Database Languages

A DBMS is software used to build, maintain and control database systems. It allows a systematic approach to the storage and retrieval of data in a computer.
Most DBMS(s) have several major components, which include the following:

1. **Data Definition Language (DDL*)*** - These are commands used for creating, removing and altering the structure of the database.

The structures comprise of Field Names, Field sizes, Type of data for each field, File organizational technique.

2. **Data Manipulation language (DML**) - This is the user language interface and is used for executing and modifying the contents of the database. These commands allow access and manipulation of data for output. They include commands for adding, inserting, deleting, sorting, displaying etc.

3. **Data Control Language (DCL)** - These are commands used to control access to the database in response to DML commands. It acts as an interface between the DML and the OS. It provides security and control to the data.

4. **Query Languages** - A query language is a formalized method of constructing queries in database system. It provides the ways in which the user interrogates the database for data without using conventional programs. For relation database, structures query languages (SQL) has emerged as the standard language. Almost all the DBMS(s) use SQL running on machines ranging from microcomputers to large main frames.

5. **Form Generator** - A form is a screen display version of a paper form, which can be used for both input and output.

6. **Menu Generator** - This is used to generate different types of menus to suit user requirements.

7. **Report Generator** - This is a tool that gives non- specialized users the capability of providing reports from one or more files through easily constructed statements. The reports may be produced either on screen or paper. A report generator has the following features: Page headings and footings, Page Numbering, Sorting, Combining data from several files, Column headings, Totalling and subtotalling, Grouping of data etc

8. **Business Graphics** - Some DBMS may provide means of generating graphical output e.g. bar charts, pie charts scatter graphics line plots etc. others will allow users to export data into graphics software.

9. **Application Generators** - This is a type of 4th generation language used to create complete application programs.

10. **Data Dictionary (DD)** - This provides the following facilities:

    Documentation of data items

    Provision of Standard definition names for data items.

    Data item description.

    Removal of redundancy in documentation of data item.

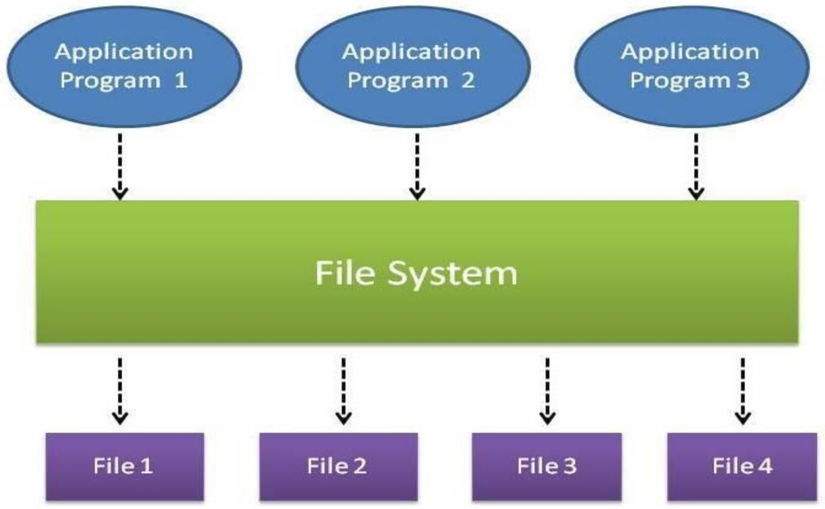
    Documentation of relationships between data items;

10. **Forth Generation Languages (4GLS'S)** - A 4GL'S is a non-procedural language in which the programs flows and not designed by the programmer but by the 4G software itself. The user requests for the result rather than a detailed procedure to obtain these results.

THE TRADITIONAL/FILE APPROACH

The term 'file-based approach' refers to the situation where data is stored in one or more separate computer files defined and managed by different application programs.

Traditional data storage model



1. In traditional approach, information is stored in flat files which are maintained by the file system under the operating system's control.
2. Application programs go through the file system in order to access these flat files

**How data is stored in flat files **

Data is stored in flat files as records.

 Records consist of various fields which are delimited by a space, comma, pipe, any special character etc.

Problems with traditional approach for storing data/

1. **Data Mapping and Access: -** Although all the related information's are grouped and stored in different files, there is no mapping between any two files. i.e.; any two dependent files are not linked. Even though Student files and Student Report files are related, they are two different files and they are not linked by any means. Hence if we need to display student details along with his report, we cannot directly pick from those two files. We have to write a lengthy program to search Student file first, get all details, then go Student_Report file and search for his report.

   When there is very huge amount of data, it is always a time consuming task to search for particular information from the file system. It is always an inefficient method to search for the data.

2. **Data Redundancy: -** There are no methods to validate the insertion of duplicate data in file system. Any user can enter any data. File system does not validate for the kind of data being entered nor does it validate for previous existence of the same data in the same file. Duplicate data in the system is not appreciated as it is a waste of space, and always lead to confusion and mishandling of data. When there are duplicate data in the file, and if we need to update or delete the record, we might end up in updating/deleting one of the record, leaving the other record in the file. Again the file system does not validate this process. Hence the purpose of storing the data is lost.

   Though the file name says Student file, there is a chance of entering staff information or his report information in the file. File system allows any information to be entered into any file. It does not isolate the data being entered from the group it belongs to.

3. **Data Dependence: -** In the files, data are stored in specific format, say tab, comma or semicolon. If the format of any of the file is changed, then the program for processing this file needs to be changed. But there would be many programs dependent on this file. We need to know in advance all the programs which are using this file and change in the entire place. Missing to change in any one place will fail whole application. Similarly, changes in storage structure, or accessing the data, affect all the places where this file is being used. We have to change it entire

programs. That is smallest change in the file affect all the programs and need changes in all them.

