

1.1 Introduction

A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.

A DBMS makes it possible for end users to create, read, update and delete data in a database. The DBMS essentially serves as an interface between the database and end users or application programs, ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified -- and the database schema, which defines the database's logical structure.

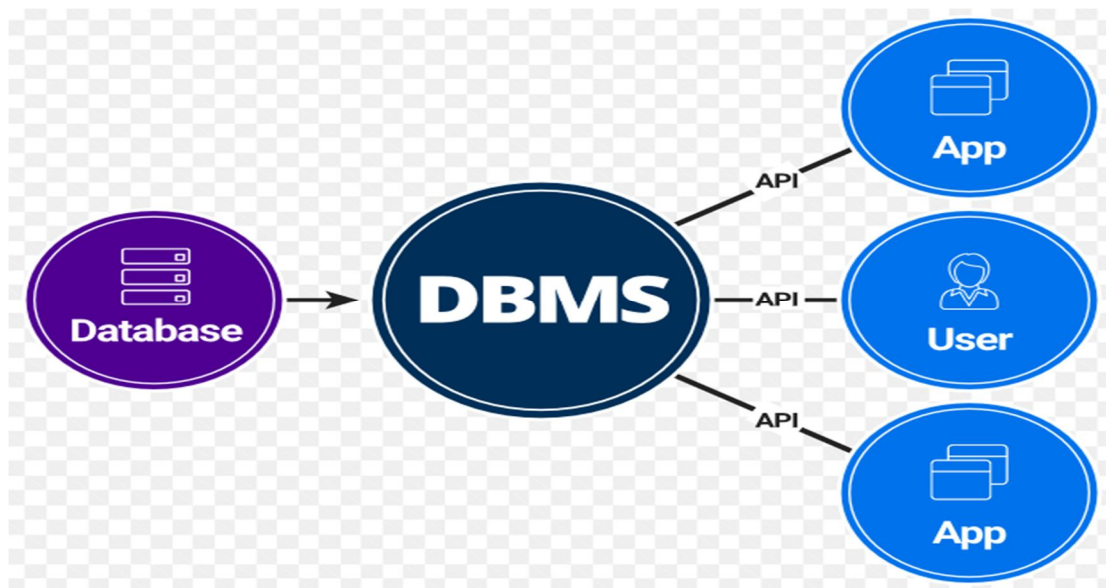


Figure: DBMS System.

DBMS act as a interface or intermediate between Database and user. It allows the user or application to interact with database in systematic way.

1.2 Database System Architecture:

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.

This architecture is a three layered architecture the bottom layer consist of the Database which consisting of sub components like data dictionary and data files with indices, middle layer is DBMS which consist of Query processor and stored data manager. Top layer is a view layer which consists of various users and applications and user trying to access data from database.

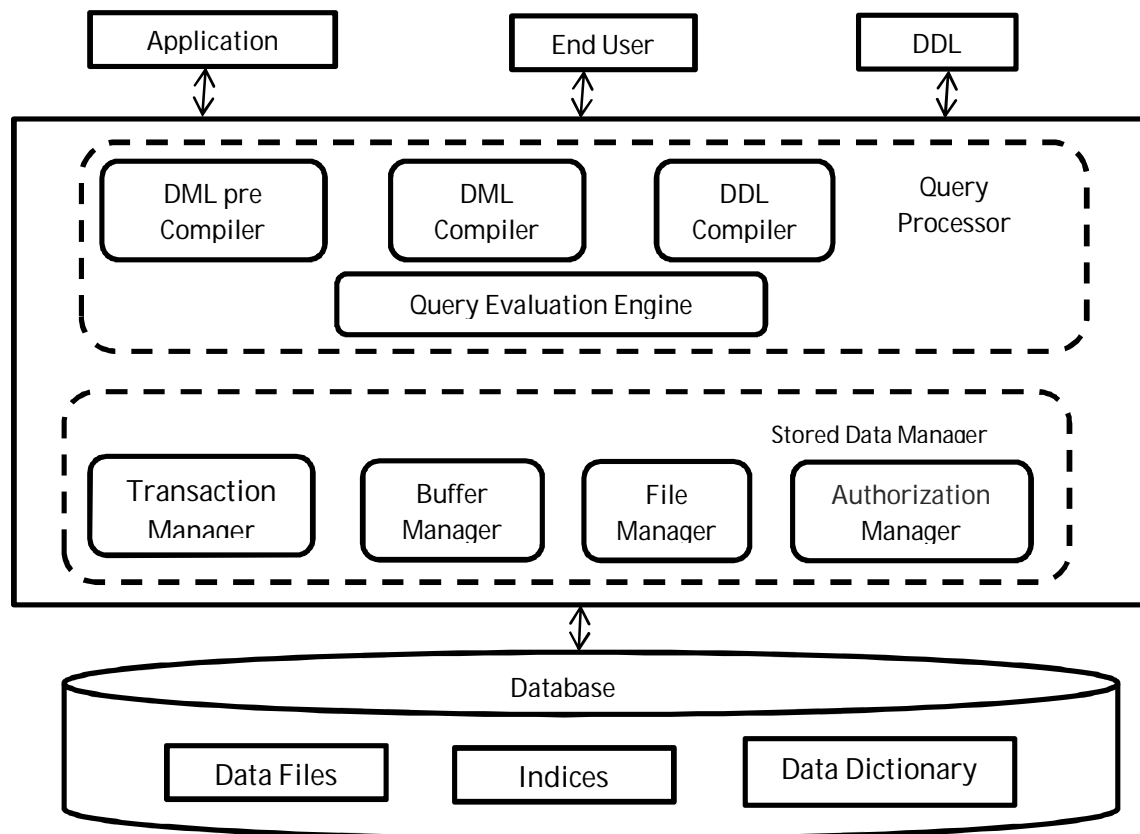


Figure: DBMS Architecture.

1. **DBMS Layer:** it is a middle layer consisting of two major components as follow.

A. Query Processor Components:

- **DML Pre-compiler:** It translates DML statements in a query language into low level instructions that query evaluation engine understands. It also attempts to transform user's request into an equivalent but more efficient form.
- **Embedded DML Pre-compiler:** It converts DML statements embedded in an application program to normal procedure calls in the host language. The Pre-compiler must interact with the DML compiler to generate the appropriate code.
- **DDL Interpreter:** It interprets the DDL statements and records them in a set of tables containing meta data or data dictionary.
- **Query Evaluation Engine:** It executes low-level instructions generated by the DML compiler.

B. Storage Manager Components: A storage manager is a program module which is responsible for storing, retrieving and updating data in the database. Following are the components of the storage manager;

- **Authorization and Integrity Manager:** It tests the integrity constraints and checks the authorization of users to access data.
- **Transaction Manager:** It ensures that no kind of change will be brought to the database until a transaction has been completed totally.
- **File Manager:** It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

➤ **Buffer Manager:** It decides which data is in need to be cached in main memory and then fetch it up in main memory. This is very important as it defines the speed in which the database can be used.

2. **Database Layer:** it is the Bottom layer consisting of following sub components.

➤ **Data Dictionary:** data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data.

The data dictionary is a crucial component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

➤ **Data Files:** A **data file** is file which stores data to be used by a application or user .

➤ **DBMS Indexing:** We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely. Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.

