductive database system, persistent object, meta-data, transaction processing application.*

b) What three main types of actions involve databases? Briefly discuss each.

c) Discuss the main characteristics of the database approach and how it differs from traditional file systems.

d) What are the responsibilities of the DBA and the database designers?

e) What are the different types of database end users? Discuss the main activities of each.

f) Discuss the capabilities that should be provided by a DBMS.

# Chapter 2: Database System Concepts and Architecture

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package is one tightly integrated system, to the modern DBMS packages that are modular in design, with a client-server system architecture. This evolution mirrors the trends in computing, where the large centralized mainframe computers are being replaced by hundreds of distributed workstations and personal computers connected via communications networks. In a basic client-server architecture, the system functionality is distributed between two types of modules. A client module is typically designed so that it will run on a user workstation or personal computer. Typically, application programs and user interfaces that access the database run in the client module. Hence, the client module handles user interaction and provides the user-friendly interfaces such as forms or menu-based GUIs (*g*raphical *u*ser *i*nterfaces). The other kind of module, called a server module, typically handles data storage, access, search, and other functions.

## 2.1 Data Models, Schemas, and Instances

One fundamental characteristic of the database approach is that it provides some level of *data abstraction* by hiding details of data storage that are irrelevant to database users.

A data model —It is a collection of concepts that can be used to describe the conceptual/logical structure of a database. It provides the necessary means to achieve this abstraction.

- By *structure* is meant the data types, relationships, and constraints that should hold for the data.
- Most data models also include a set of basic operations for specifying retrievals/updates.
- Object-oriented data models include the idea of objects having behavior (i.e., applicable methods) being stored in the database (as opposed to purely "passive" data).

According to C.J. Date (one of the leading database experts), a data model is an abstract, self-contained, logical definition of the objects, operators, and so forth, that together constitute the *abstract machine* with which users interact. The objects allow us to model the *structure* of data; the operators allow us to model its *behavior*.

In the *relational* data model, data is viewed as being organized in two-dimensional tables comprised of tuples of attribute values. This model has operations such as Project, Select, and Join.

A data model is not to be confused with its implementation, which is a physical realization on a real machine of the components of the *abstract machine* that together constitute that model.

Logical vs. physical!!

There are other well-known data models that have been the basis for database systems. The best-known models pre-dating the relational model are the hierarchical (in which the entity types form a tree) and the network (in which the entity types and relationships between them form a graph).

## 2.1.1 Categories of Data Models

Many data models have been proposed, and we can categorize them according to the types of concepts they use to describe the database structure.

High-level or conceptual data models provide concepts that are close to the way many users perceive data.

Low-level or physical data models provide concepts that describe the details of how data is stored in the computer. Concepts provided by low-level data models are generally meant for computer specialists, not for typical end users.

Between these two extremes is a class of representational (or implementation) data models, which provide concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer.

Representational data models hide some details of data storage but can be implemented on a computer system in a direct way.

Conceptual data models use concepts such as entities, attributes, and relationships.

Entity : An entity represents a real-world object or concept, such as an employee or a project, that is described in the database.

Attribute : An attribute represents some property of interest that further describes an entity, such as the employee's name or salary.

Relationship : A relationship among two or more entities represents an interaction among the entities; for example, a works-on relationship between an employee and a project.

Representational or implementation data models are the models used most frequently in traditional commercial DBMSs, and they include the widely-used relational data model, as well as the so-called legacy data models—the network and hierarchical models—that have been widely used in the past.

We can regard object data models as a new family of higher-level implementation data models that are closer to conceptual data models. Object data models are also frequently utilized as high-level conceptual models, particularly in the software engineering domain.

Physical data models describe how data is stored in the computer by representing information such as record formats, record orderings, and access paths. An access path is a structure that makes the search for particular database records efficient.

What is a database model ?

A database model shows the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. Individual database models are

designed based on the rules and concepts of whichever broader data model the designers adopt. Most data models can be represented by an accompanying database diagram.

## Types of Database Models :

There are many kinds of data models. Some of the most common ones include:

a) Hierarchical database model
b) Network model
c) Relational model
d) Object-oriented database model
e) Entity-relationship model

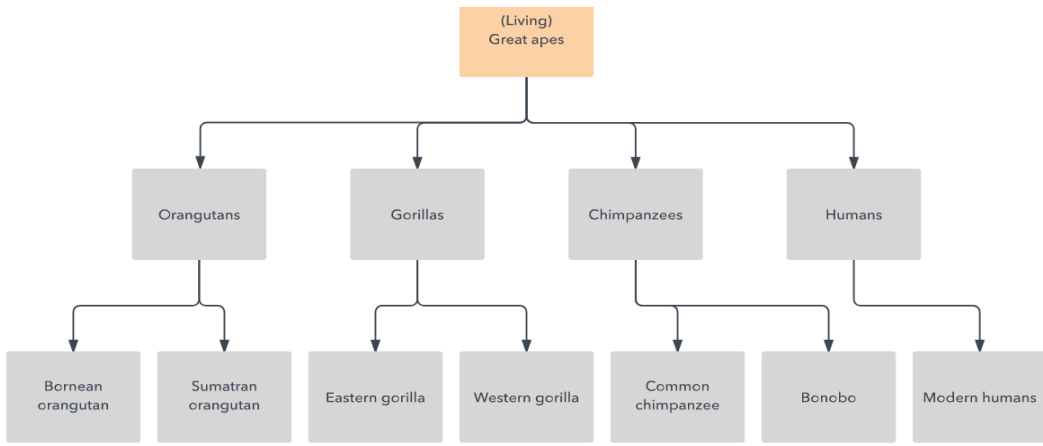
You may choose to describe a database with any one of these depending on several factors. The biggest factor is whether the database management system you are using supports a particular model. Most database management systems are built with a particular data model in mind and require their users to adopt that model, although some do support multiple models.

In addition, different models apply to different stages of the database design process. High-level conceptual data models are best for mapping out relationships between data in ways that people perceive that data. Record-based logical models, on the other hand, more closely reflect ways that the data is stored on the server.

Selecting a data model is also a matter of aligning your priorities for the database with the strengths of a particular model, whether those priorities include speed, cost reduction, usability, or something else.

## a) Hierarchical model

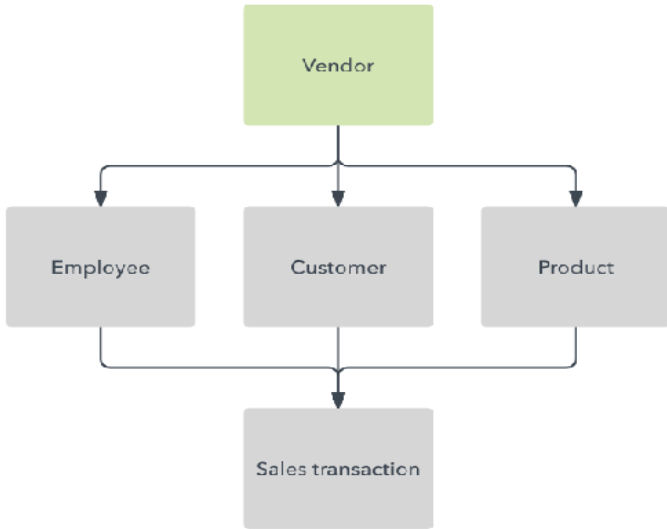
The hierarchical model organizes data into a tree-like structure, where each record has a single parent or root. Sibling records are sorted in a particular order. That order is used as the physical order for storing the database. This model is good for describing many real-world relationships.




This model was primarily used by IBM's Information Management Systems in the 60s and 70s, but they are rarely seen today due to certain operational inefficiencies.

## b) Network Model :

The network model builds on the hierarchical model by allowing many-to-many relationships between linked records, implying multiple parent records. Based on mathematical set theory, the model is constructed with sets of related records. Each set consists of one owner or parent record and one or more member or child records. A record can be a member or child in multiple sets, allowing this model to convey complex relationships.



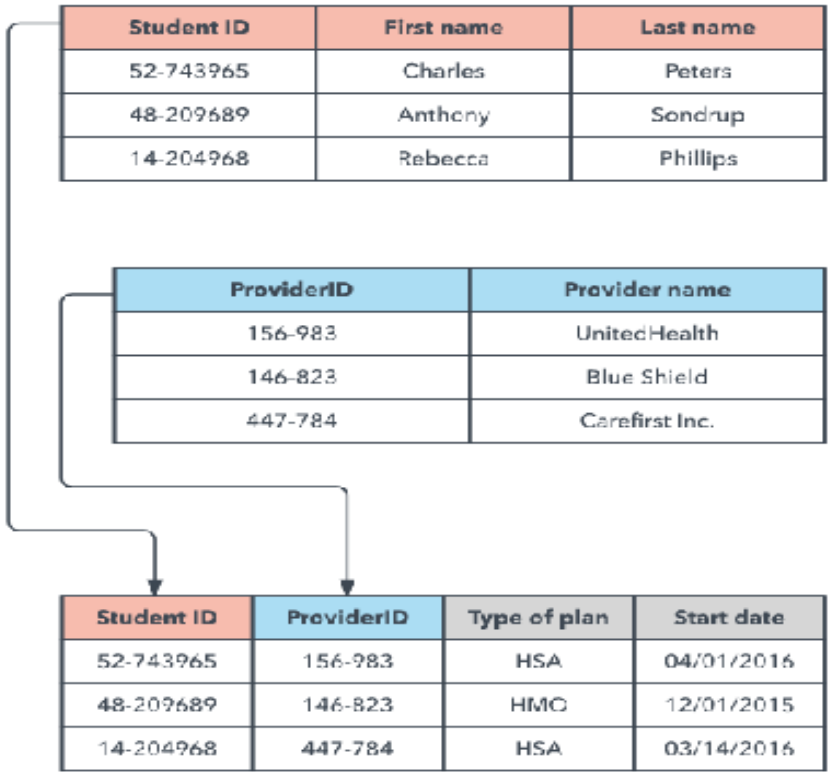

It was most popular in the 70s after it was formally defined by the Conference on Data Systems Languages (CODASYL).

## c) Relational Model

The most common model, the relational model sorts data into tables, also known as relations, each of which consists of columns and rows. Each column lists an attribute of the entity in question, such as price, zip code, or birth date. Together, the attributes in a relation are called a domain. A particular attribute or combination of attributes is chosen as a primary key that can be referred to in other tables, when it's called a foreign key.

Each row, also called a tuple, includes data about a specific instance of the entity in question, such as a particular employee.

The model also accounts for the types of relationships between those tables, including one-to-one, one-to-many, and many-to-many relationships. Here's an example:



| Student ID | First name | Last name |
|---|---|---|
| 52-743965 | Charles | Peters |
| 48-209689 | Anthony | Sondrup |
| 14-204968 | Rebecca | Phillips |

| ProviderID | Provider name |
|---|---|
| 156-983 | UnitedHealth |
| 146-823 | Blue Shield |
| 447-784 | Carefirst Inc. |

| Student ID | ProviderID | Type of plan | Start date |
|---|---|---|---|
| 52-743965 | 156-983 | HSA | 04/01/2016 |
| 48-209689 | 146-823 | HMO | 12/01/2015 |
| 14-204968 | 447-784 | HSA | 03/14/2016 |

Within the database, tables can be normalized, or brought to comply with normalization rules that make the database flexible, adaptable, and scalable. When normalized, each piece of data is atomic, or broken into the smallest useful pieces.

Relational databases are typically written in Structured Query Language (SQL). The model was introduced by E.F. Codd in 1970.

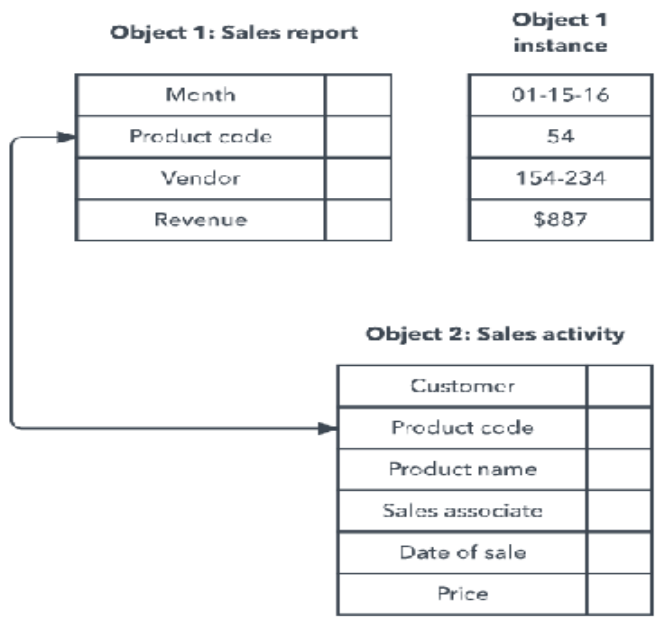
## a) Object-oriented database model

This model defines a database as a collection of objects, or reusable software elements, with associated features and methods. There are several kinds of object-oriented databases:

A multimedia database incorporates media, such as images, that could not be stored in a relational database.

A hypertext database allows any object to link to any other object. It's useful for organizing lots of disparate data, but it's not ideal for numerical analysis.

The object-oriented database model is the best known post-relational database model, since it incorporates tables, but isn't limited to tables. Such models are also known as hybrid database models.




## b) Entity Relationship Model :

This model captures the relationships between real-world entities much like the network model, but it isn't as directly tied to the physical structure of the database. Instead, it's often used for designing a database conceptually.

Here, the people, places, and things about which data points are stored are referred to as entities, each of which has certain attributes that together make up their domain. The cardinality, or relationships between entities, are mapped as well.

## 2.1.2 Schemas, Instances, and Database State

*Schema* : In any data model it is important to distinguish between the *description* of the database and the *database itself.* The description of a database is called the database schema, which is specified during database design and is not expected to change frequently.

Most data models have certain conventions for displaying the schemas as diagrams. A displayed schema is called a schema diagram. A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and some types of constraints. Other aspects are not specified in the schema diagram. Many types of constraints are not represented in schema diagrams.

*Instance* : The data in the database at a particular moment in time is called a database state or snapshot. It is also called the *current* set of occurrences or instances in the database. The actual data in a database may change quite frequently. In a given database state, each schema construct has its own *current set* of instances. Many database states can be constructed to correspond to a particular database schema. Every time we insert or delete a record, or change the value of a data item in a record, we change one state of the database into another state.