4. **Data inconsistency: -** Imagine Student and Student_Report files have student's address in it, and there was a change request for one particular student's address. The program searched only Student file for the address and it updated it correctly. There is another program which prints the student's report and mails it to the address mentioned in the Student_Report file. What happens to the report of a student whose address is being changed? There is a mismatch in the actual address and his report is sent to his old address. This mismatch in different copies of same data is called data inconsistency. This has occurred here, because there is no proper listing of files which has same copies of data.
5. **Data Isolation: -** Imagine we have to generate a single report of student, who is studying in particular class, his study report, his library book details, and hostel information. All these informations are stored in different files. How do we get all these details in one report? We have to write a program. But before writing the program, the programmer should find out which all files have the information needed, what is the format of each file, how to search data in each file etc. Once all these analysis is done, he writes a program. If there is 2-3 files involved, programming would be bit simple. Imagine if there is lot many files involved in it? It would be require lot of effort from the programmer. Since all the datas are isolated from each other in different files, programming becomes difficult.
6. **Security: -** Each file can be password protected. But what if have to give access to only few records in the file? For example, user has to be given access to view only their bank account information in the file. This is very difficult in the file system.
7. **Integrity: -** If we need to check for certain insertion criteria while entering the data into file it is not possible directly. We can do it writing programs. Say, if we have to restrict the students above age 18, then it is by means of program alone. There is no direct checking facility in the file system. Hence these kinds of integrity checks are not easy in file system.
8. **Atomicity: -** If there is any failure to insert, update or delete in the file system, there is no mechanism to switch back to the previous state. Imagine marks for one particular subject needs to be entered into the Report file and then total needs to be calculated. But after entering the new marks, file is closed without saving. That means, whole of the required transaction is not performed. Only the totaling of marks has been done, but addition of marks not being done. The total mark calculated is wrong in this case. **Atomicity refers to completion of whole transaction or not completing it at all. Partial completion of any transaction leads to incorrect data in the system. File system does not guarantee the atomicity**. It may be possible with complex programs, but introduce for each of transaction costs money.

9. **Concurrent Access: -** Accessing the same data from the same file is called concurrent access. In the file system, concurrent access leads to incorrect data. For example, a student wants to borrow a book from the library. He searches for the book in the library file and sees that only one copy is available. At the same time another student also, wants to borrow same book and checks that one copy available. First student opt for borrow and gets the book. But it is still not updated to zero copy in the file and the second student also opt for borrow! But there are no books available. This is the problem of concurrent access in the file system.

10. Separation and isolation of data

When data is isolated in separate files, it is more difficult for us to access data that should be available. The application programmer is required to synchronize the processing of two or more files to ensure the correct data is extracted

11. Incompatible file formats

The structures of the file are dependent on the application programming language. However file structure provided in one programming language such as direct file, indexedsequential file which is available in COBOL programming, may be different from the structure generated by other programming language such as C. The direct incompatibility makes them difficult to process jointly.

THE DATABASE/MODERN APPROACH

Because of the problems associated with the traditional approach to data management, many managers wanted a more efficient and effective means of organizing data. The result was the **database approach** to data management. In a database approach, a pool of related data is shared by multiple application programs. Rather than having separate data files, each application uses a collection of data that is either joined or related in the database.

It involves a shared collection of logically related data and its description designed to meet the information needs of an organization.

Types of Database Systems

✓ **Single User database systems** ✓ **Multi-user database** ✓ **Centralized Database system** ✓ **Distributed database system**

✓ **Transactional DBMS/Production DBMS -** This is a database system that supports immediate response transaction e.g. sale of a product.

✓ **Decision Support DBMS -** It focuses primarily on the production of information required to make a tactical or strategic decision at middle and high management levels.

Advantages of database approach

1. Control of data redundancy
The database approach attempts to eliminate the redundancy by integrating the file. Although the database approach does not eliminate redundancy entirely, it controls the amount of redundancy inherent in the database.

2. Data consistency
By eliminating or controlling redundancy, the database approach reduces the risk of inconsistencies occurring. It ensures all copies of the data are kept consistent.

3. More information from the same amount of data
With the integration of the operated data in the database approach, it may be possible to derive additional information for the same data.

4. Sharing of data
Database belongs to the entire organization and can be shared by all authorized users.

5. Improved data integrity/accurate
Database integrity provides the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate.

6. Improved security
Database approach provides a protection of the data from the unauthorized users. It may take the term of user names and passwords to identify user type and their access right in the operation including retrieval, insertion, updating and deletion.

7. Enforcement of standards
The integration of the database enforces the necessary standards including data formats, naming conventions, documentation standards, update procedures and access rules.

8. Economy of scale
Cost savings can be obtained by combining all organization's operational data into one database with applications to work on one source of data.

9. Balance of conflicting requirements
By having a structural design in the database, the conflicts between users or departments can be resolved. Decisions will be based on the base use of resources for the organization as a whole rather that for an individual entity.

10. Improved data accessibility and responsiveness
By having integration in the database approach, data accessing can be crossed departmental boundaries. This feature provides more functionality and better services to the users.

11. **Increased productivity:** The database approach provides all the low-level filehandling routines. The provision of these functions allows the programmer to concentrate more on the specific functionality required by the users. The fourthgeneration environment provided by the database can simplify the database application development.

12. Improved maintenance

Database approach provides a data independence. As a change of data structure in the database will be affect the application program, it simplifies database application maintenance.

13. Increased concurrency

Database can manage concurrent data access effectively. It ensures no interference between users that would not result any loss of information nor loss of integrity.