1.3 View Levels of DBMS Architecture/ Level of Data Abstraction in DBMS

This architecture describe the various views and levels of data abstraction in DBMS system which are as Physical Level, Conceptual Level and External Level

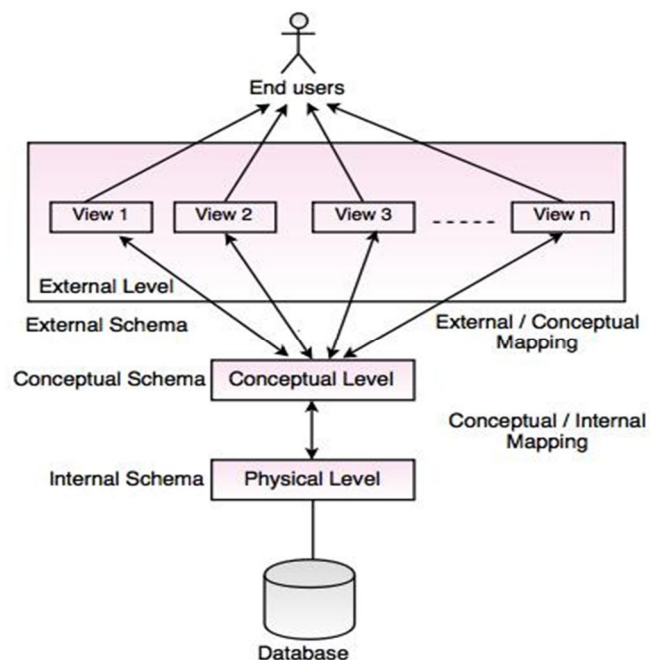


Figure: View Level/ Data Abstraction Level in DBMS

1. Physical Level: Physical level describes the physical storage structure of data in database, as well it also describe how the data is stored and retrieve in database and also describe compression and encryption techniques applied on data. 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 also called logical or global level describes the structure of the whole database for a group of users. It describe how the data will be appered to various user and relationship between various data. Conceptual schema is a representation of the entire content of the database. This schema contains all the information to build relevant external records. It hides the internal details of physical storage.

3. External Level: External level is also called view 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. The different user may have different views.

1.4 Data Models in DBMS

A **data model** is an abstract **model** that organizes elements of **data** and standardizes how they relate to one another and to properties of the real world entities. Data models describe how the data in database is get organized or stored and how it is related with each other's the different types of data models are as follow.

1. Relation model
2. Network model
3. Hierarchical model
4. Object oriented model
5. Context data model
6. Object relation model
7. Entity relation Model
8. Flat data model
9. Semi structured model
10. Associative model
11. Record base model

1. Relational Data Model: Relational model is the most popular model and the most extensively used model. In this model the data can be stored in the tables and this storing is called as relation, the relations can be normalized and the normalized relation values are called atomic values. Each row in a relation contains unique value and it is called as tuple, each column contains value from same domain and it is called as attribute.

SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

Figure: Relational Data Model

Advantages

- Structural independence – changes in the relational data structure do not affect the DBMS's data access in any way
- Improved conceptual simplicity by concentrating on the logical view
- Easier database design, implementation, management, and use
- Powerful database management system.

Disadvantages

- Can facilitate poor design and implementation

2. Network Data Model: Network model has the entities which are organized in a graphical representation and some entities in the graph can be accessed through several paths. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or lattice. It can preserve one to many or many to many relation.

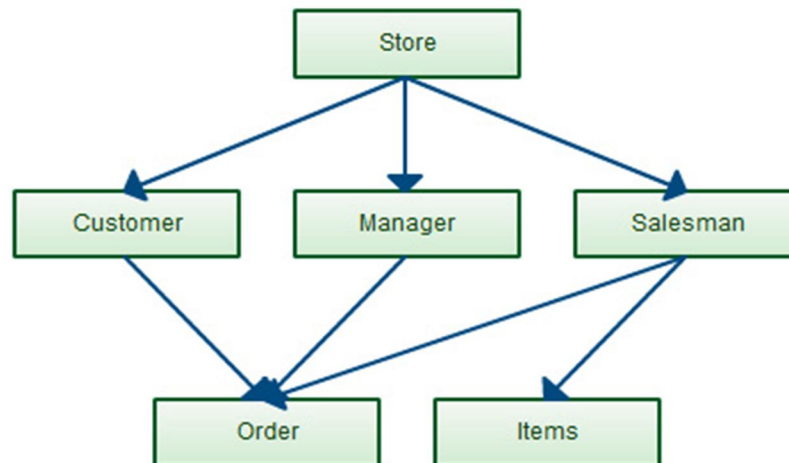


Figure: Network Data Model

Advantages

- Provide very efficient "High-speed" retrieval.
- Simplicity The network model is conceptually simple and easy to design.
- Ability to handle more relationship types The network model can handle the one-to-many and many-to-many relationships.
- Data Integrity In a network model, no member can exist without an owner.
- Data Independence The network model draws a clear line of demarcation between programs and the complex physical storage details.

Disadvantages

- System complexity In a network model, data are accessed one record at a time.
- Making structural modifications to the database is very difficult in the network database model as the data access method is navigational.

➤ Any changes made to the database structure require the application programs to be modified before they can access data.

3. Hierarchical Data Model: Hierarchical model has one parent entity with several children entity but at the top we should have only one entity called root. For example, department is the parent entity called root and it has several children entities like students, professors and many more.

A hierarchical database model is a data model in which the data are organized into a tree-like structure. The data are stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value. It preserves one to many relations.

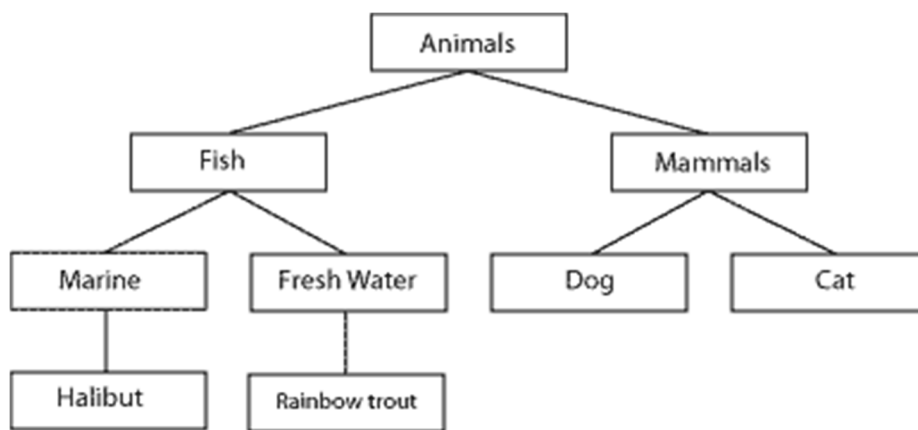


Figure: Hierarchical Data Model

Advantages

- **Simplicity:** it is easier to view data arranged in manner. This makes this type of database more suitable for the purpose.
- **Security:** This database system can enforce varying degree of security feature unlike flat-file system.
- **Database Integrity:** Because of its inherent parent-child structure, database integrity is highly promoted in these systems.
- **Efficiency:** The hierarchical database model is a very efficient, one when the database contains a large number of one-to-many relationships and when the users require large number of transactions, using data whose relationships are fixed.

Disadvantages

- **Complexity of Implementation:** The actual implementation of a hierarchical database depends on the physical storage of data. This makes the implementation complicated.
- **Difficulty in Management:** The movement of a data segment from one location to another cause all the accessing programs to be modified making database management a complex affair.
- **Complexity of Programming:** Programming a hierarchical database is relatively complex because the programmers must know the physical path of the data items.
- **Poor Portability:** The database is not easily portable mainly because there is little or no standard existing for these types of database.

➤ **Database Management Problems:** If you make any changes in the database structure of a hierarchical database, then you need to make the necessary changes in all the application programs that access the database. Thus, maintaining the database and the applications can become very difficult.

4. Object oriented Data Model: Object oriented data model is one of the developed data model and this can hold the audio, video and graphic files. These consist of data piece and the methods which are the DBMS instructions. OODBMS should be used when there is a business need, high performance required, and complex data is being used. Due to the object oriented nature of the database model, it is much simpler to approach a problem with these needs in terms of objects.

Advantages:

- Suitable for application expecting high performance.
- Due to the object oriented nature of the database model, it is much simpler to approach a problem

Disadvantages:

- complex data is being used
- Complex relations are used

5. Object relation Data Model: Object relation model is a very powerful model but coming to its design it is quiet complex. This complexity is not problem because it gives efficient results and widespread with huge applications. It has a feature which allows working with other models like working with the very known relation model.

Advantages of Object Relational model is inheritance. The Object Relational data model allows its users to inherit objects, tables etc. so that they can extend their functionality. An inherited object contains new attributes as well as the attributes that were inherited.

The object relational data model can get quite complicated and difficult to handle at times as it is a combination of the Object oriented data model and Relational data model and utilizes the functionalities of both of them. This hybrid database model combines the simplicity of the relational model with some of the advanced functionality of the object-oriented database model. In essence, it allows designers to incorporate objects into the familiar table structure.