*Database State* : It is the current state of the database. The distinction between database schema and database state is very important. When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. We get the *initial state* of the database when the database is first populated or loaded with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a *current state.*

The DBMS is partly responsible for ensuring that *every* state of the database is a valid state—that is, a state that satisfies the structure and constraints specified in the schema. Hence, specifying a correct schema to the DBMS is extremely important, and the schema must be designed with the utmost care.

*Meta-data* : It is the description of the schema constructs and constraints. The DBMS stores schema constructs and its constraints. It is also called as the system catalog. The DBMS software can refer to the schema whenever it needs. The schema is sometimes called the intension, and a database state an extension of the schema.

*Schema Evolution :* The schema is not supposed to change frequently, it is not uncommon that changes need to be applied to the schema once in a while as the application requirements change. This is known as schema evolution. Most modern DBMSs include some operations for schema evolution that can be applied while the database is operational.

## 2.2 DBMS Architecture and Data Independence :

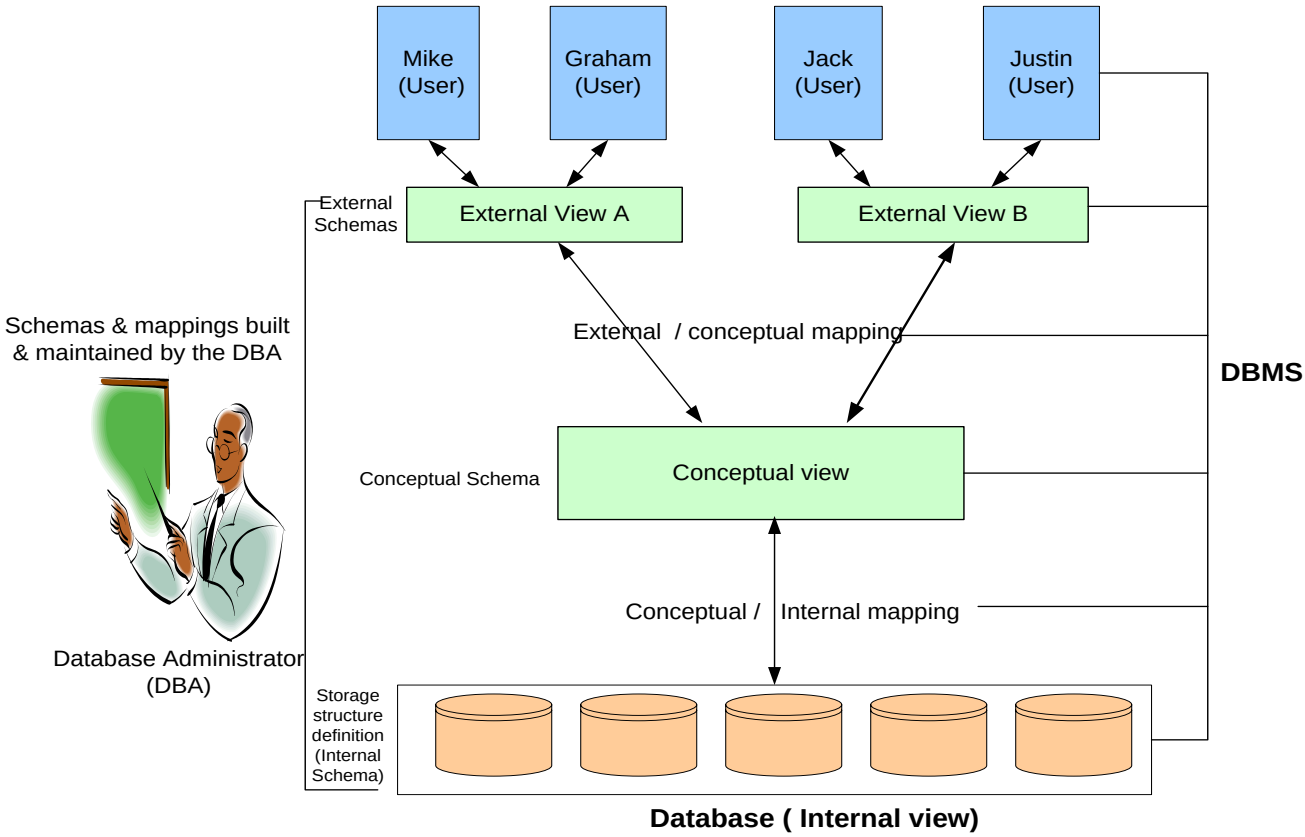
### *2.2.1 The Three-Schema Architecture*

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

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

The three-schema architecture is a convenient tool for the user to visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely, but support the three-schema architecture to some extent. Some DBMSs may include physical-level details in the conceptual schema. In most DBMSs that support user views, external schemas are specified in the

same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.




## In the above diagram,

- It shows the architecture of DBMS.
- Mapping is the process of transforming request response between various database levels of architecture.
- Mapping is not good for small database, because it takes more time.
- In External / Conceptual mapping, DBMS transforms a request on an external schema against the conceptual schema.
- In Conceptual / Internal mapping, it is necessary to transform the request from the conceptual to internal levels.

*1. Physical Level*

- Physical level describes the physical storage structure of data in database.
- It is also known as Internal Level.
- This level is very close to physical storage of data.

- At lowest level, it is stored in the form of bits with the physical addresses on the secondary storage device.
- At highest level, it can be viewed in the form of files.
- The internal schema defines the various stored data types. It uses a physical data model.

*2. Conceptual Level*

- Conceptual level describes the structure of the whole database for a group of users.
- It is also called as the data model.
- Conceptual schema is a representation of the entire content of the database.
- These schema contains all the information to build relevant external records.
- It hides the internal details of physical storage.

*3. External Level*

- External level is related to the data which is viewed by individual end users.
- This level includes a no. of user views or external schemas.
- This level is closest to the user.
- External view describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group.

## 2.2.2 Data Independence :

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

1. *Logical data independence :* It is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schemas or application programs.
2. *Physical data independence :* It is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

## 2.3 Database Languages and Interfaces

A DBMS supports a variety of users and must provide appropriate languages and interfaces for each category of users.

### 2.3.1 DBMS Languages :

DDL (Data Definition Language): used (by the DBA and/or database designers) to specify the conceptual schema. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

In DBMSs where a clear separation is maintained between the conceptual and internal levels. The mappings between the two schemas may be specified in either one of these languages.

SDL (Storage Definition Language): used for specifying the internal schema.

VDL (View Definition Language): used for specifying the external schemas (i.e., user views).

DML (Data Manipulation Language): used for performing operations such as retrieval, insertion, deletion and modification of data of the database.

SQL : In current DBMSs, the preceding types of languages are usually *not considered distinct languages;* rather, a comprehensive integrated language is used that includes constructs for conceptual schema definition, view definition, and data manipulation. Storage definition is typically kept separate, since it is used for defining physical storage structures to fine-tune the performance of the database system, and it is usually utilized by the DBA staff. A typical example of a comprehensive database language is the SQL relational database language, which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification and schema evolution. The SDL was a component in earlier versions of SQL but has been removed from the language to keep it at the conceptual and external levels only.

### 2.3.2 DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following.

*Menu-Based Interfaces for Browsing*

These interfaces present the user with lists of options, called **menus,** that lead the user through the formulation of a request. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step by step by picking options from a menu that is displayed by the system. Pull-down menus are becoming a very popular technique in window-based user interfaces. They are often used in **browsing interfaces,** which allow a user to look through the contents of a database in an exploratory and unstructured manner.

*Forms-Based Interfaces*

A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions. Many DBMSs have forms specification languages, special languages that help programmers specify such forms. Some systems have utilities that define a form by letting the end user interactively construct a sample form on the screen.

*Graphical User Interfaces*

A graphical interface (GUI) typically displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to pick certain parts of the displayed schema diagram.

*Natural Language Interfaces*

These interfaces accept requests written in English or some other language and attempt to "understand" them. A natural language interface usually has its own "schema," which is similar to the database conceptual schema. The natural language interface refers to the words in its schema, as well as to a set of standard words, to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.

*Interfaces for Parametric Users*

Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. Systems analysts and programmers design and implement a special interface for a known class of naive users. Usually, a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request. For example, function keys in a terminal can be programmed to initiate the various commands. This allows the parametric user to proceed with a minimal number of keystrokes.

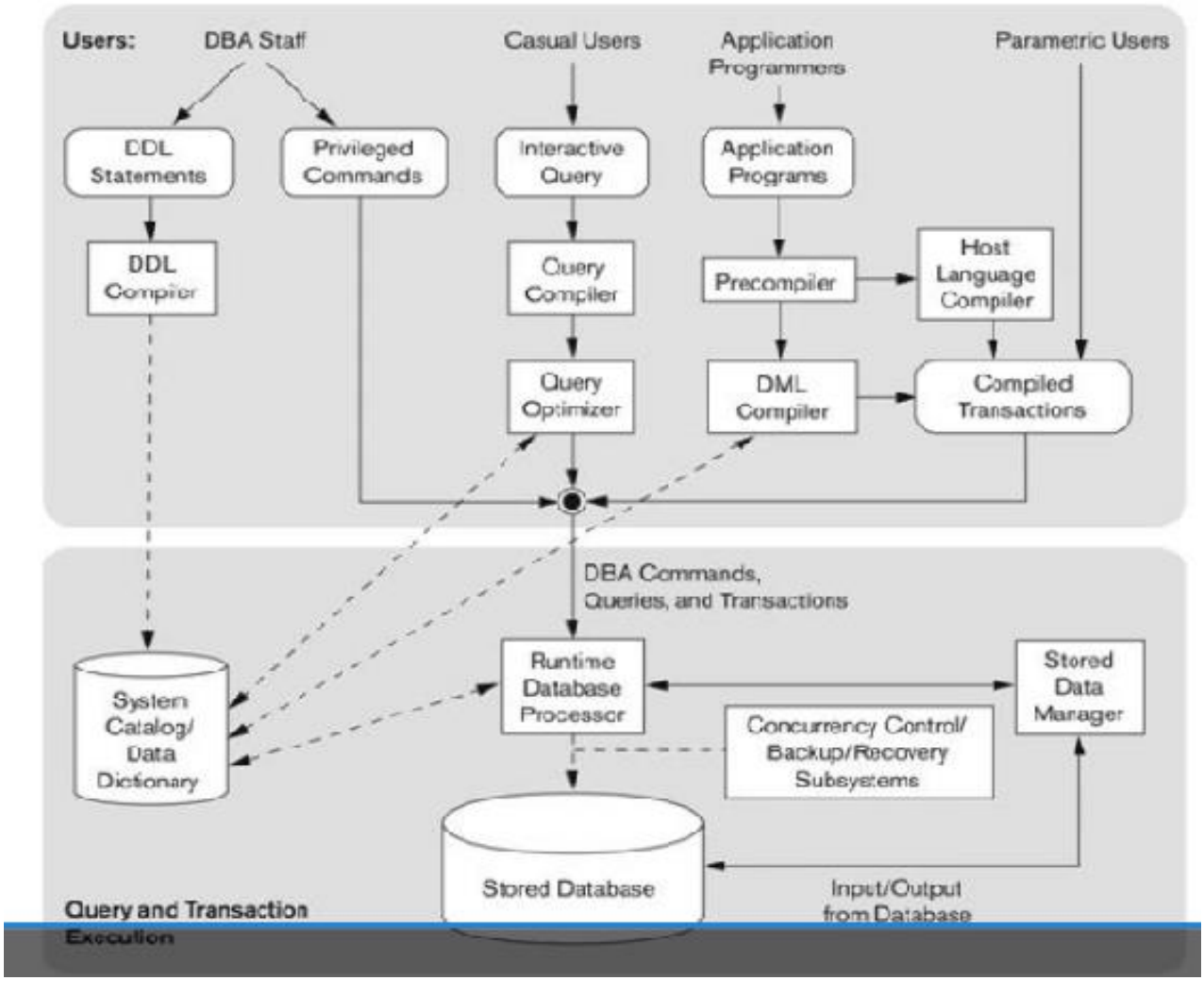
*Interfaces for the DBA*

Most database systems contain privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

## 2.4 The Database System Environment

A DBMS is a complex software system. Here, it has been discussed the types of software components that constitute a DBMS and the types of computer system software with which the DBMS interacts. Below Figure  illustrates, the simplified form along with the typical DBMS components. The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the operating system (OS), which schedules disk input/output. A higher-level stored data manager module of the DBMS controls access to DBMS information that is stored

on disk, whether it is part of the database or the catalog. The dotted lines and circles marked in below Figure illustrate accesses that are under the control of this stored data manager. The stored data manager may use basic OS services for carrying out low-level data transfer between the disk and computer main storage, but it controls other aspects of data transfer, such as handling buffers in main memory. Once the data is in main memory buffers, it can be processed by other DBMS modules, as well as by application programs.




The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names of files, data items, storage details of each file, mapping information among schemas, and constraints, in addition to many other types of information that are needed by the DBMS modules. DBMS software modules then look up the catalog information as needed.

The run-time database processor handles database accesses at run time; it receives retrieval or update operations and carries them out on the database. Access to disk goes through the stored data manager. The query compiler handles high-level queries that are entered interactively. It parses, analyzes, and compiles or interprets a query by creating database access code, and then generates calls to the run-time processor for executing the code.

The pre-compiler extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor.

The DBMS interacts with the operating system when disk accesses—to the database or to the catalog—are needed. If the computer system is shared by many users, the OS will schedule DBMS disk access requests and DBMS processing along with other processes. The DBMS also interfaces with compilers for general-purpose host programming languages. User-friendly interfaces to the DBMS can be provided to help any of the user types shown in above Figure to specify their requests.

## 2.4.2 Database System Utilities

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

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

2. *Backup:* A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape. The backup copy can be used to restore the database in case of catastrophic failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex but it saves space.

3. *File reorganization:* This utility can be used to reorganize a database file into a different file organization to improve performance.

4. *Performance monitoring:* Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files to improve performance.

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

### 2.4.3 Tools, Application Environments, and Communications Facilities

Other tools are often available to database designers, users, and DBAs. CASE tools are used in the design phase of database systems. Another tool that can be quite useful in large organizations is an expanded data dictionary (or data repository) system. In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information. Such a system is also called an information repository. This information can be accessed *directly* by users or the DBA when needed. A data dictionary utility is similar to the DBMS catalog, but it includes a wider variety of information and is accessed mainly by users rather than by the DBMS software.

Application development environments, such as the PowerBuilder system, are becoming quite popular. These systems provide an environment for developing database applications and include facilities that help in many facets of database systems, including database design, GUI development, querying and updating, and application program development.