14. Improved backing and recovery services

Modern database management system provides facilities to minimize the amount of processing that can be lost following a failure by using the transaction approach.

Disadvantages of database approach.

In spite of a large number of advantages can be found in the database approach, it is not without any challenge. The following disadvantages can be found including:

1. Complexity

Database management system is an extremely complex piece of software. All parties must be familiar with its functionality and take full advantage of it. Therefore, training for the administrators, designers and users is required.

2. **Size:** The database management system consumes a substantial amount of main memory as well as a large number amount of disk space in order to make it run efficiently.

3. Cost of DBMS

A multi-user database management system may be very expensive. Even after the installation, there is a high recurrent annual maintenance cost on the software.

4. Cost of conversion

When moving from a file-base system to a database system, the company is required to have additional expenses on hardware acquisition and training cost.

5. Performance

As the database approach is to cater for many applications rather than exclusively for a particular one, some applications may not run as fast as before.

6. Higher impact of a failure

The database approach increases the vulnerability of the system due to the centralization. As all users and applications reply on the database availability, the failure of any component can bring operations to a halt and affect the services to the customer seriously

instance of a database Is : at any particular point in time, what is the state of database with data values in its object is called database instance. It changes from time to time.

Traditional File Approach	**Database Approach**
Use separate data file for each application	All Application shares a pool of related and integrated data.
Data redundancy – independent data files included a lot of duplicated data.	Minimal data redundancy – Separate data files are integrated in to a single, logical structure.
Same data is recorded and stored in several files.	Each occurrence of a data item is recorded only once.
Data inconsistency – several versions of the same data may exist.	Single version of data exist
Same update must be done in all occurrences of same data item in each file.	Single update is required.
Users have very little opportunity to share data outside of their own application.	A database is developed to share the data among the user who access to it
There is no centralized control for overall data in different files.	There is centralized control for overall data in database.
Data dependence – description of files, records and data items are embedded within individual application programs.	Data independence – the database system separates data descriptions from the application programs that use the data in it
Modification to data files requires the programs which access that file to be modified.	Data structure can be modified without changing the programs accessing the data
High program maintenance	Less program maintenance
Lack of data integration – accessing data in several files are difficult	Data are organized in to a single logical structure with logical relationships defined between associated data
Difficult to manipulation data	Easy to manipulation data

Classification of Database Management Systems

Database management systems can be classified based on several criteria, such as the data model, user numbers and database distribution, all described below.

Classification Based on Data Model

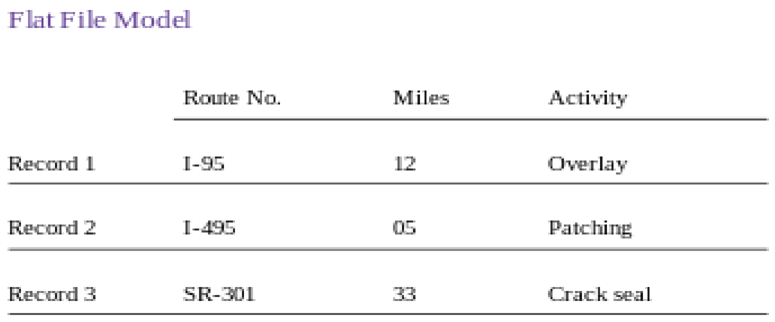
The most popular data model in use today is the relational data model. Well-known DBMSs like Oracle, MS SQL Server, DB2 and MySQL support this model. Other traditional models, such as hierarchical data models and network data models, are still used in industry mainly on mainframe platforms.

Types of data models

A **database model** is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model, which uses a table-based format. A database model is a specification describing how a database is structured and used.

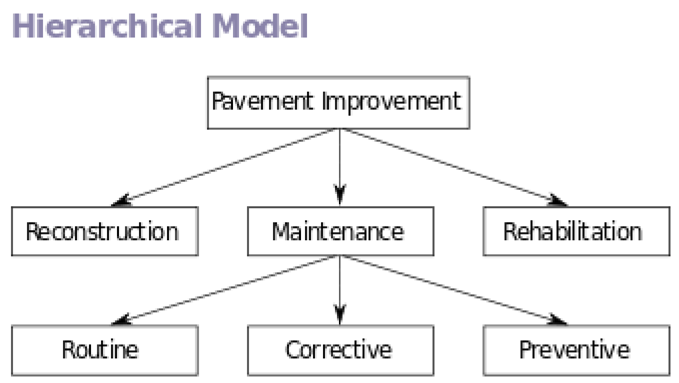
Flat model

This may not strictly qualify as a data model. The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another.


Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

Hierarchical model: In this model data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list.



<u>Advantages</u>

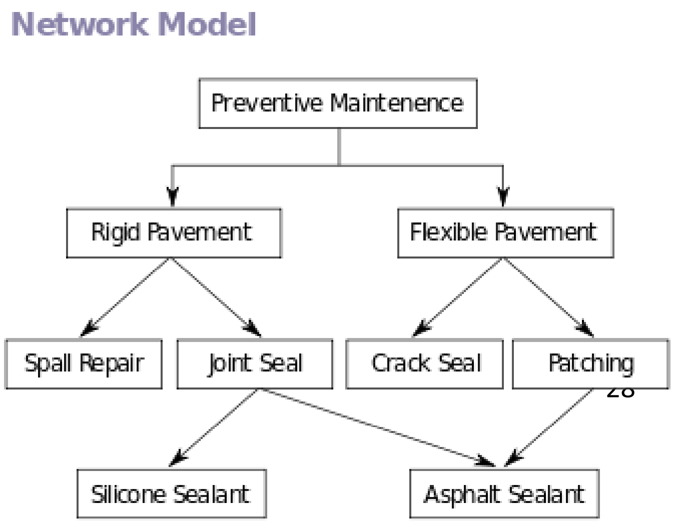
- ✓ Simplicity
- ✓ Data Security and Data Integrity
- ✓ Efficiency

<u>Disadvantages</u>

- ✓ Implementation Complexity
- ✓ Lack of structural independence
- ✓ Programming complexity

<u>Network model</u>

This model organizes data using two fundamental constructs, called records and sets. Records contain fields, and sets define one-to-many relationships between records: one owner, many members.