6. Entity Relationship Data Model: This model represents the data diagrammatically. Entity relationship model is based on the notion of the real world entities and their relationships. While formulating the real world scenario in to the database model an entity set is created. 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. and this model is dependent on two vital things and they are 1. Entity and their attributes, 2. Relationships among entities.

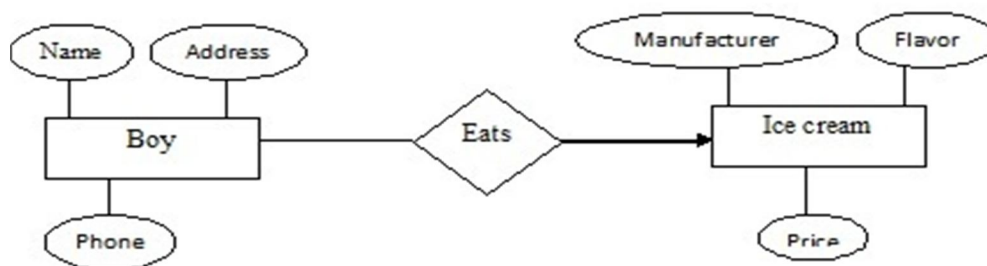


Figure: ER Data Model

7. Context Data Model: Context data model is a flexible model because it is a collection of many data models. It is a collection of the data models like object oriented data model, network model, semi structured model. So, in this different types of works can be done due to the versatility of it. A context model defines how context data are structured and maintained. It plays a key role in supporting efficient context management.

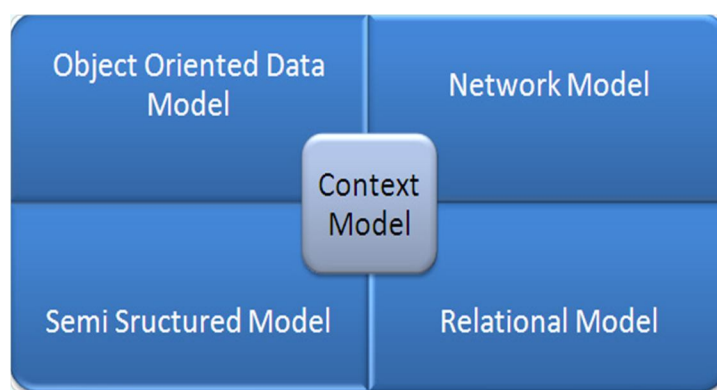


Figure: Context Data Model

8. Semi structured Data Model: Semi structured data model is a self-describing data model, in this the information that is normally associated with a scheme is contained within the data and this property is called as the self-describing property.

9. Associative Model: Associative model has a division property, this divides the real world things about which data is to be recorded in two sorts i.e. between entities and associations. Thus, this model does the division for dividing the real world data to the entities and associations. The associative model of data is a data model for database systems. Other data models, such as the relational model and the object data model, are record-based. These models involve encompassing attributes about a thing, such as a car, in a record structure. Such attributes might be registration, colour, make, model, etc. In the associative model, everything which has "discrete independent existence" is model as an entity, and relationships between them are modeled as associations.

10. Record base model: This model specifies the overall structure of database. Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

11. Flat Data Model: The flat model is the earliest, simplest data model. It simply lists all the data in a single table, consisting of columns and rows. In order to access or manipulate the

data, the computer has to read the entire flat file into memory, which makes this model inefficient for all but the smallest data sets.

1.5 Entity Relation Model

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

1.5.1 Components of E-R Model: The different components of ER Model are as follow.

1. Entity: An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.



2. Attributes: Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc. Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

Types of Attributes

- **Simple attribute** – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name. If the attributes are composite, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.
- **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth. derived attributes are depicted by dashed ellipse.

➤ **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.

➤ **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc. Multivalued attributes are depicted by double ellipse.

These attribute types can come together in a way like –

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

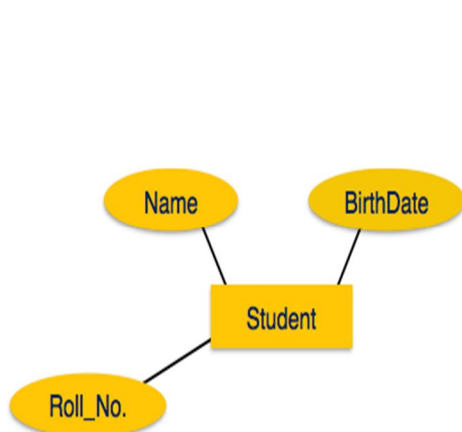


Figure: Simple Attributes

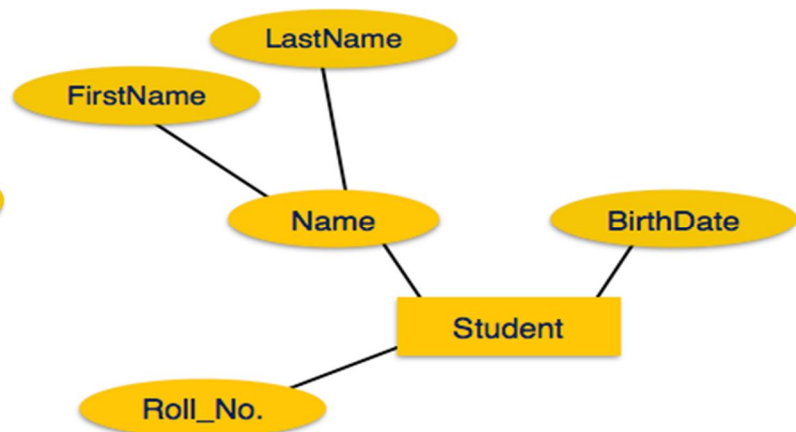


Figure: Composite Attributes (Name)

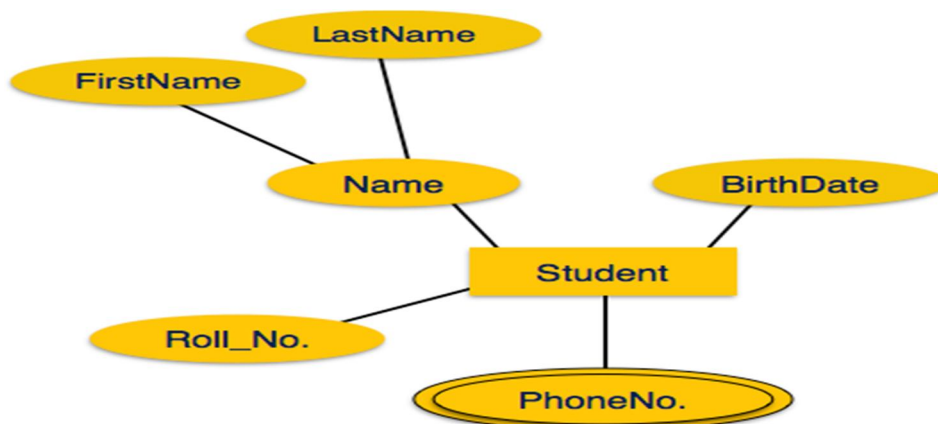


Figure 11: Multi Valued Attributes (PhoneNo)

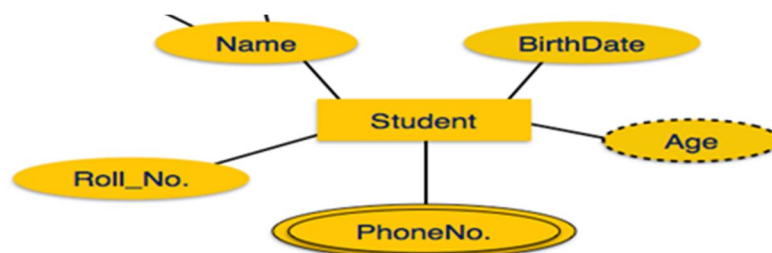


Figure : Derived Attributes (Age)

3. Entity-Set and Keys: Key is an attribute or collection of attributes that uniquely identifies an entity among entity set. For example, the roll_number of a student makes him/her identifiable among students.

➤ **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.