The DBMS also needs to interface with communications software, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or their local personal computers. These are connected to the database site through data communications hardware such as phone lines, long-haul networks, local-area networks, or satellite communication devices. Many commercial database systems have communication packages that work with the DBMS. The integrated DBMS and data communications system is called a DB/DC system. In addition, some distributed DBMSs are physically distributed over multiple machines. In this case, communications networks are needed to connect the machines. These are often local area networks (LANs) but they can also be other types of networks.

## 2.5 Architectures of DBMS

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

### a) 1-tier architecture:

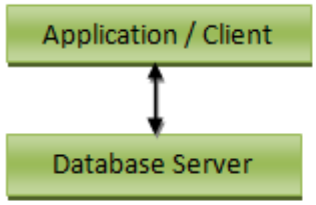
Here, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Both client programs and server programs runs on the same machine. i.e. only one computer can be used as both client machine and server machine. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

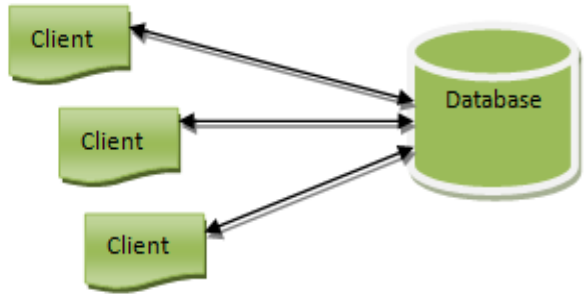
### b) 2-tier architecture :

Here, there will be two computers in which one machine runs client application and another machine runs server part of the application. In 2-tier architecture, application program directly interacts with the database. There will not be any user interface or the user involved with database interaction. Imagine a front end application of School, where we need

to display the reports of all the students who are opted for different subjects. In this case, the application will directly interact with the database and retrieve all required data. Here no inputs from the user are required. This involves 2-tier architecture of the database.

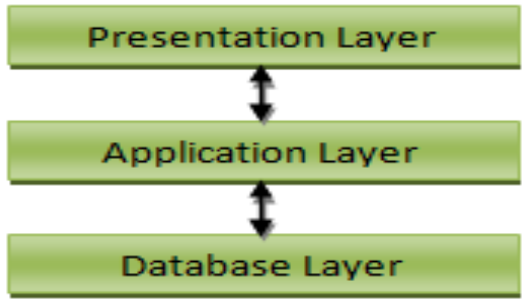
Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.




c) 3-tier architecture :

3-tier architecture is the most widely used database architecture. It can be viewed as below.




- *Presentation layer / User layer* is the layer where user uses the database. He does not have any knowledge about underlying database. He simply interacts with the database as though he has all data in front of him. You can imagine this layer as a registration form where you will be inputting your details. Did you ever guessed, after pressing 'submit' button where the data goes? No right? You just know that your details are saved. This is the presentation layer where all the details from the user are taken, sent to the next layer for processing.

- *Application layer* is the underlying program which is responsible for saving the details that you have entered, and retrieving your details to show up in the page. This layer has all the business logics like validation, calculations and manipulations of data, and then sends the requests to database to get the actual data. If this layer sees that the request is invalid, it sends back the message to presentation layer. It will not hit the database layer at all.

- *Data layer or Database layer* is the layer where actual database resides. In this layer, all the tables, their mappings and the actual data present. When you save you details from the front end, it will be inserted into the respective tables in the database layer, by using the programs in the application layer. When you want to view your details in the web

browser, a request is sent to database layer by application layer. The database layer fires queries and gets the data. These data are then transferred to the browser (presentation layer) by the programs in the application layer.

### d) N-tier / Multi-tier architecture :

Multitier architecture (often referred to as *n*-tier architecture) or multilayered architecture is a client–server architecture in which presentation, application processing, and data management functions are physically separated. *N*-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. It will distribute the work load to several machines so that processing time will be reduced.

## 2.6 Classification of Database Management Systems :

Several criteria are normally used to classify DBMSs.

*i) Based on the data model :*

The two types of data models used in many current commercial DBMSs are the relational data model and the object data model. Many legacy applications still run on database systems based on the hierarchical and network data models.

The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs that are being called object-relational DBMSs. We can hence categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.

*ii) Based on number of users :*

Single-user systems support only one user at a time and are mostly used with personal computers. Multiuser systems, which include the majority of DBMSs, support multiple users concurrently.

*iii) Based on the number of sites over which database is distributed :*

A DBMS is centralized if the data is stored at a single computer site.

A centralized DBMS can support multiple users, but the DBMS and the database themselves reside totally at a single computer site.

A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network.

Homogeneous DDBMSs use the same DBMS software at multiple sites. A recent trend is to develop software to access several autonomous preexisting databases stored under heterogeneous DBMSs. This leads to a federated DBMS (or multidatabase system), where the

participating DBMSs are loosely coupled and have a degree of local autonomy. Many DDBMSs use a client-server architecture.

*iv) Based on the cost of database :*

A fourth criterion is the **cost** of the DBMS. The majority of DBMS packages cost between $10,000 and $100,000. Single-user low-end systems that work with microcomputers cost between $100 and $3000. At the other end, a few elaborate packages cost more than $100,000.

*v) Based on the type of access path options for storing files :*

One well-known family of DBMSs is based on inverted file structures. Finally, a DBMS can be general-purpose or special-purpose. When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems developed in the past are special-purpose DBMSs. These fall into the category of on-line transaction processing (OLTP) systems, which must support a large number of concurrent transactions without imposing excessive delays.

## Review Questions :

a) Define the following terms: *data model, database schema, database state, internal schema, conceptual schema, external schema, data independence, DDL, DML, SDL, VDL, query language, host language, data sublanguage, database utility, catalog, client-server architecture*

b) Discuss the main categories of data models.

c) What is the difference between a database schema and a database state?

d) Describe the three-schema architecture. Why do we need mappings between schema levels? How do different schema definition languages support this architecture?

e) What is the difference between logical data independence and physical data independence?

f) What is the difference between procedural and nonprocedural DMLs?

g) Discuss the different types of user-friendly interfaces and the types of users who typically use each.

h) With what other computer system software does a DBMS interact?

i) Discuss some types of database utilities and tools and their functions.

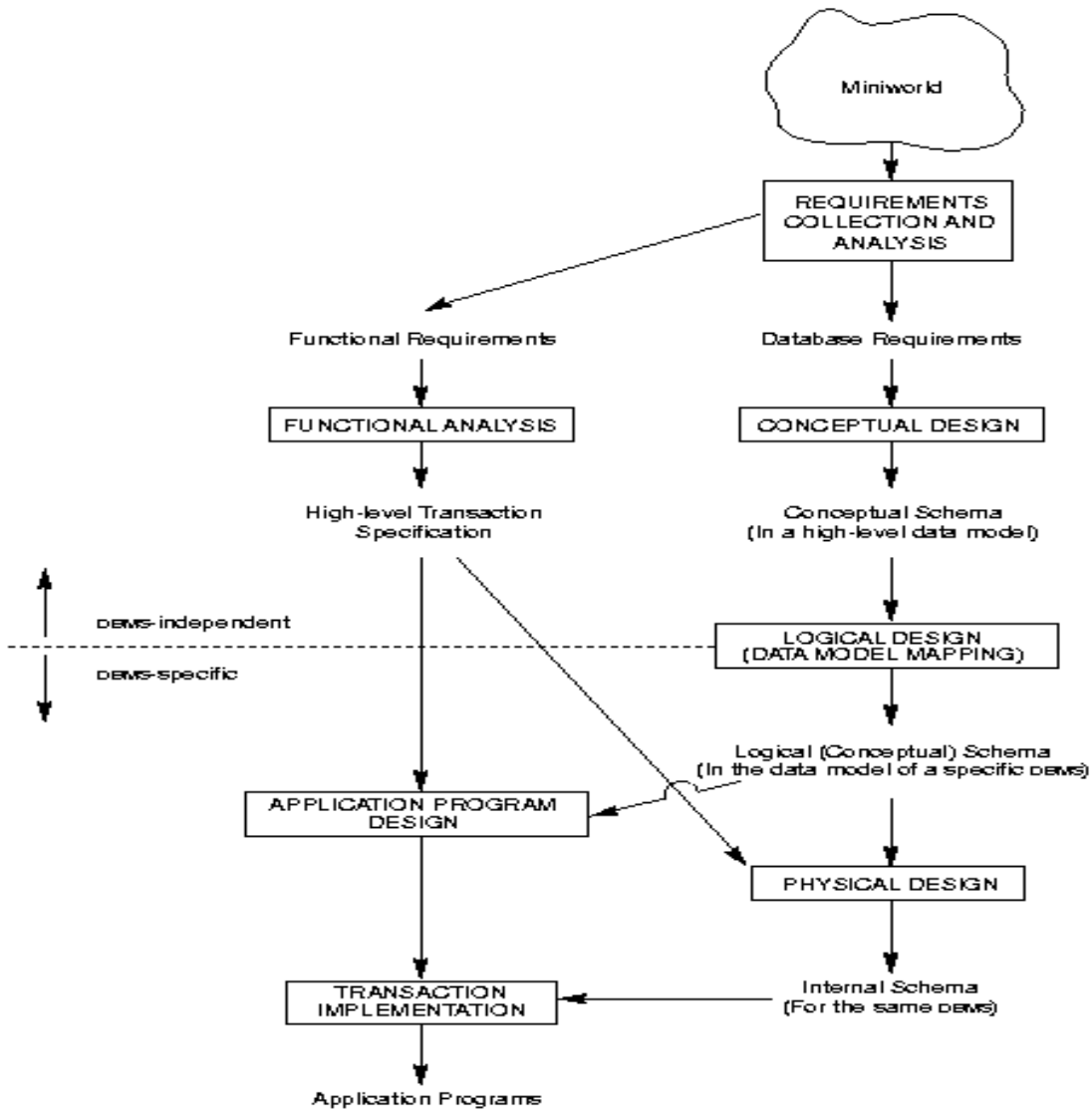
j) How do you classify the DBMSs ?

# Chapter-3. Data Modeling Using the Entity – Relationship (ER) Model.

The term database application refers to a particular database and the associated programs that implement the database queries and updates. Conceptual modeling is a very important phase in designing a successful database application. Entity-Relationship (ER) model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of the database applications and many database design tools are having its concepts.

Using High-Level Conceptual Data Models for database design : The following figure shows a simplified description of the database design process.

**Figure 3.1**



The first step shown is requirements collection and analysis. During this step, the database designers will interview prospective database users to understand and document their data requirements. These requirements should be specified in as detailed and complete form as possible. Simultaneously, while specifying data requirements, it is useful to specify the known functional requirements of the application. These consists of the user defined operations that includes retrievals and updates.

Once all the requirements have been collected and analyzed, the next step is to create a conceptual schema for the database, using a high-level conceptual data model. This step is called conceptual design. It is a concise description of data requirements of the users and includes detailed description of the entity types, relationships, and constraints. These are expressed using the concepts provided by the high level data model. This model is also considered as a reference to ensure that all users data requirements are met and that the requirements do not conflict. This approach enables database designers to concentrate on specifying properties of the data without being concerned with storage details.

The next step is the actual implementation of the database, using a commercial DBMS. Most of the current commercial DBMSs use an implementation data model such as relational or object-relational database model. This step is called logical design or data model mapping.

The last step is the physical design, in which the internal storage structures, indexes, access paths, and file organizations for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high level transaction specifications.

## Entity types, Entity sets, Attributes and Keys :

ER-Model : The ER model is a high-level conceptual data model. This was introduced by Peter Chen in 1976, and is now the most widely used conceptual data model. Much work has been done on the ER model, and various extensions and enhancements have been proposed. ER model describes data as entities, relationships and attributes.

Entity : It is a thing or real time object in the world with an independent existence. An entity may be an object with a physical existence or it may be an object with a conceptual existence. Each entity has attributes ( properties ). Entity is represented by a rectangle


Example : An employee is the entity and the attributes are empployeeID, Employeename, DOB, Address, designation, salary, ….etc

Entity Set : Group of similar entities are called as an entity set. i.e. entity with same attributes.

Example : Group of employees, group of companies….etc

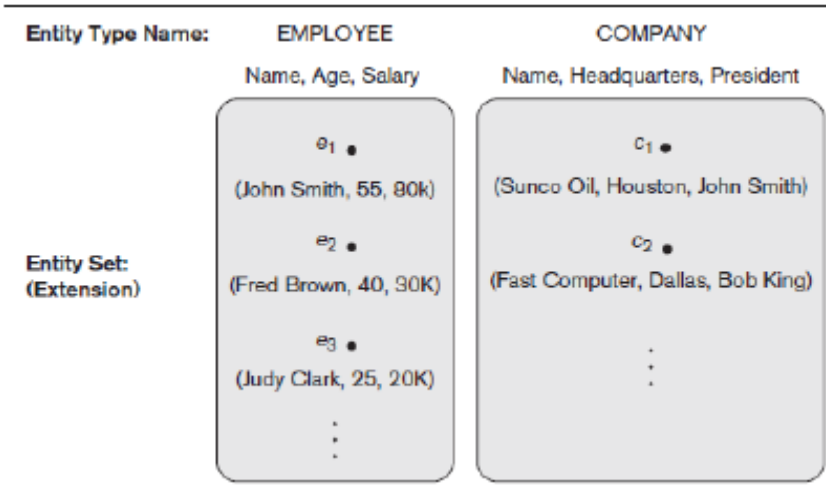
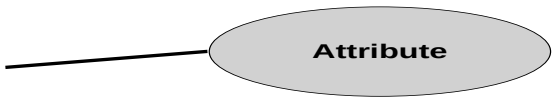
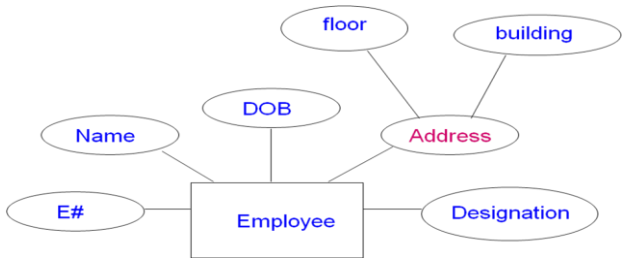



**Figure 7.6**
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

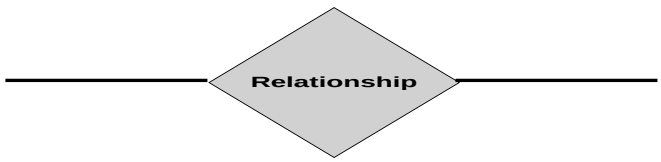
Attributes : It is a property of an entity. It is represented by oval shape with a line.