Advantages

- ✓ Conceptual Simplicity
- ✓ Ease of data access
- ✓ Data Integrity and capability to handle more relationship types
- ✓ Data independence
- ✓ Database standards

Disadvantages

- ✓ System complexity
- ✓ Absence of structural independence

Relational model

It is a database model based on first-order predicate logic. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. **Properties of Relational Tables:**

- ✓ Values Are Atomic
- ✓ Each Row is Unique
- ✓ Column Values Are of the Same Kind
- ✓ The Sequence of Columns is Insignificant
- ✓ The Sequence of Rows is Insignificant
- ✓ Each Column Has a Unique Name

Advantages

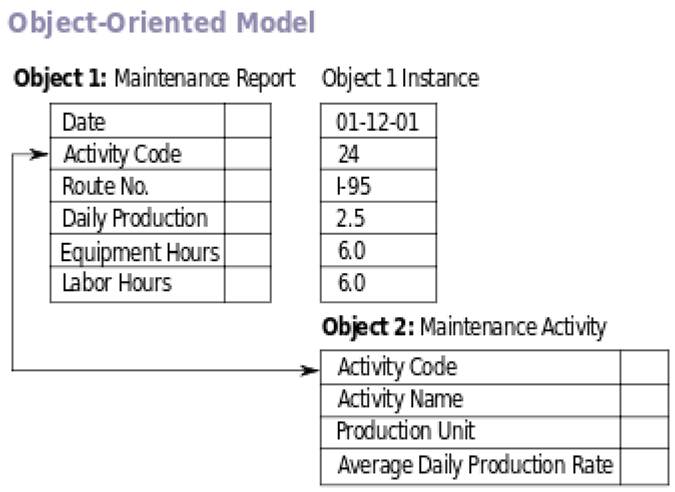
- ✓ Structural Independence
- ✓ Conceptual Simplicity
- ✓ Ease of design, implementation, maintenance and usage.
- ✓ Ad hoc query capability

Disadvantages

- ✓ Hardware Overheads
- ✓ Ease of design can lead to bad design

Object-relational model

Similar to a relational database model, but objects, classes and inheritance are directly supported in database schemas and in the query language.



Advantages

- ✓ Capability to handle large number of different data types
- ✓ Marriage of object-oriented programming and database technology ✓ Data access is easier

Disadvantages

- ✓ Difficult to maintain
- ✓ Not suited for all applications **Classification**

Based on User Numbers

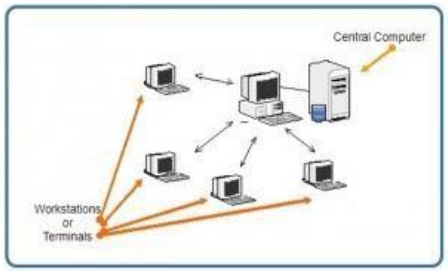
A DBMS can be classification based on the number of users it supports. It can be a *singleuser database system*, which supports one user at a time, or a *multiuser database system*, which supports multiple users concurrently.

Classification Based on Database Distribution

There are four main distribution systems for database systems and these, in turn, can be used to classify the DBMS.

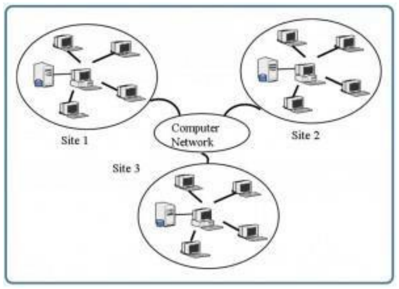
Centralized systems

With a *centralized database system*, the DBMS and database are stored at a single site that is used by several other systems too.



Distributed database system

In a *distributed database system*, the actual database and the DBMS software are distributed from various sites that are connected by a computer network,



Homogeneous distributed database systems

Homogeneous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily. For example, library information systems by the same vendor, such as Geac Computer Corporation, use the same DBMS software which allows easy data exchange between the various Geac library sites.

Heterogeneous distributed database systems

In a *heterogeneous distributed database system*, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites. For example, the various library database systems use the same machine-readable cataloguing (MARC) format to support library record data exchange.

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

Hierarchical database: A hierarchical database organizes information in a tree-like structure in which data elements are related to each other in a parent (superior) to child (subordinate) relationship. A data element can be a data field, a record, or a database file. The hierarchical database provides a one-to-many relationship, in a top-down manner. To access the employee of any department, you must specify the department because department is the parent of employee. If you have no information on the parent, it is impossible to retrieve the item because you must access the item through its parent. A hierarchical structure is particularly useful for databases containing structured information where access to information is keyed to the structure, that is, the logical access is in the same hierarchy as the physical layout of the database. The rigid structure of a hierarchical database enables it to be updated efficiently. Typically, it is used in applications such as inventory management, where a large number (hundreds of thousands or millions) of records are in the system

Network database: A network database is similar to a hierarchical database except that a child in the system can have more than one parent. Thus, because more than one path to a particular data element exists, the database structure is many-to-many. Network databases are particularly efficient for looking up information because they permit access from more than one starting point. Unlike a hierarchical database, the process of querying a network database is less restrictive. A network database is appropriate for situations where queries of the database may not follow a predetermined pattern. An example is a database of students and their course enrolment, where a student can be enrolled in multiple courses. The relationship between students and courses is thus many-to many, and a hierarchical

MODEL	ADVANTAGES	DISADVANTAGES
Hierarchical Data Structure	Ease with which data can be stored and retrieved in structured, routine types of transactions. Ease with which data can be extracted for reporting purposes. Routine types of transaction processing is fast and efficiently.	Hierarchical one-to many relationships must be specified in advance, and are not flexible. Cannot easily handle ad hoc requests for information. Modifying a hierarchical database structure is complex. Great deal of redundancy. Requires knowledge of a programming language.