A superkey is a set of attributes within a table whose values can be used to uniquely identify a tuple. A candidate key is a minimal set of attributes necessary to identify a tuple; this is also called a minimal superkey. Given an employee schema consisting of the attributes employeeID, name, job, and departmentID, where no value in the employeeID attribute is ever repeated, we could use the employeeID in combination with any or all other attributes of this table to uniquely identify a tuple in the table. Examples of super keys in this schema would be {employeeID, Name}, {employeeID, Name, job}, and {employeeID, Name, job, departmentID}. The last example is known as trivial super key, because it uses all attributes of this table to identify the tuple.

➤ **Candidate Key** – A minimal super key is called a candidate key. An entity set may have more than one candidate key.

The minimal set of attribute which can uniquely identify a tuple is known as **candidate key**. For Example, STUD_NO in STUDENT relation. The value of **Candidate Key** is unique and non-null for every tuple. For **Example**, STUD_NO as well as STUD_PHONE both are **candidate keys** for relation STUDENT.

➤ **Primary Key** – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

4. Relationship: The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

5. Relationship Set: A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

6. Degree of Relationship: The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

7. Mapping Cardinalities: **Cardinality** defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

➤ **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.

➤ **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

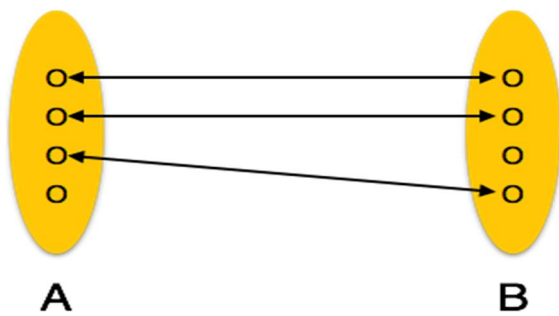


Figure: One To One Cardinality

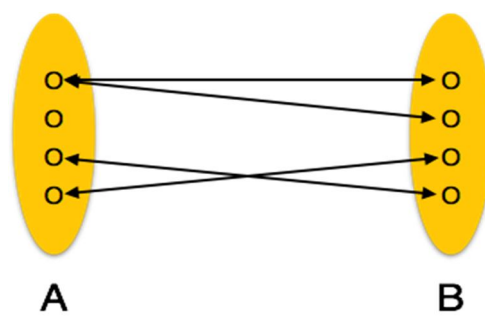


Figure : One To Many Cardinality

➤ **Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

➤ **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.

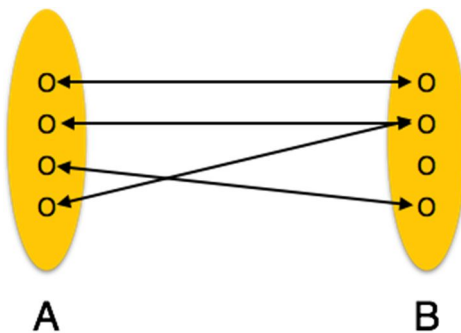


Figure : Many To One Cardinality

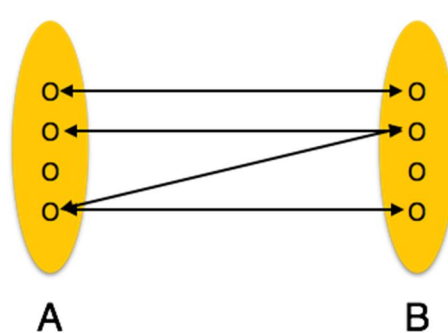
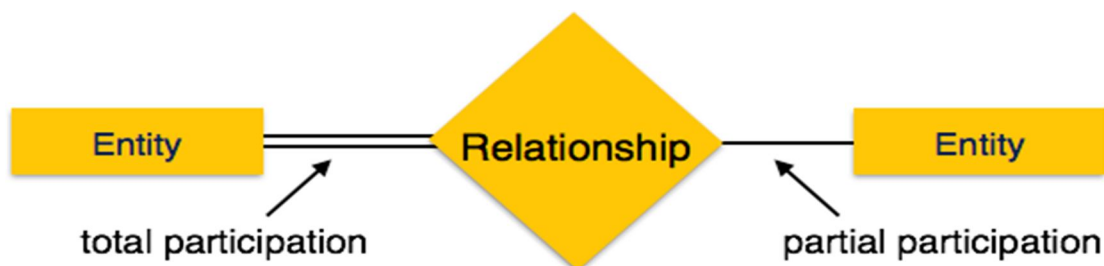


Figure: Many To Many Cardinality

8. Participation Constraints

➤ **Total Participation** – Each entity is involved in the relationship. Total participation is represented by double lines.

➤ **Partial participation** – Not all entities are involved in the relationship. Partial participation is represented by single lines.



The ER Model has the power of expressing database entities in a conceptual hierarchical manner. As the hierarchy goes up, it generalizes the view of entities, and as we go deep in the hierarchy, it gives us the detail of every entity included.

1.6 Enhanced ER Model: EER is a high-level data model that incorporates the extensions to the original ER model. It is a diagrammatic technique for displaying the following concepts.

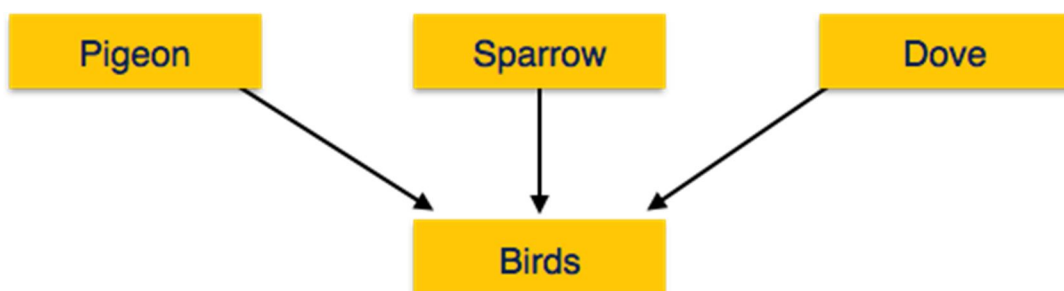
These concepts are used when the comes in EER schema and the resulting schema diagrams called as EER Diagrams.

- Sub Class and Super Class
- Specialization and Generalization
- Union or Category
- Aggregation

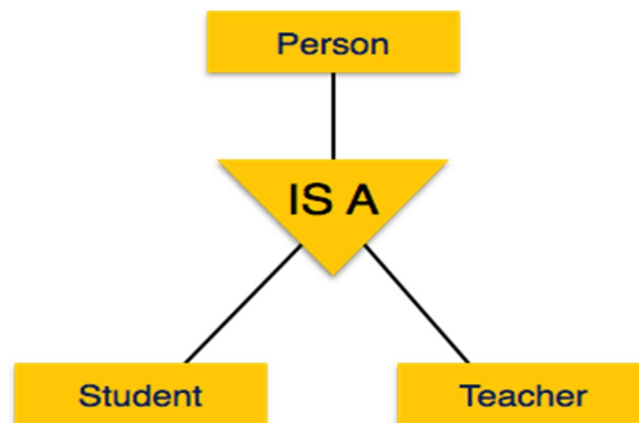
1.6.1 Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

1. **Generalization:** As mentioned above, the process of generalizing entities, where the generalized entities contain the properties of all the generalized entities, is called generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds. Generalization is the process of generalizing the entities which contain the properties of all the generalized entities. It is a bottom approach, in which two lower level entities combine to form a higher level entity. Generalization is the reverse process of Specialization.