Example : Attributes of an entity employee are empployeeID, Employeename, DOB, Address, designation, salary, ….etc

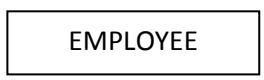



Relationship : It is the type of association between two entities, more entities or with in the entity itself. It is represented by rhombus with associated lines.

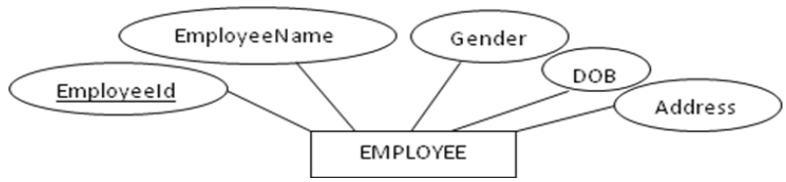



Types of Entities : There are two types of entities. i.e. Strong entity and Weak entity. This depends on the type of the key attributes involved in the entities.

Strong Entity : Strong entities are the ones which are having their own key attributes. By default all the entities are strong. Only at the special occasions, some entities will become weak. Pictorially it is represented by rectangle.

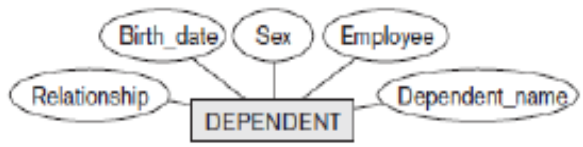


Example : Employee is a strong entity. Because, Employee is a independent entity, and every employee is identified by a key attribute EmployeeId which does not depends on any other entity.




Weak Entity : A weak entity does not having its own key attribute. Even if it is having a key attribute, it depends on the key attribute of other entity. It does not having its individual identity. It is always identified under the shadows of other entity. It is represented by double lined rectangle.

Example : Dependents of an employee.




In an organization, any dependent of an employee is always identified by the help of EmployeeId, because, without that nobody knows that he belongs to which employee.

Types of attributes : There are several types of attributes that depends upon the type of the values and number of values they hold. They are

i) Simple or atomic attribute
ii) Key attribute
iii) Composite attribute
iv) Multivalued attribute
v) Derived attribute

i) Simple/atomic attribute : This attribute is having a single value. i.e. its value cannot be divisible further. It is represented by a oval shape with a line. By default every attribute is of simple type.


ii) Key attribute : This attribute is having a key value. i.e. by using its value we can able to identify the values of remaining attributes of a given entity. It is always acts as primary identifier. It is represented by a oval shape with an underline.

Example : EmployeeId attribute in an Employee entity is a key attribute.


iii) Composite attribute : This attribute is having few more sub attributes in the form of extended components. i.e. Its value can be extended further in the form of separate components. It is represented by a oval shape with an extended components

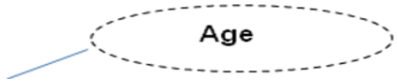
Example : EmployeeName attribute in an Employee entity can be extended as firstname, middlename, and lastname.


iv) Multivalued attribute : This attribute is having more than one value. i.e. Its attribute is only one, buts its values are more than one. It is represented by a double lined oval shape.

Example : Skills attribute in an employee entity is having more than one value some times. Because an employee is having expertise in various skill sets.


v) Derived attribute : This attribute is having a value which is derived from the value of other attribute. It is represented by a dotted lined oval shape.

Example : Age attribute in an employee entity is deriving its value from the dateofbirth attribute's value.

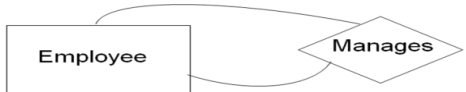



Types of Relationships : There are three relationships. This depends upon the number of entities involved in the relationship. It is also called as degree of relationship.

i) Unary relationship
ii) Binary relationship
iii) Ternary relationship

i) Unary relationship : In this relationship, the number of entities involved is only one. The association exists with in the entity itself. It is also called as recursive relationship.

Example : An employee (supervisor) is managing other employees. Here, basically supervisor is an employee. He is managing other employees. Manage is the relationship.

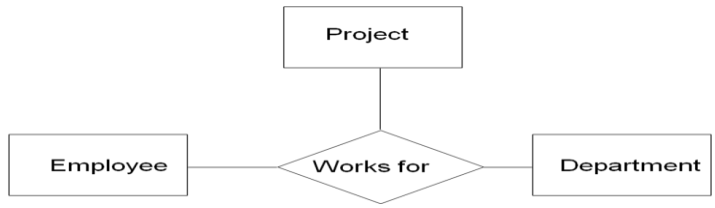


ii) Binary relationship : In this relationship, the number of entities involved is two. The association exists with two entities.

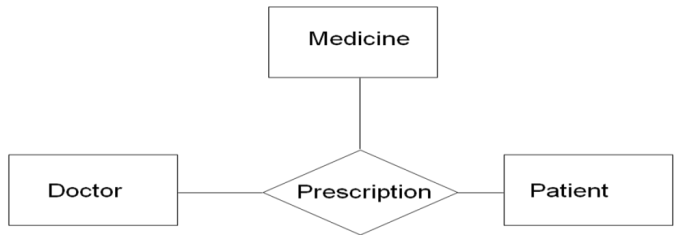
Example : An employee works for the department. Here employee and department are the two entities. Works is the relationship exists between two entities.


iii) Ternary relationship : In this relationship, the number of entities involved is more than two. The association exists with more than two entities.

Example 1 : An employee works on a particular project in a particular department. Here employee, project and department are the three entities. Works is the relationship exists between three entities.




Example 2 : A doctor is giving prescription to patients on medicines. Here, doctor, patient and medicine are the three entities. Prescribe is the relationship exists between three entities.




Cardinality ratio : This ratio will show the association among members of the involving entities in a relationship. It also shows number of instances that a relationship exists among the members of the entities. There are three types.

i) One to one (1:1)
ii) One to many or Many to one (1:M / M:1)
iii) Many to many (M:N)

c) Cardinality ratio in unary relationship : In unary relationship, the number of participating entities are only one. Here, we need to verify whether this is possible to apply it or not.

For example an employee is managing other employees.

i) One to one : According to this ratio, every employee is managed by another employee. Rarely, we can see this situation in real time environment.

ii) One to many : According to this ratio, a group of employees are managed by one employee (supervisor). Commonly, we can see this situation in real time environment. i.e. one manager with many employees.

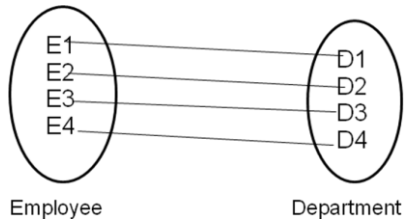
Many to one : According to this ratio, many managers are managing a single employee. Rarely, we can see this situation in real time environment. i.e. Many managers with one employee.

iii) Many to Many : According to this ratio, a manager is managing group of employees and an employee is managed by many managers. Rarely, we can see this situation in real time environment. i.e. one manager with many employees and an employee with many managers.

d) Cardinality ratio in Binary relationship : In binary relationship, the number of participating entities are two. Here, we need to verify whether these ratios can be apply it or not.

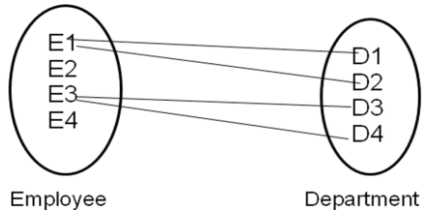
For example an employee is working in a department


i) One to one : According to this ratio, every employee is working for a separate department. i.e. for each of the employee, there should be a individual department. Rarely, we can see this situation in real time environment.



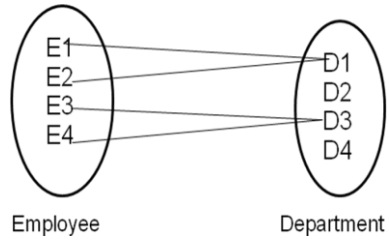

From the above diagram, every employee from the set employee is working individually for a separate department in the set department.

ii) One to many : According to this ratio, an employee is working for more than one department. It is shown in the following diagram. Here, the employee E1 is working for the departments D1 and D2. Similarly, employee E3 is working for the departments D3 and D4. Rarely, we can see this situation in real time environment. i.e. an employee is working for many departments.

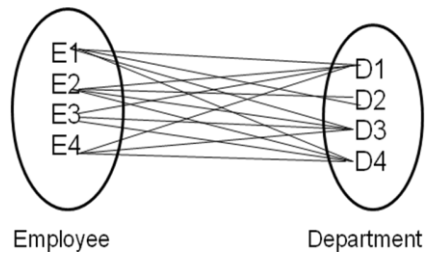


Many to one : According to this ratio, many employees are working for a department. It is shown in the following diagram. Here, the employee E1 and E2 are working for the departments D1. Similarly, employee E3 and E4 are working for the department D3. Commonly, we can see this situation in real time environment. i.e. many employees are working for single department.

Rarely, we can see this situation in real time environment. i.e. Many managers with one employee.

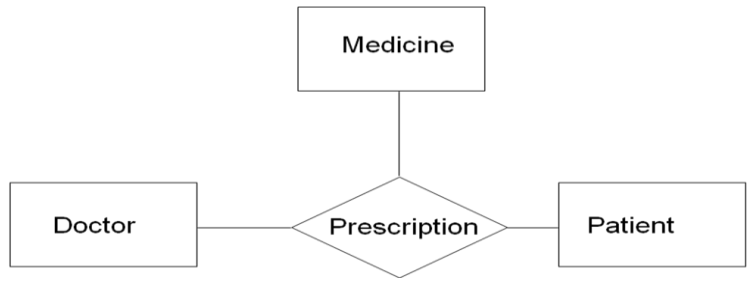



iii) Many to Many : According to this ratio, an employee is working for different departments and every department is having many employees. Rarely, we can see this situation in real time environment. i.e. one employee works for many departments and every department works with many employees.

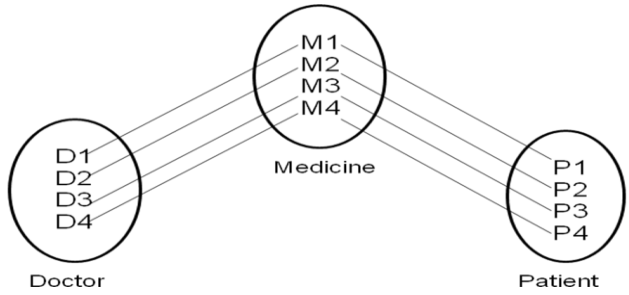



e) Cardinality ratio in Ternary relationship : In ternary relationship, the number of participating entities are more than two. Here, we need to verify whether these ratios can be applicable or not.

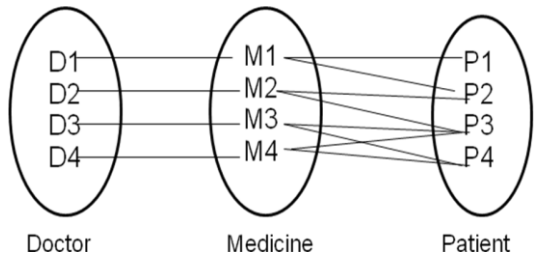
Example : A doctor is giving prescription to patients on medicines. Here, doctor, patient and medicine are the three entities. Prescribe is the relationship exists between three entities.



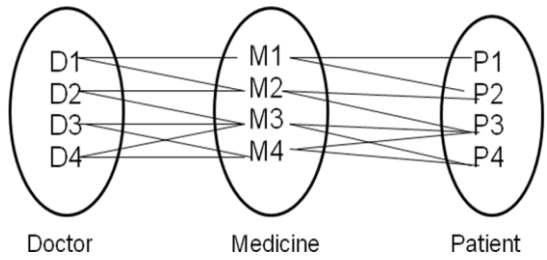

i) One to one : According to this ratio, every doctor is giving prescription on single medicine to a single patient. i.e. for each of the doctor, there will be a single patient and prescription will be given on only one medicine. Rarely, we can see this situation in real time environment.



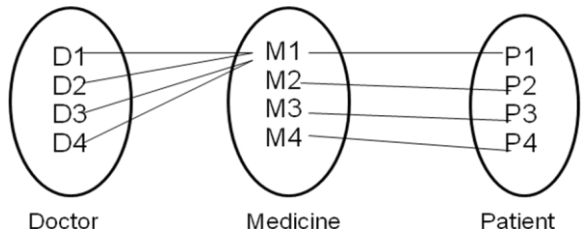

ii) One to many : According to this ratio, every doctor is giving prescription on single medicine to many patients. i.e. for each of the doctor, there will be a single medicine and prescription will be given to many patients. Rarely, we can see this situation in real time environment. This is show in the following diagram.



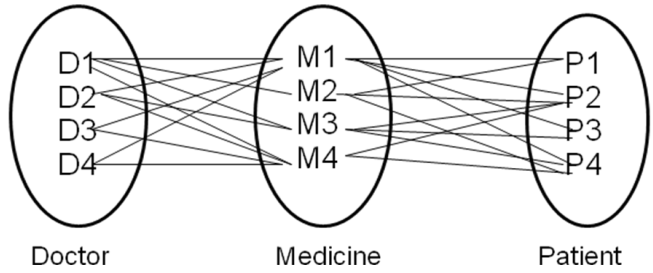

In another case, every doctor is giving prescription on many medicine to many patients. i.e. for each of the doctor, there will be many medicines and prescription will be given to many patients. Commonly, we can see this situation in real time environment. This is show in the following diagram.




Many to one : In this case, many doctors are prescribing a single medicine to a single patient. It is shown in the following diagram. Here, the doctors D1, D2, D3 and D4 are prescribing a medicine M1 to a single patient P1. Rarely, we can see this situation in real time environment.