Network Structure	More flexible that the hierarchical model. Ability to provide sophisticated logical relationships among the records	Network many-to-many relationships must be specified in advance User is limited to retrieving data that can be accessed using the established links between records.
		Cannot easily handle ad hoc requests for information. Requires knowledge of a programming language.
Relational Structure	Flexible in that it can handle ad hoc information requests. Easy for programmers to work with. End users can use this model with litter effort or training. Easier to maintain than the hierarchical and network models.	Cannot process large amounts of business transactions as quickly and efficiently as the hierarchical and network models.

database is inappropriate.

DATA ORGANIZATION

DISTRIBUTED DATABASE APPROACH

Data organization refers to the method of classifying and organizing data sets to make them more useful.

A single logical database that is spread physically across computers in multiple locations that are connected by a data communications link.

OR

A distributed database system is a collection of logically related database that co-operate in a transparent manner.

In a distributed database scheme, smaller pieces of the overall database are stored at various physical locations. Each piece is controlled by a local database server that satisfies requests for data valid over its local domain only. All local data servers are connected by a common network so that any application connected to this network has access to every server and its local data.

Functions of Distributed Database:

Keeping Track of Data: The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.

Distributed Query Processing: The ability to access remote sites and transmit queries and data among the various sites via a communication network.

Distributed transaction management: The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data maintain integrity of the overall database.

Distributed Data Recovery: The ability to recover from individual site crashes and from new type of failures such as the failure of a communication links.

Security: Distributed transactions must be executed with the proper management, security of the data and the authorization/access privileges of users**.**

Distributed Catalog Management: A catalog contains information about data in the database. The catalog may be global for the entire Distributed database or local for each site. The placement and distribution of the catalog are design and policy issues.

ADVANTAGES

Increased Reliability and Availability: when a centralized system fails, the database is unavailable to all users. A distributed system will continue to function at some reduced level, however even when a component fails. The reliability and availability will depend on how the data are distributed.

Lower Communication costs: With a distributed system data can be located closer to their point of use. This can reduce communication costs, compared to a central system.

Faster Response: Depending on how data are distributed, most requests for data by user at a particular site can be satisfied by data stored at that site. This speed up query processing since communication and central computer delays are minimized.
Local Control: The database is brought nearer to its user. This can effect a cultural change as it allows potentially greater control over local data.

Easier Expansion: In a distributed environment, expansion of the system in terms of adding more data, increasing database size, or adding more processors is much easier.

DISADVANTAGES

Data Integrity: A by-product of the increased complexity and need for coordination is the additional exposure to improper updating and other problems of data integrity.

Slow Response: If the data are not distributed properly according to their usage, or if queries are not formulated correctly, response to request for data can be extremely slow.

Processing Overhead: The various sites must exchange messages and perform additional calculations to ensure proper coordination among data at the different sites.

Architectural Complexity.

Lack of Standards.

Lack of Experience.

Database design more complex.

Factors for adoption Distributed Database Approach:

When the data is divided into different sites distributed database is suitable.

Distributed Database is suitable for accessing data and transmits queries among different sites.

Distributed database is suitable for proper management and security of the data and the authorized/access privileges of user.

For achieving local control over data for each user/site Distributed database is suitable.

CENTRALIZED DATABASE APPROACH

A centralized database consists of a single data server into which all data are stored and from which all data are retrieved. All the data reside at a single location and all applications must retrieve all data from that location.

The centralized approach consists of a central server into which all forecast data are stored. At some predefined time, software on this central server requests data from each of the local data servers scattered throughout the country. These data are received, processed and stored, possibly at lower spatial and temporal resolution than the data from which it was derived.

Advantages of Centralized Approach

Decreased Risk: With Centralized data management, all edits and manipulation to core data are housed and stored centrally. This model allows for staunch controls, detailed audit trails, and enables business users to access consistent data.
Data Consistency: When data feeds are managed in a central repository, an organization can achieve consistent data management and distribution throughout its global offices and internal systems.

Data Quality:

Operational Efficiency: When one business unit controls an organization data centrally, the resources previously devoted to data management can be redirected back to core business needs.

Single Point of Entry: By introducing single point of entry for data, this allows changes from data vendors to be implemented once, rather than in multiple instances.

Cost Saving: With data management centralized, costs attributed to vendor relationships are better controlled, minimizing any redundancy in market data contracts and their associated costs.

DISADVANTAGES

Response Time: The size of a centralized database could cause data retrieval delays.

Equipment Cost: Depend on architecture, probably expensive if main frames are used.

Incremental Growth: Updating a centralized system is likely to be more difficult and more costly particularly if is it build on main frames. Large systems are typically more problematic to upgrade.

Single Point of Failure: Entire data is sorted at a single point (central server), if the data failed or corrupted then all the data will be lose.

Factors for Adoption of Centralized Approach:

Data can be organized in single point, by introducing single point of entry for data Database Administrator can implement the data only once instead of in multiple sites.