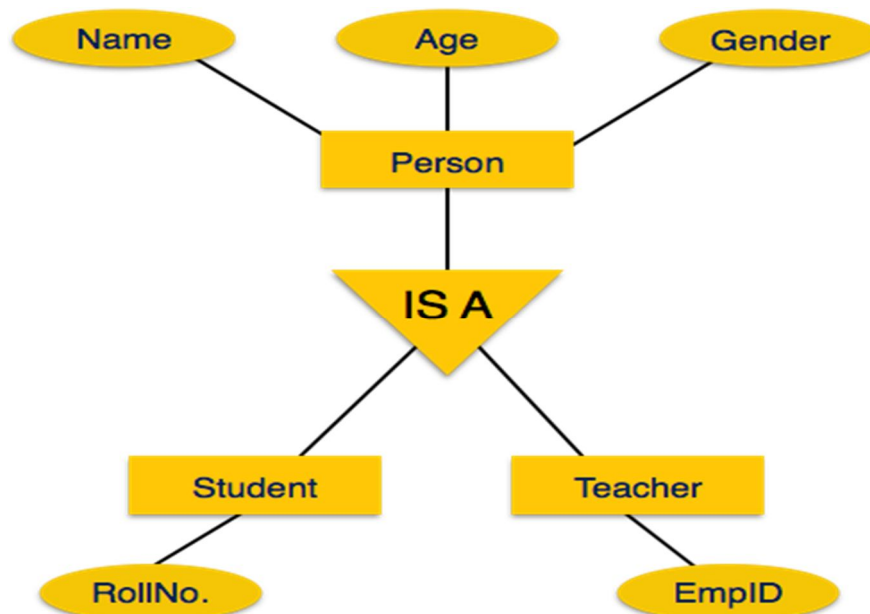
2. **Specialization:** Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company. Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic. It is a top down approach, in which one higher entity can be broken down into two lower level entity. It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.




Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.

3. **Inheritance:** We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as abstraction.

Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.



For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

4. **Sub Class and Super Class:** Sub class and Super class relationship leads the concept of Inheritance. The relationship between sub class and super class is denoted with  symbol.

➤ **Super Class:** Super class is an entity type that has a relationship with one or more subtypes. An entity cannot exist in database merely by being member of any super class.

For example: Shape super class is having sub groups as Square, Circle, Triangle.

➤ **Sub Class:** Sub class is a group of entities with unique attributes. Sub class inherits properties and attributes from its super class. For example: Square, Circle, Triangle are the sub class of Shape super class.

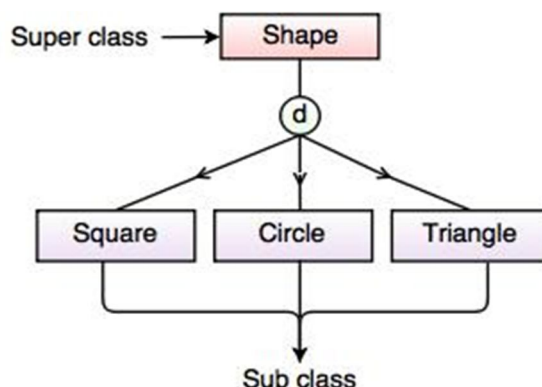


Fig. Super class/Sub class Relationship

5. **Category or Union:** Category represents a single super class or sub class relationship with more than one super class. It can be a total or partial participation. **example** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.

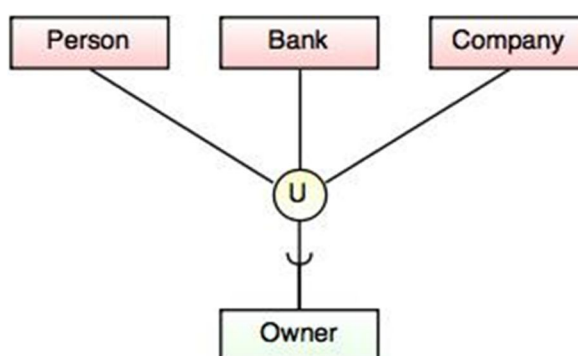


Fig. Categories (Union Type)

6: **Aggregation:** Aggregation is a process that represents a relationship between a whole object and its component parts. It abstracts a relationship between objects and viewing the relationship as an object. It is a process when two entity is treated as a single entity. In the example, the relation between College and Course is acting as an Entity in Relation with Student.

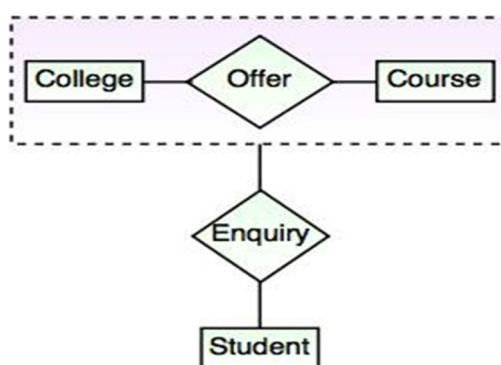
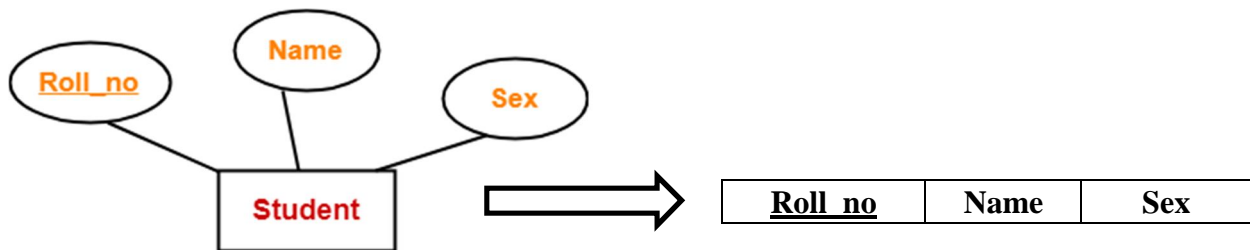


Fig. Aggregation

1.7 Converting ER Diagram to Tables:

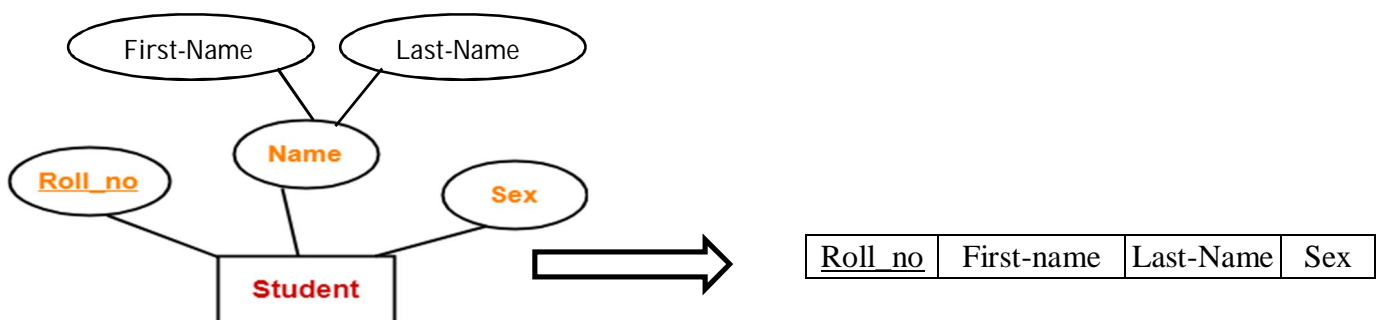
Rule-01: For Strong Entity Set With Only Simple Attributes: A strong entity set with only simple attributes will require only one table in relational model.

- Attributes of the table will be the attributes of the entity set.
- The primary key of the table will be the key attribute of the entity set.



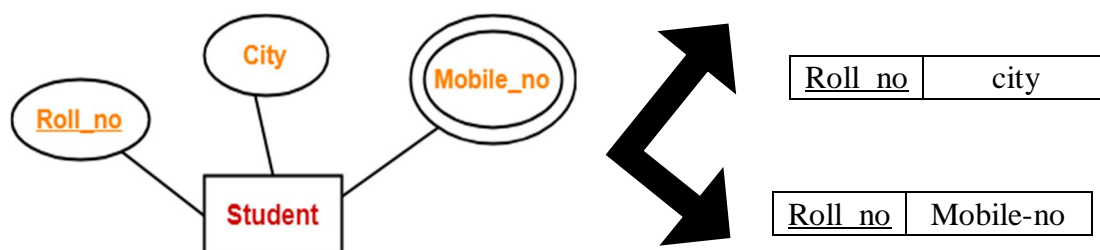
Rule-02: For Strong Entity Set With Composite Attributes:

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.