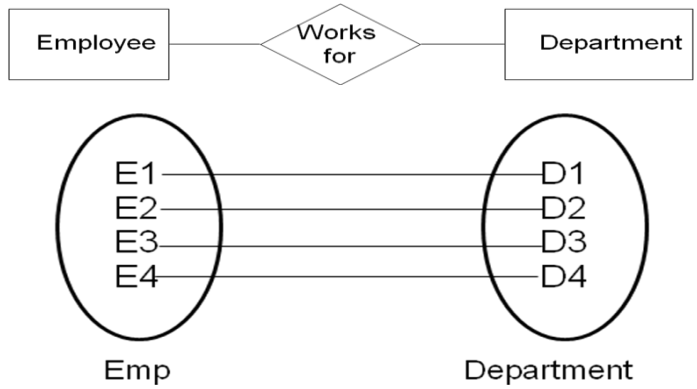

iii) Many to Many : In this case, many doctors are prescribing many medicines to many patients. It is shown in the following diagram. Here, the doctors D1 is prescribing many medicines M1, M2, M3.. to many patients P1, P2 and P3. Similar is the case for other doctors, medicines and patients. Commonly, we can see this situation in real time environment.




Participation Constraints and Existence Dependencies : The participation constraint is the degree of involvement of the relationship among the members of one entity with members of another participating entity. It specifies whether the existence of an entity depends on its being related o another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called as the minimum cardinality constraint.

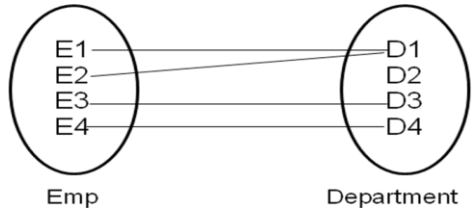
There are two types of participation constraints. i) Total participation and ii) Partial participation.

i) Total participation / Existence dependency : To explain this concept, consider the following two sets (entities), Employee and Department and relationships exists among these two is works for.




In the above diagram, every employee in the set Emp is working for department. The relationship "works for" is true for all the members of both the entities. i.e. every employee is working in a department and in each of the department, there is an employee is working. This is called total or full participation. It is also called as existence dependency. It is represented by thick line or double lines which as shown below.

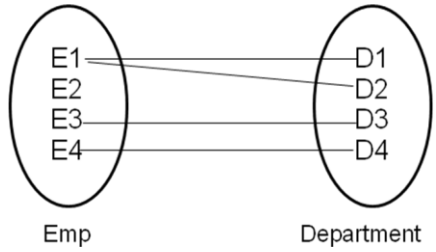

ii) Partial participation : Consider the following diagram with two entities Emp and Department.



Here, the relationship “works for” is not true with every members of the department entity and it is true for all the members of emp entity. This is called as partial participation. It is represented by a thin line or single line which is shown in the following figure.

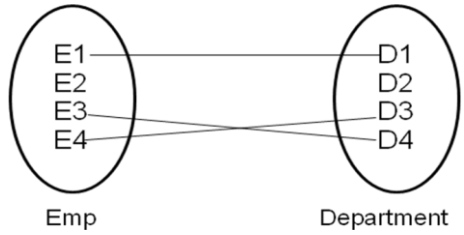

Here, the entity Emp is having total participation with relationship “works for”, since all its members are working in different departments. But, the department entity is partial, because, in one of its departments ‘D2’, no employee is working which results in false with relationship “works for”. Therefore, emp is total and department is partial.

Consider the following case:



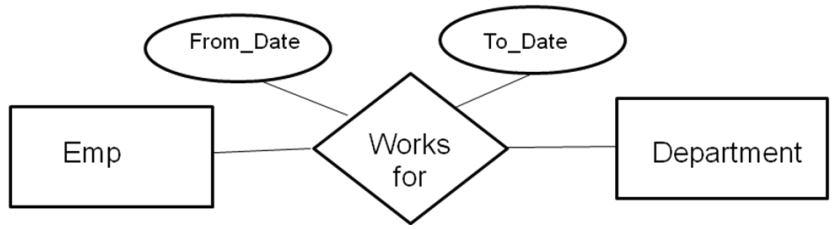

Here, the entity department is having total participation with relationship “works for”, since all its departments are working by some employees. But, the emp entity is partial, because, one of its employees ‘E2’ is not working for any department which results in false with relationship “works for”. Therefore, department is total and emp is partial. Pictorially it is represented as follows.


Consider the following case:

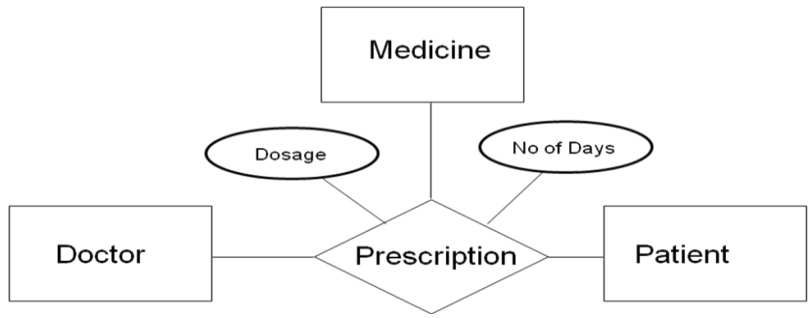



Here, the entity emp is having partial participation with relationship “works for”, since one of its employees ‘E2’ is not working for any of the departments. Similarly, the department entity is also partial, because, in one of its departments ‘D2’, no employee is working which results in false with relationship “works for”. Therefore, emp is partial and department is also partial. Pictorially it is represented as follows.

Relationship attributes : These are the attributes defined on the relationship. Consider the following ER diagram.




Here, the attributes “From_date” and “To_date” are of relationship type attributes, because they are defined on the relationship “works for”. These attributes best defines the relationship rather than any individual entity. i.e. an employee is working from so and so starting date to so and so ending date for a particular department. Following is the one more example.

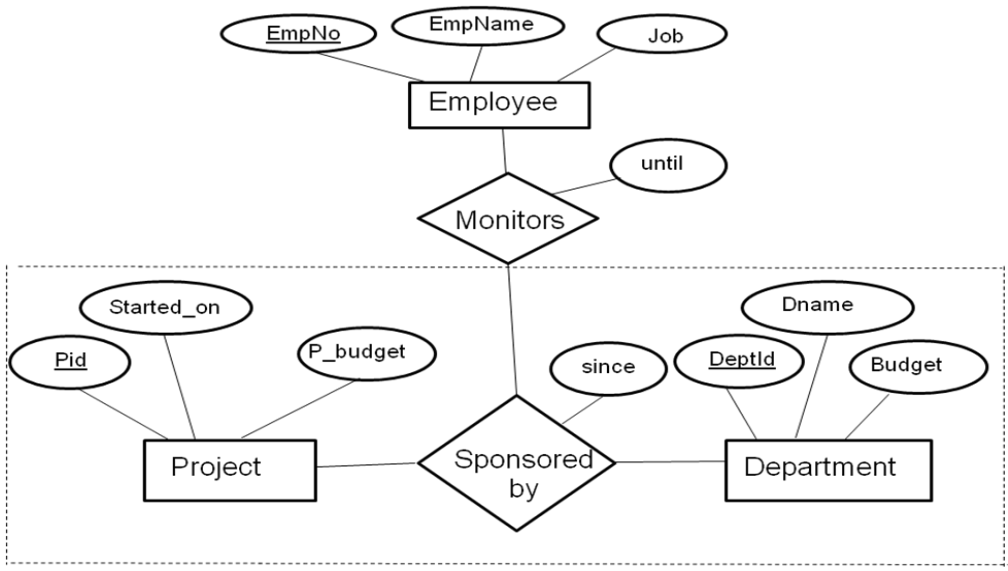



Here, the attributes “Dosage” and “No of Days” are of relationship type attributes, because they are defined on the relationship “Prescription”. i.e. a doctor is prescribing some dosage of medicine to patient for some number of days.

Aggregation : It is the relationship exists with the another relationship. A relationship is an association between entity sets. Sometimes, we have to model a relationship between collection of entities and relationships. Aggregation is meant to represent a relationship between a whole object and its component parts. It is used when we have to model a relationship involving entity sets and relationship sets. It allows us to treat a relationship set as an entity set for the purpose of participation in other relationships.

Example: A project is sponsored by a department. Here, project is an entity set and each project is sponsored by one or more departments. Aggregation allows relationship set participates in another relationship set. A department that sponsors project might assign employees to monitor the relationship.

Intuitively, monitors should be a relationship set that associates a sponsored by relationship with an employee entity rather than project. Here, we have defined relationships to associate two or more entities.




When we use aggregation ?

We use it, when we need to express a relationship among relationships.

Extended Entity Relationship features : These are the new features added to the previous ER model to accommodate the present trend requirements of industries such as manufacturing, telecom and geographic information systems. These types of databases have more complex requirements than the more traditional applications. This led to the development of additional semantic data modeling concepts that were incorporated into conceptual data models. Various semantic data models have been proposed in the literature. Many of these concepts were also developed independently in related areas of computer science such as knowledge representation area of artificial intelligence and the object modeling areas in software engineering.

Superclasses, Subclasses and Inheritance : These are the concepts of extended ER model's specialization and generalization.

Superclass: It is the entity type with common features which can be extended as sub classes.

Subclass: It is the subgroup in a entity type with some specialized features along with common features of its superclass. In many cases an entity type has numerous sub groupings of its entities that are meaningful and need to be represented explicitly because of their significance to the database application.

Example : In an entity type of EMPLOYEE, we have many group of employees in the form of their designations such as secretary, engineer, manager, technician, operator, accountant and so on. The set of entities in each of the latter groupings is a subset of the entities that belong to the EMPLOYEE entity set. We call each of these subgroupings as subclass of the EMPLOYEE entity type and the EMPLOYEE entity type is called the superclass for each of these subclasses.

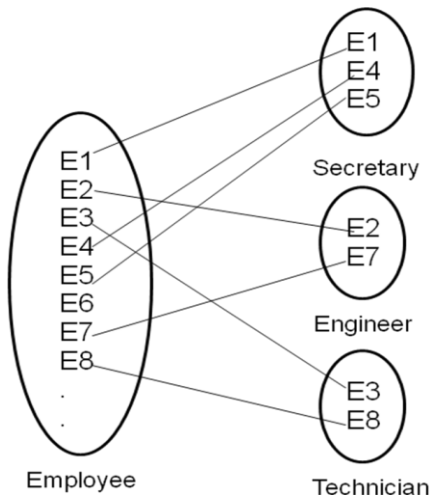
We call the relationship between a superclass and any one of its subclasses a superclass/subclass or simply class/subclass relationship.

Inheritance : The entity in the subclass represents the same features from the superclass, as it possess values for its specific attributes as well as values of its attributes as a member of the superclass. We say that an entity that is a member of a subclass inherits all the attributes of the entity as a member of the superclass. The entity also inherits all the relationships in which the superclass participates.

## Specialization and Generalization :

Specialization : It is the process of defining a set of subclasses of an entity type. The set of subclasses forms a specialization and it is defined on the basis of some distinguishing characteristics of the entities in the superclass.

Example-1 : The set of subclasses { secretary, engineer, technician} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the job type of each employee entity. Following diagram shows specialization among employee entity.




Here, a few entity instances that belong to subclasses of the { secretary, engineer, technician} specialization. Again notice that an entity that belongs to a subclass represents the same real world entity connected to it in the EMPLOYEE superclass.

The specialization allows us to the do the following :

i) Define a set of subclasses of an entity type.

ii)Establish additional specific attributes with each subclass.

iii) Establish additional specific relationship types between each subclass and other entity types or other subclasses.

The set of subclasses { secretary, engineer, technician} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the job type of each employee entity. Following diagram shows specialization among employee entity.

Generalization : It is the reverse process of specialization in which we suppress the differences among several entity types, identify their common features and generalize them into a single superclass of which the original entity types are special subclasses. It can be viewed functionally the inverse of the specialization process.

Example : Consider the entity types CAR and TRUCK which are shown in the following diagram-A. They have several common attributes, they can be generalized into the entity type VEHICLE. Both CAR and TRUCK are now subclasses of the generalized superclass VEHICLE.

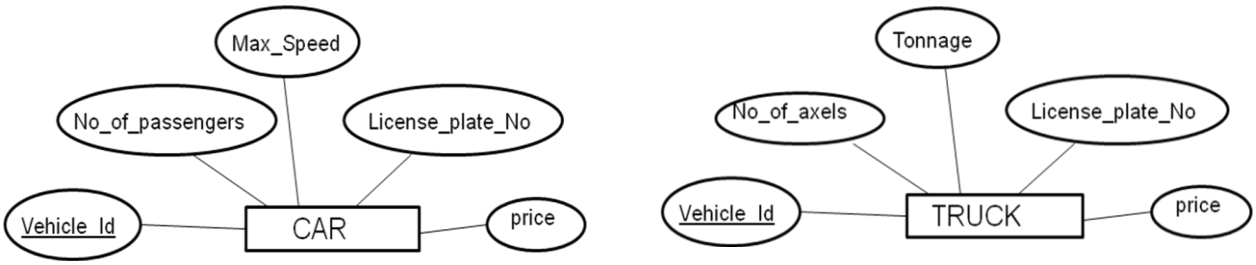



Diagram-A

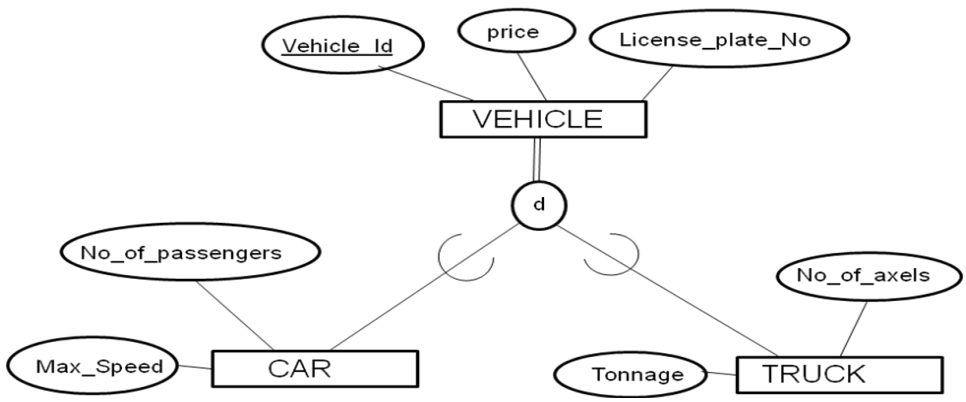
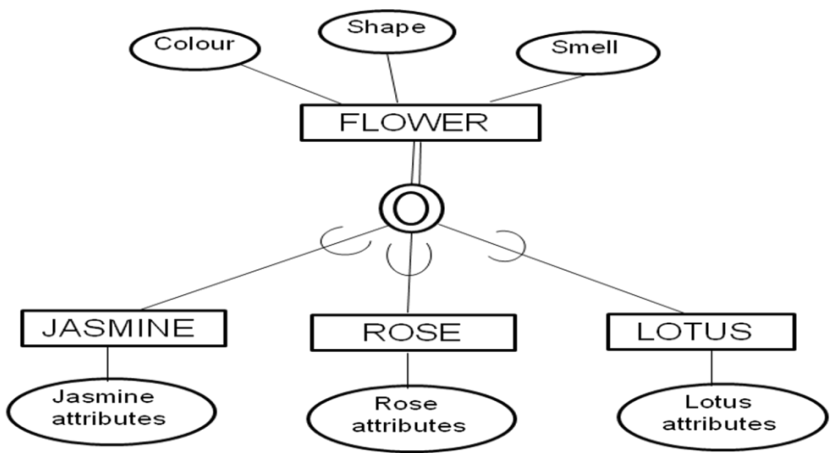



Diagram-B

Here, we are using the term 'generalization' to refer the process of defining a generalized entity types. The set {CAR, TRUCK} is the specialization of VEHICLE rather than viewing VEHICLE as a generalization of CAR and TRUCK. An arrow pointing to the generalized superclass represents a generalization, whereas arrows pointing to the specialized subclass represent a specialization.