Data consistency can achieve by introducing data-centric approach.

Centralized database approach is suitable for establishment of data standards across an enterprise.

For batter security purpose Centralized database approach is suitable.

For quick efficient searching Centralized approach is good one.

For controlled access to the database repository.

CLIENT/SERVER DATABASE APPROACH

The **client–server model** is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

-Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system.
-A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function.

-Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect.

-A client is any component of a system that request services or resources from the other systems components.

-A server is any system component that provides services or resources to other systems components.

ADVANTAGES OF CLIENT/SERVER DATABASE APPROACH

1) **Centralization :** In this architecture there is a centralized control
2) **Proper Management :** All the files are stored at the same place. In this way, management of files becomes easy. Also it becomes easier to find files.

3) **Back-up and Recovery possible :** As all the data is stored on server its easy to make a back-up of it. Also, in case of some break-down if data is lost, it can be recovered easily and efficiently.

4) **Upgradation and Scalability in Client-server set-up :** Changes can be made easily by just upgrading the server. Also new resources and systems can be added by making necessary changes in server.

5) **Accessibility:** From various platforms in the network, server can be accessed remotely.

7) **Security:** Rules defining security and access rights can be defined at the time of set-up of server.

8) Servers can play different roles for different clients.

Disadvantages of Client Server

1) **Congestion in Network:** Too many requests from the clients may lead to congestion, which rarely takes place in P2P network. Overload can lead to breakingdown of servers. In peer-to-peer, the total bandwidth of the network increases as the number of peer's increase.
2) Client-Server architecture is **not as robust** and if the server fails, the whole network goes down. Also, if you are downloading a file from server and it gets abandoned due to some error, download stops altogether. However, if there would have been peers, they would have provided the broken parts of file.
3) Cost : It is very expensive to install and manage this type of computing. **4)** You **need professional IT people** to maintain the servers and other technical details of network.

In a client server model, the database server is responsible of the following:

✓ Enforcing data security

✓ Processing data modification and retrieval

✓ Performing data processing

The DB client is responsible of:

✓ Submitting of requests to the server

✓ Presenting the data to the user in an easily useful format.

Advantages

1. Increased performance – if the client and server resides on different computers, parallel processing of applications is possible

2. H/W cost may be reduced – the server needs to have powerful storage. These leads to purchase of few H/W

3. Security – Use of server enforces data security of data security of data in the DB.

PRINCIPLES AND TECHNIQUES OF DATABASE DESIGN

Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

The term database design can be used to describe many different parts of the design of an overall database system. The process of doing database design generally consists of a number of steps which will be carried out by the database designer. Usually, the designer must:

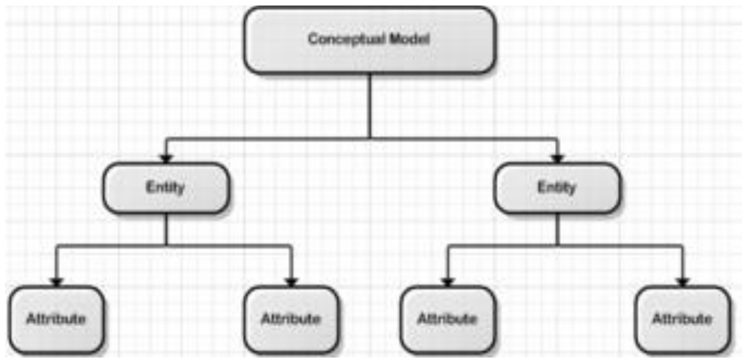
Database designs also include ER (entity-relationship model) diagrams. An ER diagram is a diagram that helps to design databases in an efficient way.

Database Design Strategies

There are two approaches for developing any database, the top-down method and the bottom-up method. While these approaches appear radically different, they share the common goal of uniting a system by describing all of the interaction between the processes.

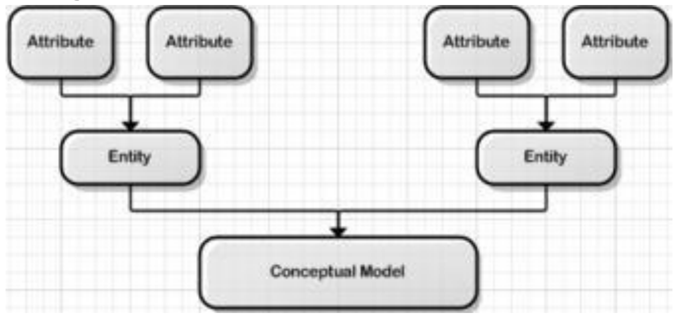
Top – down design method

The top-down design method starts from the general and moves to the specific. In other words, you start with a general idea of what is needed for the system and then work your way down to the more specific details of how the system will interact. This process involves the identification of different entity types and the definition of each entity's attributes.