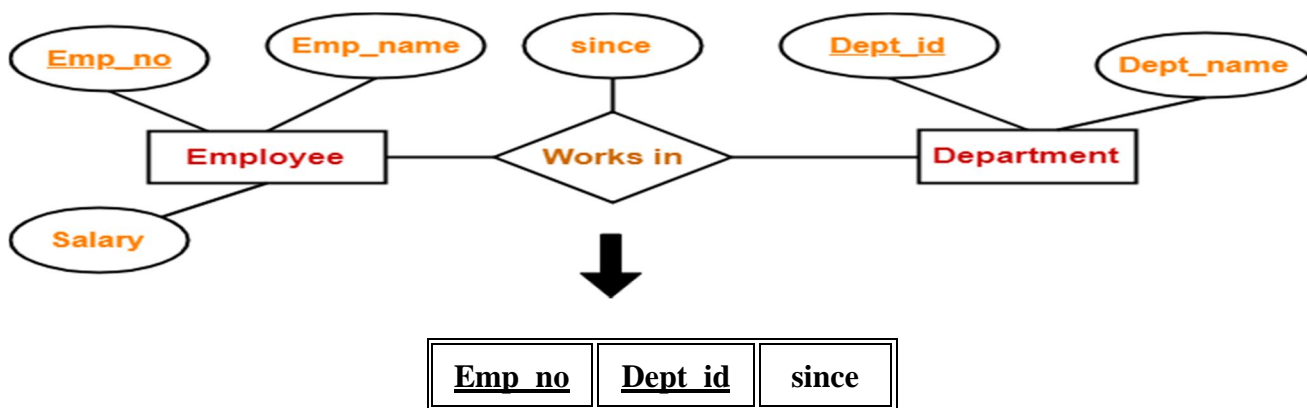
Rule-03: For Strong Entity Set With Multi Valued Attributes: A strong entity set with any number of multi valued attributes will require two tables in relational model.

- One table will contain all the simple attributes with the primary key.
- Other table will contain the primary key and all the multi valued attributes.



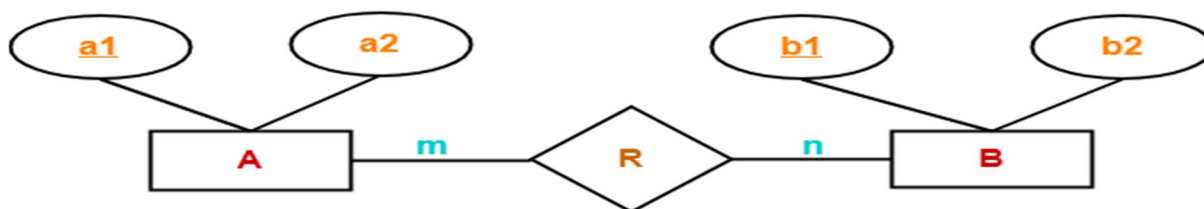
Rule-04: Translating Relationship Set into a Table- A relationship set will require one table in the relational model. Attributes of the table are

- Primary key attributes of the participating entity sets
- Its own descriptive attributes if any.
- Set of non-descriptive attributes will be the primary key.



Rule-05: For Binary Relationships With Cardinality Ratios: The following four cases are possible

Case-01: Binary relationship with cardinality ratio m:n : in this case separate table for relationship will be required which will include the attributes which describe the identity of relation entities i.e. primary key from both entities.



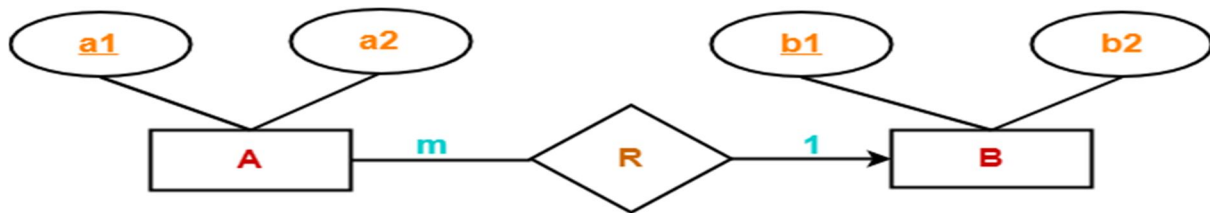
Here, three tables will be required: 1. A (a1 , a2) 2. R (a1 , b1) 3. B (b1 , b2)

Case-02: For Binary Relationship With Cardinality Ratio 1:n: in this case separate table for relationship is not required. The table for Relationship can be merged with entity having many relation. For binary relationship with cardinality ratio either m : 1 or 1 : n , always remember "many side will consume the relationship" i.e. a combined table will be drawn for many side entity set and relationship set. Here, combined table will be drawn for the entity set B and relationship set R.



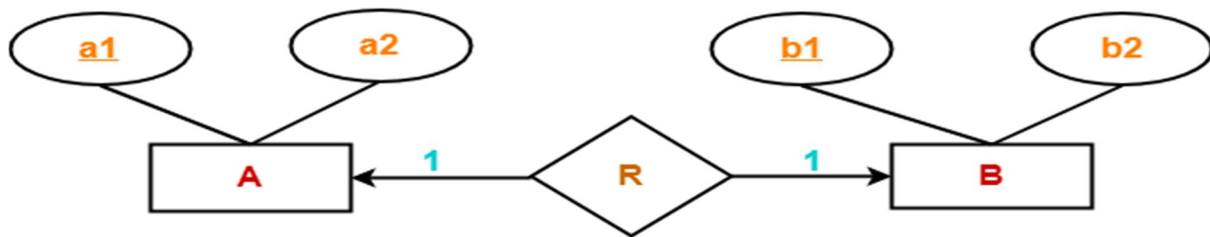
Here, two tables will be required: 1. A (a1 , a2) 2. BR (b1 , b2, a1)

Case-03: For Binary Relationship With Cardinality Ratio m:1: in this case separate table for relationship is not required. The table for Relationship can be merged with entity having many relation. For binary relationship with cardinality ratio either m : 1 or 1 : n , always remember "many side will consume the relationship" i.e. a combined table will be drawn for many side entity set and relationship set. Here, combined table will be drawn for the entity set A and relationship set R.



Here, two tables will be required: 1. AR (a1 , a2, b1) 2. B (b1 , b2)

Case-04: For Binary Relationship With Cardinality Ratio 1:1 For binary relationship with cardinality ratio 1 : 1 , two tables will be required. You can combine the relationship set with any one of the entity sets. Here, two tables will be required. Either combine 'R' with 'A' or 'B'



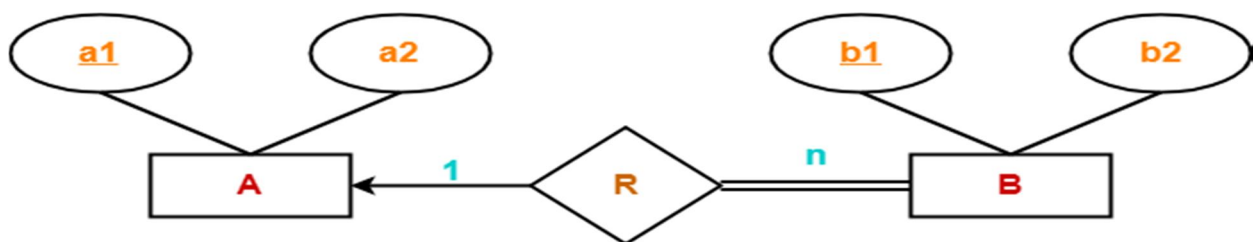
Way-01: AR (a1 , a2 , b1) & B (b1 , b2)

Way-02: A (a1 , a2) & BR (a1 , b1 , b2)

Rule-06: For Binary Relationship With Both Cardinality Constraints and Participation Constraints:

- Cardinality constraints will be implemented as discussed in Rule-05.
- Because of the total participation constraint, foreign key acquires NOT NULL constraint i.e. now foreign key cannot be null.