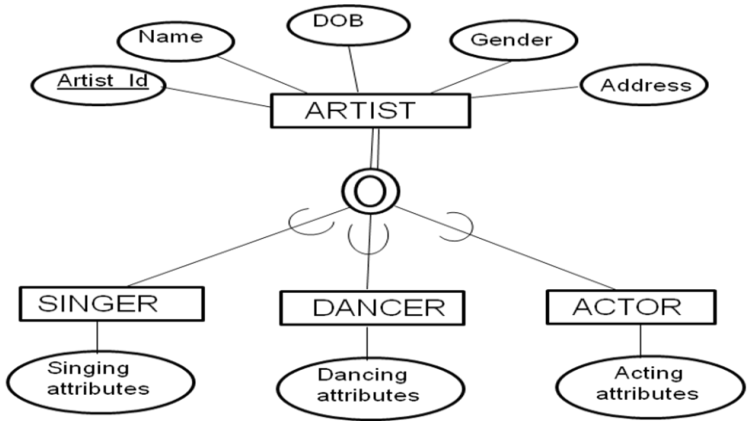
Example-2 : Sometimes different entity types are actually specializations of a more general entity type:

Basically, rose, jasmine, lotus are all flowers. Here, some attributes are common to all, others are specific to one entity type which are represented by generalization hierarchy. Subtypes may be disjoint or overlapping type. The attributes that are common belong to the supertype and those that are specific belong to the particular subtype.



This depicts a disjoint subset. For example, a particular flower that is jasmine can only belong to the entity set jasmine and it cannot belongs to set rose.

Overlapping Subtypes :




The above diagram depicts a overlapping subtype relationship.

Example : An artist is basically a human being and he is always identified by the common attributes like artist_id, name, dob, gender and an address. An artist is considered here as a entity of supertype, but it can be extended as subtypes {singer, dancer, actor}. Singer is a subtype under artist and he is having singing attributes along with the common attributes of artist. Similarly, the case of dancer and actor.

Case Study : Here, it is given with a problem statement with their requirements. We need to read the case study carefully and then do analysis with following steps.

i) Identify all possible existing entities

ii) Identify all given attributes for all the identified entities.

iii) Identify the key attributes for every entity

iv) Identify strong and weak entities

v) Identify all possible relationships among entities based on the given case study

vi) Identity the type of the cardinality ratios on all the identified relationships

vii) Identify the type of participation on entities with relationships

viii) After all these steps, draw an ER diagram which represents all the requirements of the case study.

i) Banking scenario :

- A Bank is identified by name, Id number, location of the main office and an address
- A bank has many branches
- Each branch is identified by branchId, branch name, and an address
- A branch has many customers who hold the account
- A customer is identified by customerId, customer_name and an address
- Account is held by Customer and a customer may have single account and many accounts (SB account, Joint Account, Business Account)
- Each Account is identified by account_no, account_type, balance_amount
- A bank offers Loans to customers through branches
- Loans are identified by loanId, loanType (House, Car, Business, Personal ) , loan_amount

Draw an ER diagram to represent this application

Solution :

Step-1 : Identify all Entities :

Bank, Branch, Customer, Account, Loan

Step-2 : Identify all given attributes for all the identified entities.

Bank { Bank_Id, Bank_name, Bank_Location}

Branch { Branch_Id, Branch_Name, Branch_Address}

Customer { Customer_Id, Customer_Name, Customer_Address}

Account { Account_No, Account_Type, Balance_Amount}

Loan { Loan_Id, Loan_Type, Loan_Amount}

Step-3: Identify the key attributes for every entity

Bank { **Bank_Id**}, Branch { **Branch_Id**}, Customer { **Customer_Id**},

Account { **Account_No**}, Loan { **Loan_Id**}

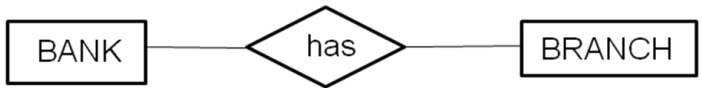
Step-4: Identify strong and weak entities

Strong entities : Bank, Customer, Account, Loan

Weak entities : Branch

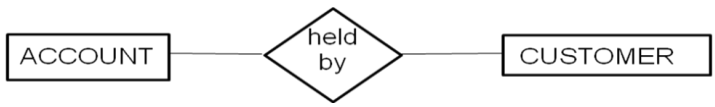
Step-5 : Identify Relationships among all entities based on the given case study :

Bank has branches



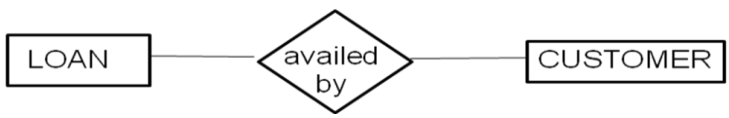

Branch maintains account


Account is held by Customer




Branch Offers Loans


Customers can avail loans

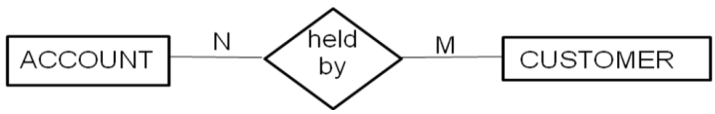



Step-6 : Identify cardinality ratios among the entities with their relationships

Bank has many branches → 1 : M

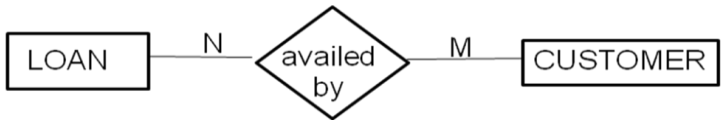

Branch maintains Accounts → 1 : M


Account is held by Customer → N : M




Branch Offers Loans → 1 : N

Customers can avail loans → N : M

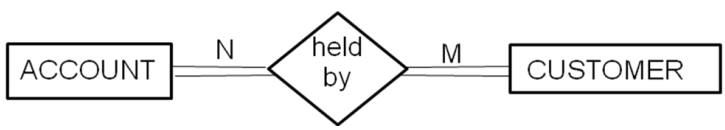



Step - 7 : Identify type of participation among entities with their relationships :

Bank has many branches → Bank : Total Branches : Total

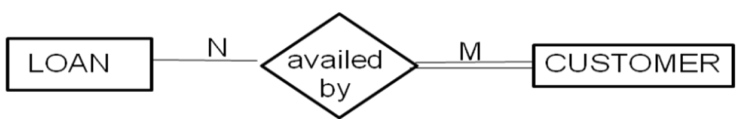

Branch maintains Accounts → Branch : Partial Account : Total


Account is held by Customer → Account : Total Customer : Total



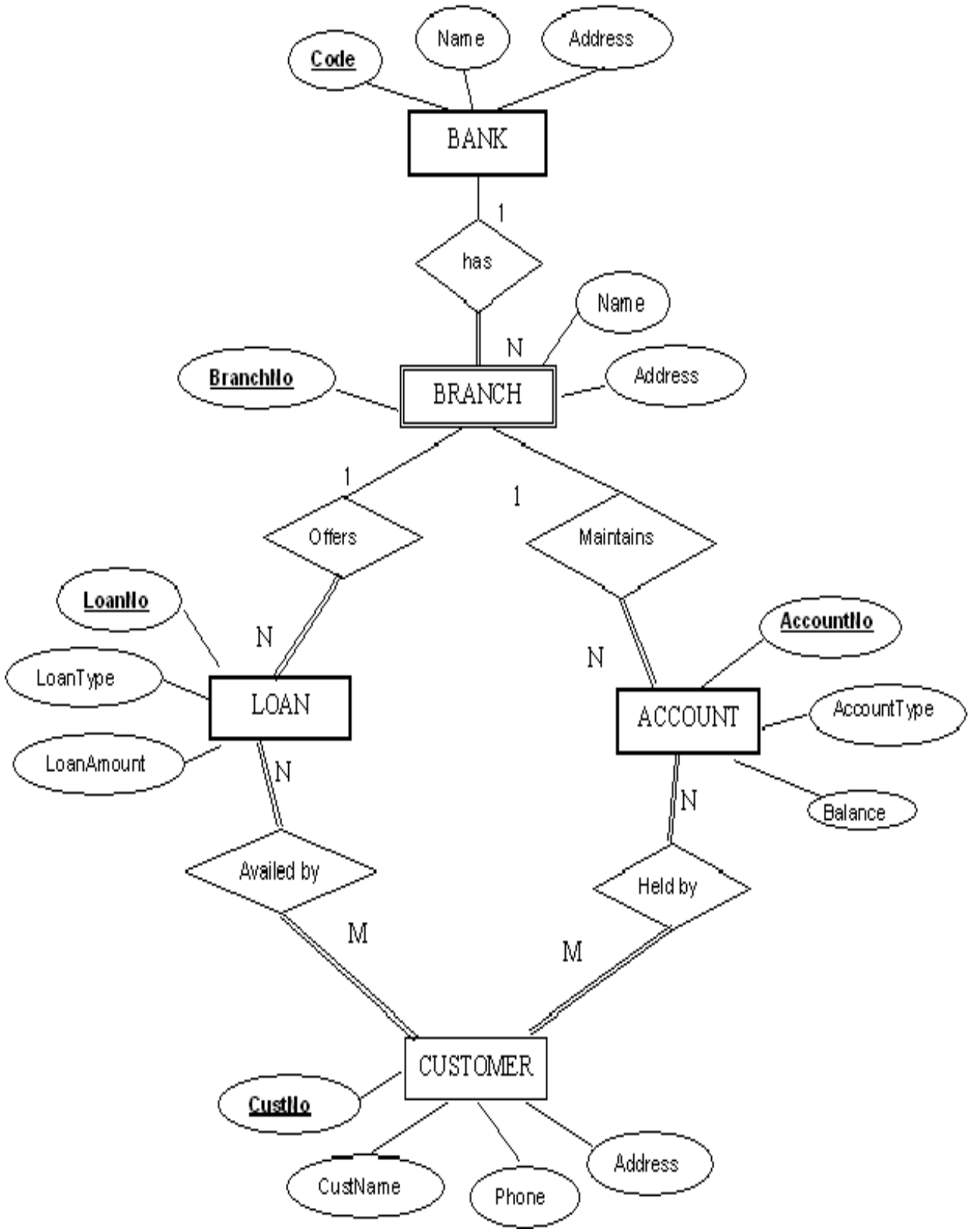

Branch Offers Loans → Branch : Total Loans : Total


Customers can avail loans → Customers : Partial Loans : Total




Step-8 : Write an ER diagram :

Note : Here, we have given identification of relationships, cardinality ratios and participation types in separate steps (i.e. step-5,step-6,step-7), But we can also write these steps in one single step.


Code
Name
Address
BANK
1
has
N
Name
Address
BranchNo
BRANCH
1
1
Offers
Maintains
LoanNo
N
LoanType
LOAN
LoanAmount
N
N
AccountNo
ACCOUNT
AccountType
N
Balance
Availed by
Held by
M
M
CUSTOMER
CustNo
CustName
Phone
Address

## ii) A Company Scenario :

- Company
  - Organized into departments, Each department has a name, number and manager who manages the department. The company keeps track of the date that employee managing the department. A department may have a several locations.
- Department
  - A department controls a number of projects each of which has a unique name, number and a single Location.
- Employee
  - Name, Age, Gender, BirthDate, SSN, Address, Salary. An employee is assigned to one department, may work on several projects which are not controlled by the department. Track of the number of hours per week is also controlled.
- Dependant
  - Keep track of the dependents of each employee for insurance policies : We keep each dependant first name, gender, date_of_birth and relationship to the employee.

Draw an ER diagram to represent this application

Solution :

Step-1 : Identify all Entities :

Employee, Department, Project and Dependent

Step-2 : Identify all given attributes for all the identified entities.