Bottom – up design method

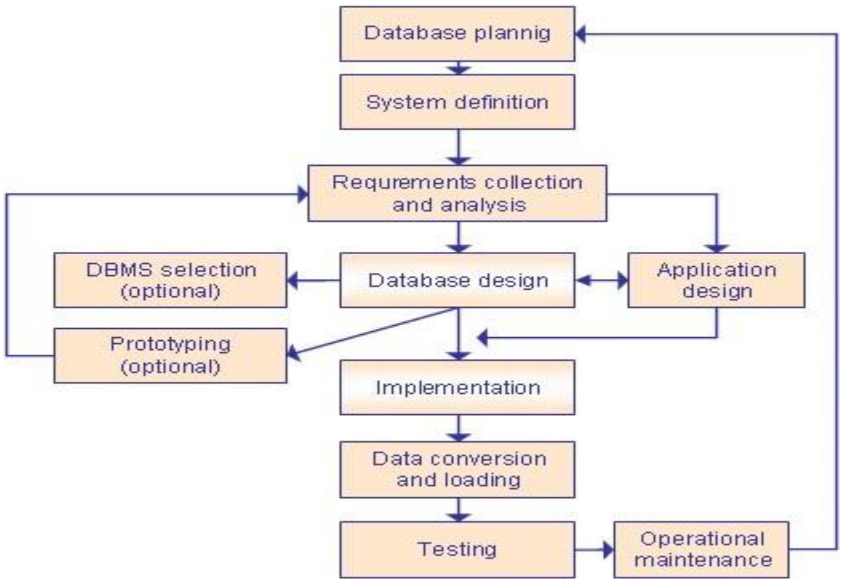
The bottom-up approach begins with the specific details and moves up to the general. This is done by first identifying the data elements (items) and then grouping them together in data sets. In other words, this method first identifies the attributes, and then groups them to form entities.



Two general approaches (top – down and bottom – up) to the design of the databases can be heavily influenced by factors like scope, size of the system, the organizations management style, and the organizations structure. Depending on such factors, the design of the database might use two very different approaches, centralized design and decentralized design.

Database management system lifecycle A database is usually a fundamental component of the information system, especially in business oriented systems. Thus database design is part of system development. The following picture shows how database design is involved in the system development lifecycle.

The phases in the middle of the picture (Database Design, Database Implementation) are the phases that you concentrate on in the Database Design course. The other phases are briefly described. They are part of the contents of the Systems Analysis and Design courses, for example.
There are various methods of how the different phases of information system design, analysis and implementation can be done. Here the main tasks or goals are described but no method is introduced.



Database Planning

The database planning includes the activities that allow the stages of the database system development lifecycle to be realized as efficiently and effectively as possible. This phase must be integrated with the overall Information System strategy of the organization.

The very first step in database planning is to define the mission statement and objectives for the database system. That is the definition of:
- the major aims of the database system
- the purpose of the database system
- the supported tasks of the database system
- the resources of the database system

Systems Definition

In the systems definition phase, the scope and boundaries of the database application are described. This description includes:
- links with the other information systems of the organization
- what the planned system is going to do now and in the future
- who the users are now and in the future.

The major user views are also described. i.e. what is required of a database system from the perspectives of particular job roles or enterprise application areas.

Requirements Collection and Analysis

During the requirements collection and analysis phase, the collection and analysis of the information about the part of the enterprise to be served by the database are completed. The results may include eg:
- the description of the data used or generated
- the details how the data is to be used or generated
- any additional requirements for the new database system

Database Design

The database design phase is divided into three steps:
- conceptual database design
- logical database design
- physical database design

In the conceptual database design phase, the model of the data to be used independent of all physical considerations is to be constructed. The model is based on the requirements specification of the system.

In the logical database design phase, the model of the data to be used is based on a specific data model, but independent of a particular database management system is constructed. This is based on the target data model for the database e.g. relational data model.

In the physical database design phase, the description of the implementation of the database on secondary storage is created. The base relations, indexes, integrity constraints, security, etc. are defined using the SQL language.

Database Management System Selection

This in an optional phase. When there is a need for a new database management system (DBMS), this phase is done. DBMS means a database system like Access, SQL Server, MySQL, Oracle.

In this phase the criteria for the new DBMS are defined. Then several products are evaluated according to the criteria. Finally the recommendation for the selection is decided.

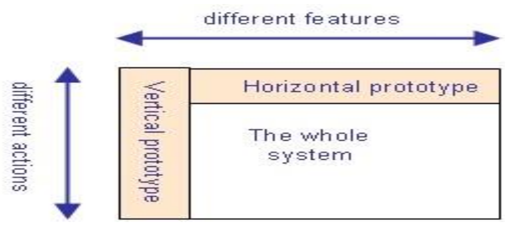
Application Design

In the application design phase, the design of the user interface and the application programs that use and process the database are defined and designed.

Protyping

The purpose of a prototype is to allow the users to use the prototype to identify the features of the system using the computer.

There are horizontal and vertical prototypes. A horizontal prototype has many features (e.g. user interfaces) but they are not working. A vertical prototype has very few features but they are working. See the following picture.



Implementation

During the implementation phase, the physical realization of the database and application designs are to be done. This is the programming phase of the systems development.

Data Conversion and Loading

This phase is needed when a new database is replacing an old system. During this phase the existing data will be transferred into the new database.

Testing

Before the new system is going to live, it should be thoroughly tested. The goal of testing is to find errors! The goal is not to prove the software is working well.

Operational Maintenance

The operational maintenance is the process of monitoring and maintaining the database system.

-Monitoring means that the performance of the system is observed. If the performance of the system falls below an acceptable level, tuning or reorganization of the database may be required.

-Maintaining and upgrading the database system means that, when new requirements arise, the new development lifecycle will be done.

RELATIONAL DATABASE SYSTEM

A relational database (RDB) is a collective set of multiple data sets organized by tables, records and columns. RDBs establish a well-defined relationship between database tables. Tables communicate and share information, which facilitates data searchability, organization and reporting.

Key : A key is a single or combination of multiple fields. Its purpose is to access or retrieve data rows from table according to the requirement. The keys are defined in tables to access or sequence the stored data quickly and smoothly. They are also used to create links between different tables.

Types of Keys The following tables or relations will be used to define different types of keys.

Primary Key The attribute or combination of attributes that uniquely identifies a row or record in a relation is known as primary key.