Case-01: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side

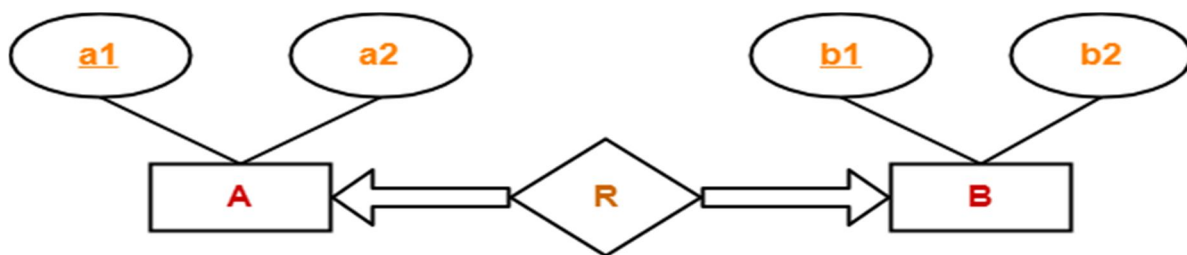


Because cardinality ratio = 1 : n , so we will combine the entity set B and relationship set R. Because of total participation, foreign key a1 has acquired NOT NULL constraint, so it can't be null now.

Then, two tables will be required: 1. AR (a1 , a2, b1) 2. B (b1 , b2)

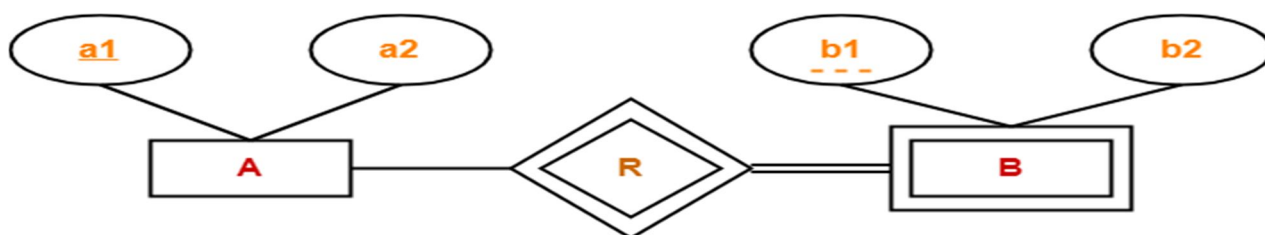
Case-02: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides:

- If there is a key constraint from both the sides of an entity set with total participation, then that binary relationship is represented using only single table.



Here, Only one table is required. 1. ARB (a1 , a2 , b1 , b2)

Rule-07: For Binary Relationship With Weak Entity Set-: Weak entity set always appears in association with identifying relationship with total participation constraint.



Here, two tables will be required: 1. A (a1 , a2) 2. BR (a1 , b1 , b2)

1.8 Database Users & architecture of DBA

1.8.1 Database Users: Database users are the one who really use and take the benefits of database. There will be different types of users depending on their need and way of accessing the database.

➤ **Application Programmers** - They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, Pascal etc. These queries are converted into object code to communicate with the database. For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C Program.

➤ **Sophisticated Users** - They are database developers, who write SQL queries to select/insert/delete/update data. They do not use any application or programs to request the database. They directly interact with the database by means of query language like SQL. These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement. In short, we can say this category includes designers and developers of DBMS and SQL.

➤ **Specialized Users** - These are also sophisticated users, but they write special database application programs. They are the developers who develop the complex programs to the requirement.

➤ **Stand-alone Users** - These users will have stand –alone database for their personal use. These kinds of database will have readymade database packages which will have menus and graphical interfaces.

➤ **Native Users** - these are the users who use the existing application to interact with the database. For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

1.8.2 Types of DBA: There are different kinds of DBA depending on the responsibility that he owns.

- **Administrative DBA** - This DBA is mainly concerned with installing, and maintaining DBMS servers. His prime tasks are installing, backups, recovery, security, replications, memory management, configurations and tuning. He is mainly responsible for all administrative tasks of a database.
- **Development DBA** - He is responsible for creating queries and procedure for the requirement. Basically his task is similar to any database developer.
- **Database Architect** - Database architect is responsible for creating and maintaining the users, roles, access rights, tables, views, constraints and indexes. He is mainly responsible for designing the structure of the database depending on the requirement. These structures will be used by developers and development DBA to code.
- **Data Warehouse DBA** -DBA should be able to maintain the data and procedures from various sources in the data warehouse. These sources can be files, COBOL, or any other programs. Here data and programs will be from different sources. A good DBA should be able to keep the performance and function levels from these sources at same pace to make the data warehouse to work.
- **Application DBA** -He acts like a bridge between the application program and the database. He makes sure all the application program is optimized to interact with the database. He ensures all the activities from installing, upgrading, and patching, maintaining, backup, recovery to executing the records works without any issues.
- **OLAP DBA** - He is responsible for installing and maintaining the database in OLAP systems. He maintains only OLAP databases.

1.8.3 Database Administrators: The life cycle of database starts from designing, implementing to administration of it. A database for any kind of requirement needs to be designed perfectly so that it should work without any issues. Once all the design is complete, it needs to be installed. Once this step is complete, users start using the database. The database grows as the data grows in the database. When the database becomes huge, its performance comes down. Also accessing the data from the database becomes challenge. There will be unused memory in database, making the memory inevitably huge. These administration and maintenance of database is taken care by database Administrator DBA. A DBA has many responsibilities. A good performing database is in the hands of DBA.

- **Installing and upgrading the DBMS Servers:** - DBA is responsible for installing a new DBMS server for the new projects. He is also responsible for upgrading these servers as there are new versions comes in the market or requirement. If there is any failure in upgradation of the existing servers, he should be able revert the new changes back to the older version, thus maintaining the DBMS working. He is also responsible for updating the service packs/ hot fixes/ patches to the DBMS servers.
- **Design and implementation:** - Designing the database and implementing is also DBA's responsibility. He should be able to decide proper memory management, file organizations, error handling, log maintenance etc for the database.
- **Performance tuning:** - Since database is huge and it will have lots of tables, data, constraints and indices, there will be variations in the performance from time to time. Also,

because of some designing issues or data growth, the database will not work as expected. It is responsibility of the DBA to tune the database performance. He is responsible to make sure all the queries and programs works in fraction of seconds.

- **Migrate database servers:** - Sometimes, users using oracle would like to shift to SQL server or Netezza. It is the responsibility of DBA to make sure that migration happens without any failure, and there is no data loss.
- **Backup and Recovery:** - Proper backup and recovery programs needs to be developed by DBA and has to be maintained him. This is one of the main responsibilities of DBA. Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.
- **Security:** - DBA is responsible for creating various database users and roles, and giving them different levels of access rights.
- **Documentation:** - DBA should be properly documenting all his activities so that if he quits or any new DBA comes in, he should be able to understand the database without any effort. He should basically maintain all his installation, backup, recovery, security methods. He should keep various reports about database performance.

1.9 Legacy System:

Software systems that are developed specially for an organisation have a long lifetime ,Many software systems that are still in use were developed many years ago using technologies that are now obsolete, These systems are still business critical that is, they are essential for the normal functioning of the business They have been given the name legacy systems. The Software system which are developed long year ago but still they are in use are called as legacy system. There is a significant business risk in simply scrapping a legacy system and replacing it with a system that has been developed using modern technology. During their lifetime they have undergone major changes which may not have been documented. Business processes are reliant on the legacy system, the system may embed business rules that are not formally documented elsewhere new software development is risky and may not be successful.

Legacy systems are not simply old software systems. Legacy systems are socio-technical computer-based systems so they include software, hardware, data and business processes. Changes to one part of the system inevitably involve further changes to other components. Decisions about these systems are not always governed by objective engineering criteria but are affected by broader organisational strategies and politics. The different logical parts of a legacy system and their relationships are illustrated in the following figure.