Employee { SSN, Name (Fname, Lname), Bdate, Sex, Salary, Address }

Department { DeptNo, Dept_Name, Location}

Project { Project_No, Project_Name, Project_location}

Dependent { Name, Bdate, Sex, Relationship }

Step-3: Identify the key attributes for every entity

Employee { SSN}, Department { DeptNo}, Project { Project_No}, Dependent { Name}

Step-4: Identify strong and weak entities

Strong entities : Employee, Department, Project

Weak entities : Dependent

Step-5 : Identify Relationships, Cardinality ratios and Participation types among all entities :

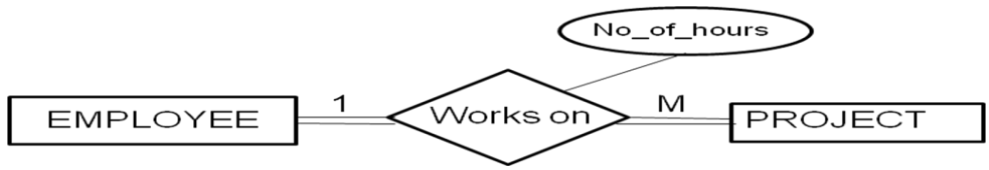
Employee manages the department


Employee works for department

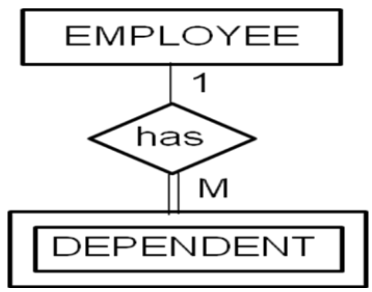

Department controls project


Employee supervises his subordinates

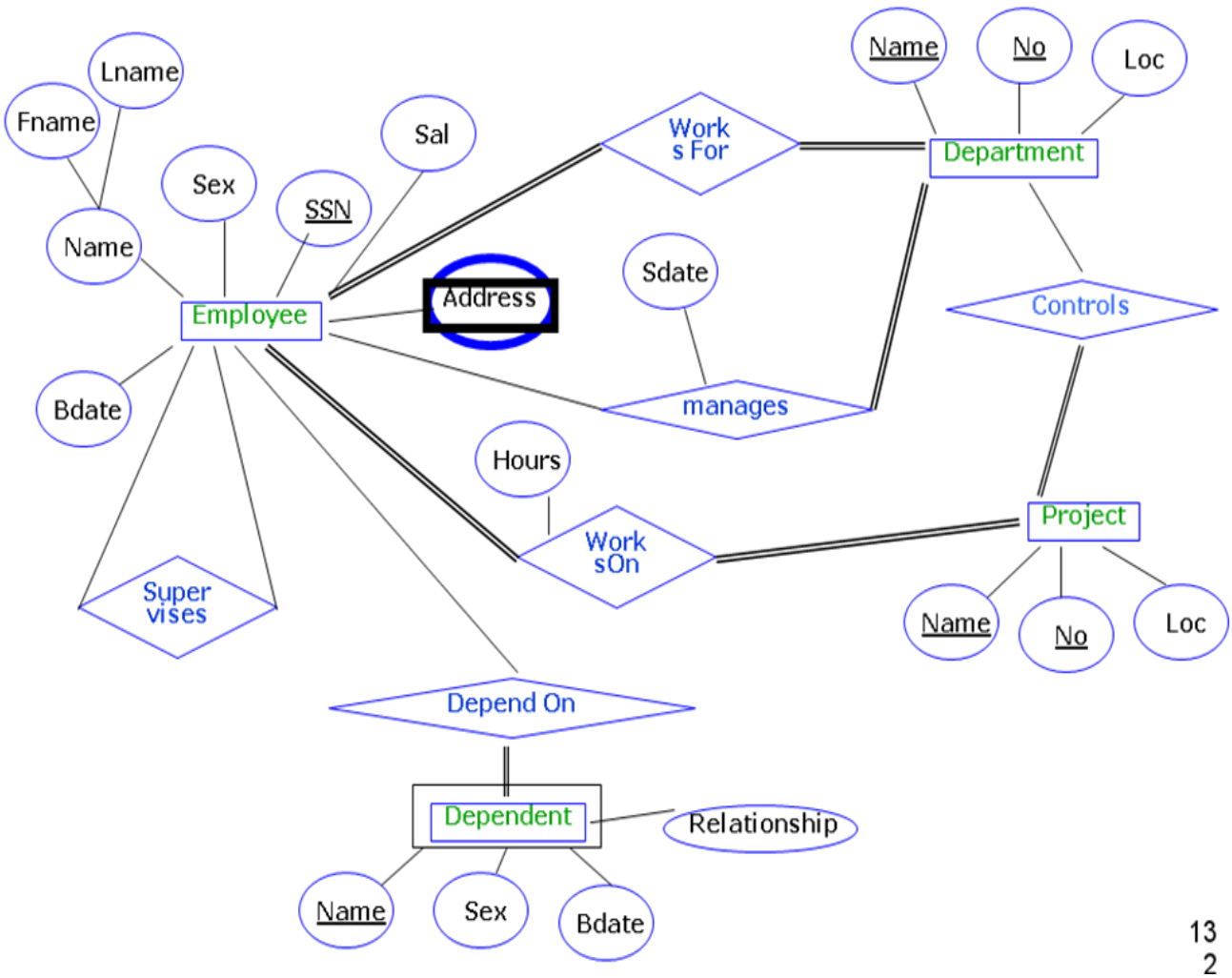

Employee works on project




Employee has dependents



Step-6 : Write an ER diagram :




## Converting entity types to table :

i) Each entity type becomes a table.

ii) Each single-valued attribute becomes a column.

iii) Derived attributes are ignored.

iv) Composite attributes are represented by its equivalent parts.(i.e. each extended attribute will become a individual column)

v) Multi-valued attributes are represented by a separate table.

vi) The key attribute of the entity type will becomes the primary key of the table.

## Entity – Relationship examples :

i) Employee has many skills

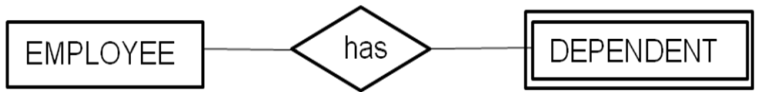
Employee { Employee#, EmpName, DoorNo, Street, City, Pincode, Date_of_join)

AND

Emp_skillset { Employee#, Skillset)

In the above table Employee, Employee# is the primary key and same key acts as a foreign key to the child table Emp_skillset.

ii) Employee has dependents




Employee { Employee#, EmpName, Date_of_join, Designation)

AND

Dependent { Employee#, DependentId, DependentName, Gender, DOB, Age)

## Converting Relationships :

## Binary relationship 1:1 :

i) Employee is heading the department


Employee { Employee#, EmpName, Date_of_join, Designation)

AND

Department { Department#, DepartmentName, Location, Head)

In the above table Employee, Employee# is the primary key and same key acts as a foreign key to the child table department in the name head. In department table Department# acts as a primary key.

ii) Employee sits on chair


Employee { Employee#, EmpName, Date_of_join, Designation)

Chair { Chair#, Model, Location, Used_by)

In the above table Employee, Employee# is the primary key and same key acts as a foreign key to the child table Chair in the name Used_by. In Chair table Chair# acts as a primary key.

It can also be designed in the following way.

Employee { Employee#, EmpName, Date_of_join, Sits_on)

AND

Chair { Chair#, Model, Location )

In the above table Chair, Chair# is the primary key and same key acts as a foreign key to the child table Employee in the name Sits_on. In Employee table Employee# acts as a primary key.

## Binary relationship 1: N

Teacher teaches a subject:


Teacher { Teacher#, TeacherName, ContactNo, EmailId, Branch)

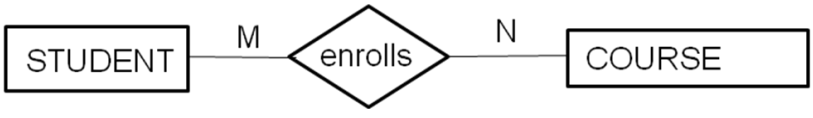
AND

Subject { Subject#, SubjectName, Duration, Taught_by)

In the above table Teacher, Teacher# is the primary key and same key acts as a foreign key to the child table Subject in the name Taught_by. In Subject table Subject# acts as a primary key.

## Binary relationship M : N

Student enrolls course:




Student { Student#, StudentName, DOB..) Course { Course#, CourseName, Duration)

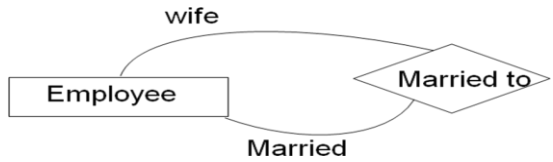
AND

Enrolls { Student#, Course#, )

In the above table Student, Student# is the primary key and same key acts as a foreign key to the child table Enrolls. In Course table, Course# is the primary key and same key acts as a foreign key to the child table Enrolls.

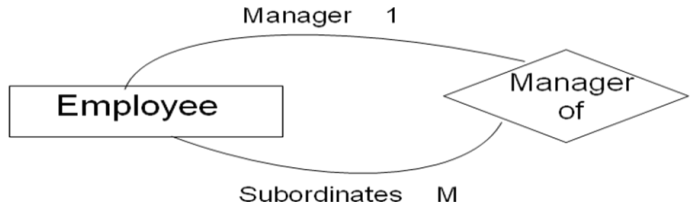
## Unary relationship : 1 : 1

Employee married with another employee ( Assuming both husband and wife are working in the same company)




Employee { Employee#, EmployeeName, DOJ, Designation, Salary, Spouse)

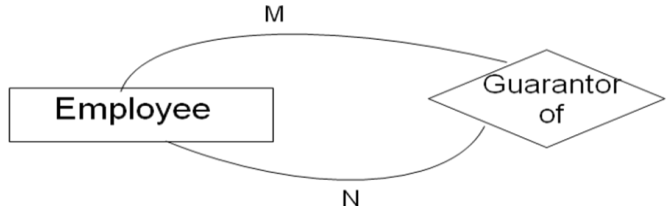
## Unary 1 : M :-




Employee { Employee#, EmployeeName, DOJ, Designation, Salary, Manager}

In the above table Employee, Employee# is the primary key and it acts as a foreign key in the same table in the name of Manager. This foreign key is also called as self referential key. It is an example for unary relationship. This shows that, one primary key acts as a foreign key in same table itself. Because, both manager and employee are employees working in the same organization.

## Unary M : N :-



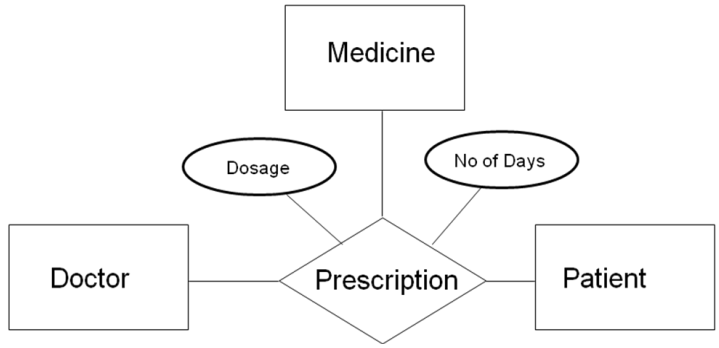

Employee { Employee#, EmployeeName, DOJ, Designation, Salary}

AND

Guaranty { Guarantor, Beneficiary )

In the above table Employee, Employee# is the primary key and same key acts as a foreign key to the child table Guaranty in the name of Guarantor and Beneficiary. This shows that, one primary key acts as many foreign keys in one table itself. Because, both guarantor and beneficiary are employees working in the same organization.

## Ternary Relationship :-




Doctor { Doctor#, DoctorName, Specialization, PlaceofWork, ContactNo}

Medicine { Medicine#, MedicineName, MedicineType, Manufactureddate, ExpiryDate}

Patient { Patient#, PatientName, SufferingDesease, Age, Gender, Address, ContactNo}

Prescription { Doctor#, Medicine#, Patient#, From_Date, To_date, Dosage, No_Times_a_day }

In the above table Doctor, Doctor# is the primary key and same key acts as a foreign key to the child table Prescription. In the table Medicine, Medicine# is the primary key and same key acts as a foreign key to the child table Prescription. In the table Patient, Patient# is the primary key and same key acts as a foreign key to the child table Prescription. In the child table prescription, there are three foreign keys, doctor#, medicine#, patient# and other attributes. This means that a doctor is prescribing a dosage of medicine to the patient about number of times a day from starting date to ending date.

## The Relational Data Model and Relational Database Constraints :

Relational Model concepts : The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or file of records. Each row in the table represents collection of related data values and a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

In the formal relational model terminology, a row is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is represented by a domain of possible values.

## Domains, Attributes, Tuples and Relations :

Relation : It is a table. The way storing the data in the form rows and columns.

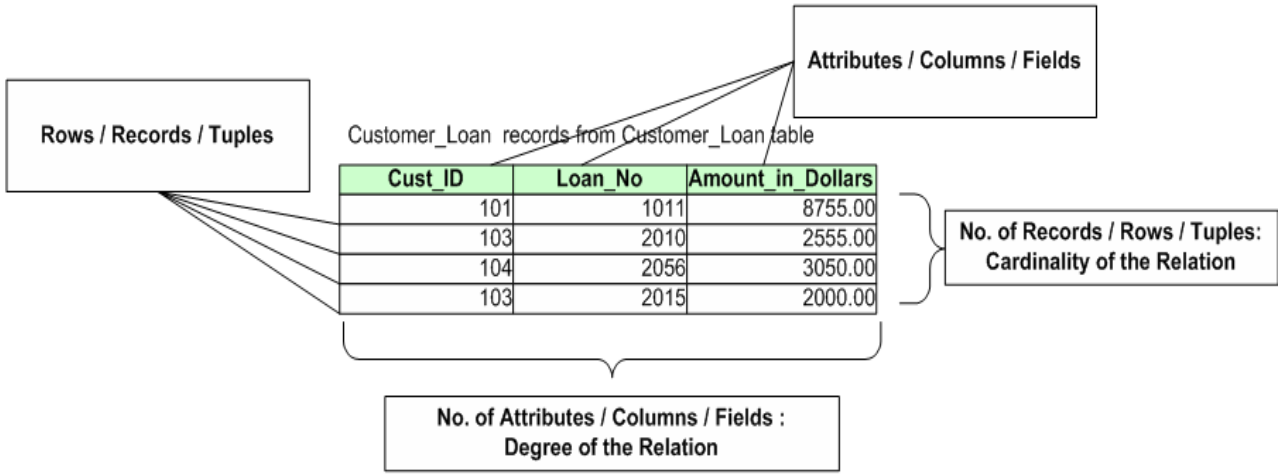
Tuple : Each row in a table is called tuple. It is also called record.

Field / Column header : Attribute of a table is called a column/ field.

Domain : The data type describing the types of values that can appear in each column is called as domain. In simple sense, range / set of values for a given column.

Degree of a relation : Number of columns present in a relation is called a degree of relation. i.e. number of columns in a table.

Cardinality of relation : Number of tuples in a relation is called a cardinality of a relation. i.e. number of records / rows in a table.




| Cust_ID | Cust_Last_Name | Cust_Mid_Name | Cust_First_Name | Account_No | Account_Type | Bank_Branch | Cust_Email |
|---|---|---|---|---|---|---|---|
| 101 | Smith | A. | Mike | 1020 | Savings | Downtown | Smith_Mike@yahoo.com |
| 102 | Smith | S. | Graham | 2348 | Checking | Bridgewater | Smith_Graham@rediffmail.com |
| 103 | Langer | G. | Justin | 3421 | Savings | Plainsboro | Langer_Justin@yahoo.com |
| 104 | Quails | D. | Jack | 2367 | Checking | Downtown | Quails_Jack@yahoo.com |
| 105 | Jones | E. | Simon | 2389 | Checking | Brighton | Jones_Simon@rediffmail.com |

records from Customer_Details table

- A relation schema 'R', denoted by $R(A_1, A_2, - - - , A_n)$, is made up of a relation name 'R' and a list of attributes $A_1, A_2, - - - - , A_n$.
- Each attribute Ai is the name of a role played by some domain 'D' in the relation schema 'R'. 'D' is called the domain of $A_i$ and is denoted by $dom(A_i)$.
- A relation schema is used to describe a relation ; 'R' is called the name of this relation.
- The degree (or arity) of a relation is the number of attributes 'n' of its relation schema.
- A relation is a set of n-tuples $r = \{t_1, t_2, - - - -, t_n\}$. Each n-tuple 't' is an ordered list of n values $t = <v_1, v_2, - - - - - V_n>$ where each value Vi, $1 <= i <= n$ is an element of $dom(A_i)$ or is a special NULL value.

Example : A relation schema for a relation of degree Seven, which describes university students,

STUDENT { Name, Ssn, Home_Phone, Address, Office_phone, Age, Gpa}

For this relation, STUDENT is the name of the relation, with seven attributes (i.e. Name, Ssn, Home_Phone, Address, Office_phone, Age, Gpa )

## Characteristics of a Relations :

Ordering of tuples in a relation : A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them, hence, tuples in a relation do not have any particular order.

Ordering of values within a tuple and an alternative definition of a relation (i.e. ordering of attribute values ) : The order of attributes and their values is not important as long as the correspondence between attributes and values is maintained. When a relation is implemented as a file, the attributes are physically ordered as fields with in a record. We generally use the first definition of a relation, where the attributes and the values within tuples are ordered, because, it simplifies much of the notation.

Values and NULLs in the tuples : Each value in a tuple is an atomic value; that is it is not divisible into components within the framework of the basic relational model. Hence, composite and multivalued attributes are not allowed. Multivalued attributes must be represented by separate relations and composite attributes are represented only by their component attributes in the basic relational model.

There are several meanings for NULL values, such as unknown value, empty value, undefined value and value exists but is not available…etc. NULL values arise due to several reasons at the time of data entry into the relations.

Interpretation of a relation : The relation schema can be interpreted as a declaration or a type of assertion. Each tuple in a relation can be interpreted as a fact or a particular instance of the assertion. Notice that some relations may represent facts about entities, whereas other relations may represent facts about the relationships. Hence, the relational model represents facts about both entities and relationships uniformly as relations.

In simple sense :

i) There are no duplicate rows / tuples

ii) Tuples are unordered, top to bottom

iii) Attributes are unordered, left to right

iv) All attribute values are atomic in nature

v) Relational databases do not allow repeating groups

Relational Model Constraints :

Constraints are nothing but restrictions applied on relations in a database. Generally, there are many constraints on the actual values in a database state. These constraints are derived from the rules in the miniworld that the database represents. Constraints on database can generally be divided into three main categories :

i) Constraints that are inherent in the data model called as inherent model-based constraints or implicit constraints
ii) Constraints that can be directly expressed in schemas of the data model called as schema-based constraints or explicit constraints
iii) Constraints that cannot be directly expressed in schemas of the data model, and hence must be expressed and enforced by the application programs called as application-based constraints or semantic constraints or business rules.

In general, constraints are classified as

- Entity integrity constraints or Key constraints
- Domain constraints
- Referential integrity constraints

i) Entity – Integrity constraints or Key constraints : A relation is defined as a set of tuples. By the definition, all the elements of a set are distinct, hence, all the tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes. This is possible with the inclusion of key constraints in a relation.

Examples : Primary key, Candidate key and Super key.

Primary Key **:** An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set. We also call it as a primary key. It can be applied on one column or multiple columns in a relation. For a given table / relation, we can include maximum of only one primary key. By default, primary key does not accept NULL value.

NOTE : Inclusion of a primary key to every relation / table in a database is not mandatory. It depends on our requirement.

Super Key : If we add additional attributes to a key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called as super keys.

Candidate Keys : There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called candidate keys.

- Primary Key
- Alternate Key

Example : In an employee relation, ContactNo and EmailID are the candidate keys, because, their values can also be considered to identify every record / tuple uniquely.

Secondary Keys : A secondary key is an attribute or combination of attributes that may not be a candidate key but that classifies the entity set on a particular characteristic.
A case in point is the entity set EMPLOYEE having the attribute department, which identifies by its value all instances EMPLOYEE who belong to a given department.

Simple Key :
Any key consisting of a single attribute is called a **simple key**

Composite Key :
Any key that consisting of a combination of attributes is called a **composite key**.

ii) Domain Constraints : Domain constraints specify that within each tuple, the value of each attribute Ai must be atomic value from the domain dom(A). These constraints mainly concentrates the type of the data values they accepts to the corresponding attributes / columns. We have three constraints with this type. NULL, NOT NULL and CHECK

NULL constraint : It is the constraint which facilitate the given column to accept the NULL value. No explicit create statement is required to include this constraint on any column. Because, by default every column is of NULL type except key attribute/column.

NOT NULL constraint : It is the constraint which restricts the given column to accept the NULL value. We have explicit create statement to include this constraint on any column.

CHECK constraint : It is the constraint which restrict the given column to accept only predefined value. It can be used generally for validation purpose. Explicit create statement is required to include this constraint on any column.

iii) Referential Integrity Constraints : These constraints establish the relationship between master table and child table. Foreign key is the only constraint available under this type.

FOREIGN KEY : The primary key of a master table acts as a foreign key to the child table. In this way, a relationship is formed.

Features of Foreign key :

- For a given table, any number of foreign keys can be included that depends upon the number of existing columns and our requirements.
- The primary key of one table can acts as a foreign key to any number of columns of the other tables.
- The primary key of one table can acts as a foreign key to more than one columns of the same table.
- The primary key of a table can also acts as a foreign key to another column of the same table. This is called as self referential key.
- During the creation of these tables, first, we need to create master table with a primary key and then to create child table with a foreign key.
- During the insertion of the data values, first we need to insert the data into master table and then to child table, because, child table's foreign key column will accept the value which is there in the master table's primary key column. If we enter other

value, it is giving an error message 'Integrity constraint violated, parent key not found'.

- During the deletion of the data values, first we need to delete the data from child table and then from master table, If we try to delete master table values, then it is giving an error message 'Integrity constraint violated, child record found'.
- A foreign key column can accept NULL value and redundant values.

## The Relational Algebra and Relational Calculus :

The basic set of operations for the relational model is the relational algebra. These operations enable a user to specify basic retrievals requests. The result of a retrieval is a new relation, which may have been formed from one or more relations. The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra. A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query.

The relational algebra is very important for several reasons.

First, it provides a formal foundation for relational model operations.

Second, it is used as a basis for implementing and optimizing queries in relational database management systems.

Third, some of its concepts are incorporated into the SQL, structured query language for RDBMSs.

Although, no commercial RDBMS in use today provides an interface for relational algebra queries, the core operations and functions of any relational system are based on relational algebra operations. Whereas the algebra defines a set of operations for the relational model, the relational calculus provides a higher –level declarative notation for specifying relational queries. A relational calculus expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations or over columns of the stored relations.

In a query result, a calculus expression specifies only what information the result should contain. This is the main distinguishing feature between relational algebra and relational calculus. The relational calculus is important, because, it has a firm basis in mathematical logic and because the standard query language for RDBMSs has some of its foundation in the tuple relational calculus.

The relational algebra is often considered to be an integral part of the relational data model. Its operations can be divided into two groups. One group includes set operations from mathematical set theory, these are applicable, because each relation is defined to be a set of tuples in the formal relation model. Set operations include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT. The other group consists of operations developed specially for relational databases, these include SELECT, PROJECT and JOIN.

## Relational Operations

The PROJECT operation : It selects certain columns from the table and discards the remaining columns. If we want to retrieve only certain attributes of a relation or all the attributes, we use the PROJECT operation. The general form of the PROJECT operation is

$$\pi_{<\text{attribute list}>}(R)$$

Here, π is the symbol used to represent Projection, <attribute list> is the list of columns (attributes) of a relation R and (R) is the relation name (table name)

How to write SQL queries in relational algebraic expressions ?

Example : To display employee details from the table emp, the SQL query can be written as

```
SQL> select empno, empname, job from emp;
```

The same SQL query can be represented in relational algebraic expression as follows :

$$R \rightarrow \pi_{<\text{empno, empname, job}>}(\text{Emp})$$

The SELECT operation : It is used to select a subset of the records from a relation that satisfies a select condition. It acts like a filter by selecting only required records by putting the condition in a query. The general form of the SELECT operation is

$$\sigma_{<\text{select condition}>}(R)$$

The Boolean condition specified in <select condition> is made up of a number of clauses of the form

<attribute name> <comparison operator> <constant value>

Here, attribute name is column name, comparison operator is equal operator '=' and constant value is a user defined value.

To display employee details from the table emp with the designation CLERK, the SQL query can be written as

```
SQL> select empno, empname, job from emp where job='CLERK' ;
```

The above query is a combination of projection and selection. It can be represented in relational algebraic expression as follows :

$$R1 \rightarrow \pi_{<\text{empno, empname, job}>}(\text{Emp})$$

$$R2 \rightarrow \sigma_{<\text{job='CLERK'}>}(\text{Emp})$$

$$R \rightarrow R1(R2)$$

The same query can also be represented as follows :

$$R \rightarrow \pi_{<empno, empname, job>} (\sigma_{<job='CLERK'>} (Emp))$$

The JOIN operation : It is used to combine related records from one, two or more relations in to a single record. A general form of JOIN operation on two relations is

$$R \infty_{<join\ condition>} S$$

Here R is the first relation, S is the second relation,
<join condition> contains R.Column=S.Column, R and S are the two relations, Column is the common column available from both the relations.

If number conditions increases, then conditions will be included by using Logical operators (i.e. AND, OR, NOT)

Example1 : To display employee details from the table emp and dept, the SQL query can be written as

```
SQL> select empno, empname, job, dname
     from emp, dept
      Where emp.deptno=dept.deptno
```

The above query is a combination of projection, join. It can be represented in relational algebraic expression as follows :

$$R1 \rightarrow \pi_{<empno, empname, job,dname>} (Emp, Dept)$$
$$R2 \rightarrow Emp \infty_{<deptno = deptno>} Dept$$
$$R \rightarrow R1(R2)$$

The same query can also be represented as follows :

$$R \rightarrow \pi_{<empno, empname, job,dname>} (Emp \infty_{<deptno = deptno>} Dept)$$

Example2: To display employee details from the table emp and dept with the designation CLERK, the SQL query can be written as

```
SQL> select empno, empname, job, dname
     from emp, dept
      Where emp.deptno=dept.deptno
       and  job='CLERK' ;
```

The above query is a combination of projection, selection and join. It can be represented in relational algebraic expression as follows :

R1→ $\pi$< empno, empname, job,dname> (Emp, Dept)

R2 → Emp $\infty$ < deptno = deptno > Dept

R3 → $\sigma$ < job='CLERK'> (Emp, Dept)

R → R1(R2(R3))

The same query can also be represented as follows :

R→ $\pi$< empno, empname, job,dname> ($\sigma$ < job='CLERK'> (Emp $\infty$ < deptno = deptno > Dept))

The DIVISION operation : It is used for a special kind of query that sometimes occurs in database applications. It is denoted by ÷ , In general, the DIVISION operation is applied to two relations

R(Z) ÷ S(X), where X $\subseteq$ Z .

Let Y = Z – X ( and hence Z = X U Y); that is, let Y be the set of attributes of R that are not attributes of S. The result of DIVISION is a relation T(Y) that includes a tuple 't' if tuples $t_R$ appear in R with $t_R[Y] = t$, and with $t_R[X] = ts$ for every tuple ts in S. This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S.

The Cartesian Product : If two tables in a join query have no join condition, then query results in a type of a result set in the form of **Cartesian product**. Query combines each row of one table with each row of the other [5]. It is in the form of M X N. i.e. M is the number of records in first table and N is the number of records in the second table [4]. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables, each with 50 rows, has 2,500 rows. Therefore always be careful in using join conditions. If a query joins three or more tables and you do not specify a join condition for a specific pair, then the optimizer may choose a join order that avoids producing an intermediate Cartesian product.

## Relational Algebra Operations from Set Theory :

The next group of relational algebra operations are the standard mathematical operations on sets. Several set theoretic operations are used to merge the elements of two sets in various ways, that include UNION, INTERSECTION, and SET DIFFERENCE (MINUS). These are binary operations; that is, each is applied to two sets (of tuples) When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples. This condition has been called union compatibility.

Two relations $R(A_1, A_2, - - - A_n)$ and $S(B_1, B_2, - - - - B_n)$ are said to be union compatible if they have the same degree 'n' and if $dom(A_i) = dom(B_i)$ for $1 <= i <= n$. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

UNION : The result of this operation, denoted by R U S, is a relation that includes all tuples that are either in 'R' or in 'S' or in both R and S. Duplicate tuples are eliminated.

INTERSECTION : The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that includes all tuples that are in both R and S.

SET DIFFERENCE ( or MINUS) : The result of this operation, denoted by R - S, is a relation that includes all tuples that are in 'R' but not in 'S'.

Notice that both UNION and INTERSECTION are commutative operations, that is

R ∪ S = S ∪ R and R ∩ S = S ∩ R

Both UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are associative operations; that is

R ∪ ( S ∪ T ) = ( R ∪ S ) ∪ T and ( R ∩ S ) ∩ T = R ∩ ( S ∩ T )

The MINUS operation is not commutative; that is, in general,

R – S ≠ S - R

Aggregate Functions and Grouping : Another type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database. These functions are used in simple statistical queries that summarize information from database tuples. Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.

Another common type of request involves grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group. We can define an AGGREGATE FUNCTION operation using the symbol ' ζ ' to specify these types of requests.

<Grouping attributes> ζ <Function List> (R)

Where <Grouping attributes> is a list of attributes of the relation specified in R, and
<Function List> is a list of (< function > < attribute>) pairs.

In each such pair, <function> is one of the allowed functions such as SUM, AVG, MAX, MIN, COUNT and <attribute> is an attribute of the relation specified by R.

The resulting relation has the grouping attributes plus one attribute for each element in the function list.

To display the Number of employees according to their designation, we write SQL query in a following way :

```
SQL> select Job, count(job) as noofemps, sum(salary) as Total_Salary
        From emp
        Group by job;
```

The same SQL query can be represented in relational algebraic expression as follows :

R → < job > ζ < Count (job), Sum(salary) > (Emp)