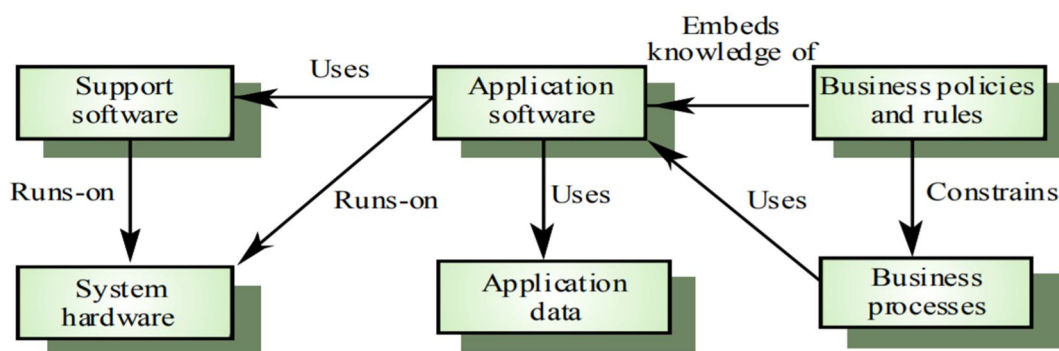


Figure: Components of Legacy System

- 1. System hardware:** In many cases, legacy systems have been written for mainframe hardware which is no longer available, which is expensive to maintain and which may not be compatible with current organisational IT purchasing policies.
- 2. Support software:** The legacy system may rely on a range of different support software from the operating system and utilities provided by the hardware manufacturer through to the compilers used for system development. Again, these may be obsolete and no longer supported by their original providers.
- 3. Application software:** As I discuss later, the application system which provides the business services is usually composed of a number of separate programs which have been developed at different times. Sometime the term legacy system means this application software system rather than the entire system.
- 4. Application data:** These are the data which are processed by the application system. In many legacy systems, an immense volume of data has accumulated over the lifetime of the system. This data may be inconsistent and may be duplicated in different files.
- 5. Business processes:** These are processes which are used in the business to achieve some business objective. An example of a business process in an insurance company would be issuing an insurance policy; in a manufacturing company, a business process would be accepting an order for products and setting up the associated manufacturing process.
- 6. Business policies and rules:** These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

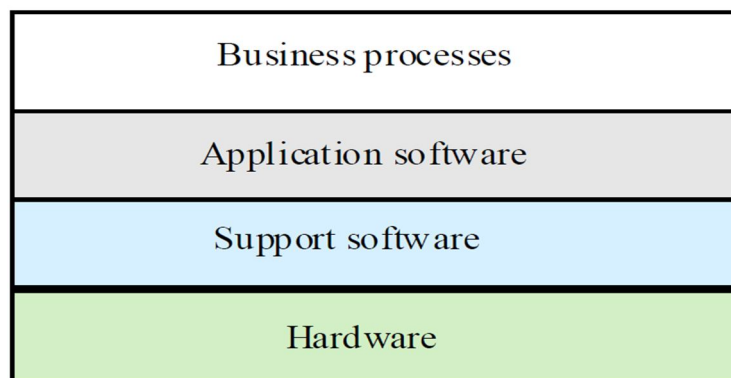


Figure: Layered model of a legacy system

An alternative way of looking at these different components of a legacy system is as series of layers. Each layer depends on the layer immediately below it and interfaces with that layer. If interfaces are maintained, then it should be possible to make changes within a layer without affecting either of the adjacent layers.

Legacy application system: The application software in a legacy system is not a single application program but usually includes a number of different programs. The system may have started as a single program processing one or two data files but, over time, changes may have been implemented by adding new programs which share the data and which communicate with other programs in the system. Similarly, the initial system data files are added to as new information is required. Different programs share data files so that changes to one program that affect data inevitably result in changes to other programs.

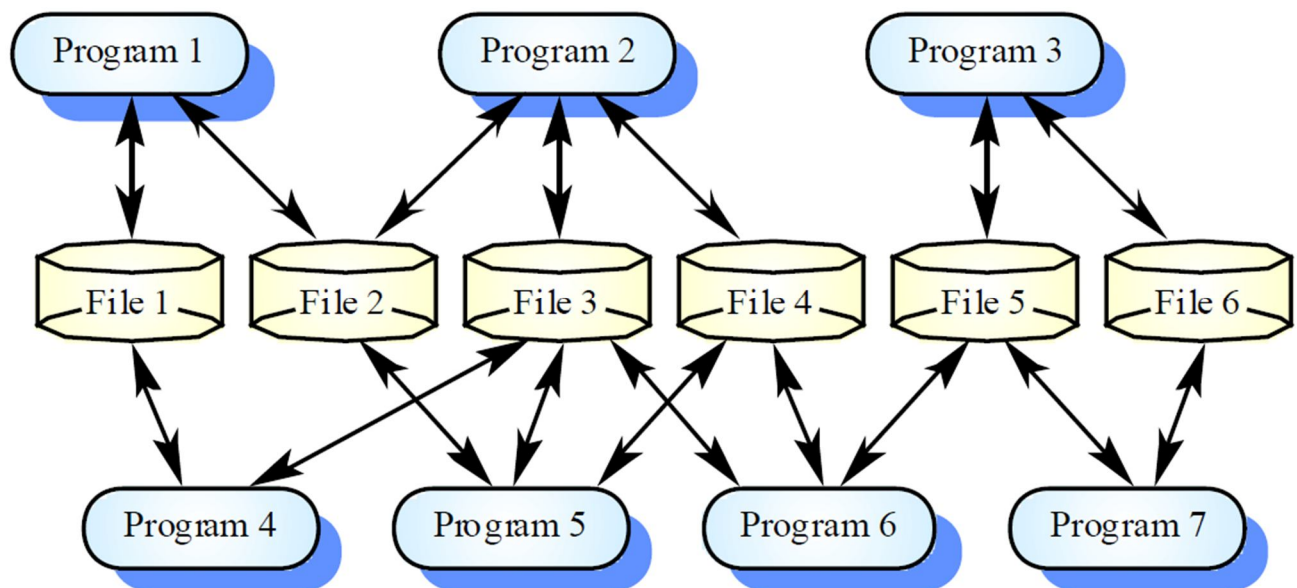


Figure Legacy application systems

The different programs in the legacy application system have usually been written by different people and are often written in different programming languages or in different versions of a programming language. For example, the original software may have been developed in COBOL-72 but later programs implemented in a new version of the language, COBOL-80. Compilers and support software for all of these languages may have to be maintained. This all adds to the complexity of the system and increases the costs of making changes to it. While there are still legacy systems that use separate files to maintain their data.

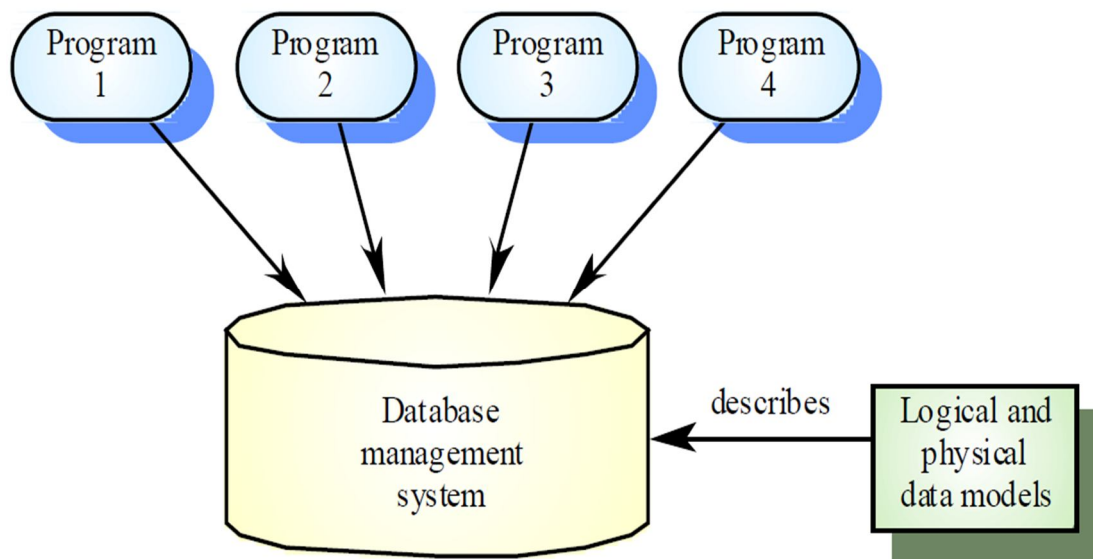


Figure: Database-centred systems