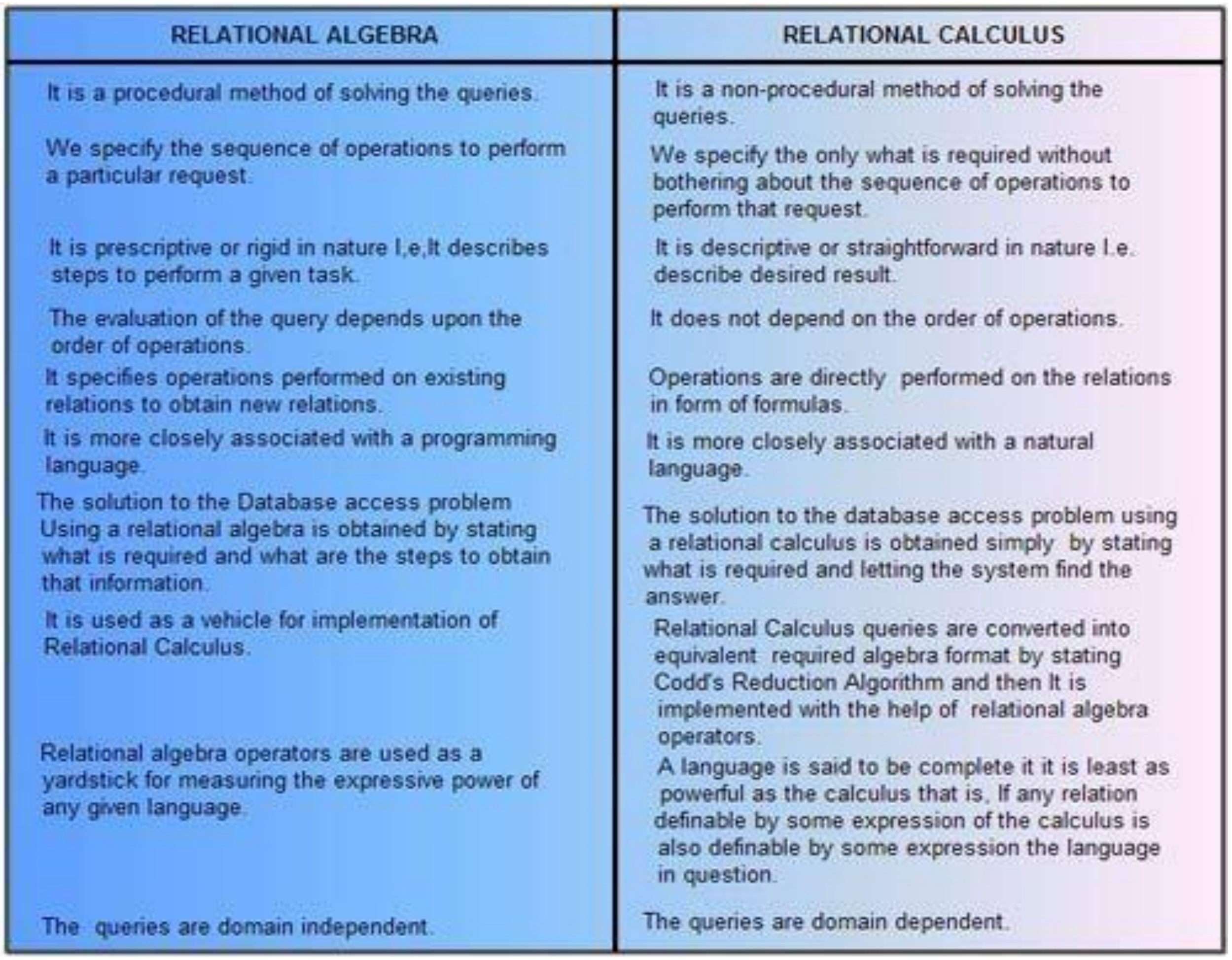
Secondary key A field or combination of fields that is basis for retrieval is known as secondary key. Secondary key is a non-unique field. One secondary key value may refer to many records.

Candidate Key or Alternate key A relation can have only one primary key. It may contain many fields or combination of fields that can be used as primary key. One field or combination of fields is used as primary key. The fields or combination of fields that are not used as primary key are known as candidate key or alternate key.-

-**Composite key or concatenate key** A primary key that consists of two or more attributes is known as composite key.

Sort Or control key A field or combination of fields that is used to physically sequence the stored data called sort key. It is also known s control key.

A **superkey** is a combination of attributes that can be uniquely used to identify a database record. A table might have many superkeys. Candidate keys are a special subset of superkeys that do not have any extraneous information in them.



RELATIONAL ALGEBRA	RELATIONAL CALCULUS
It is a procedural method of solving the queries.	It is a non-procedural method of solving the queries.
We specify the sequence of operations to perform a particular request.	We specify the only what is required without bothering about the sequence of operations to perform that request.
It is prescriptive or rigid in nature I,e,It describes steps to perform a given task.	It is descriptive or straightforward in nature I.e. describe desired result.
The evaluation of the query depends upon the order of operations.	It does not depend on the order of operations.
It specifies operations performed on existing relations to obtain new relations.	Operations are directly performed on the relations in form of formulas.
It is more closely associated with a programming language.	It is more closely associated with a natural language.
The solution to the Database access problem Using a relational algebra is obtained by stating what is required and what are the steps to obtain that information.	The solution to the database access problem using a relational calculus is obtained simply by stating what is required and letting the system find the answer.
It is used as a vehicle for implementation of Relational Calculus.	Relational Calculus queries are converted into equivalent required algebra format by stating Codd's Reduction Algorithm and then It is implemented with the help of relational algebra operators.
Relational algebra operators are used as a yardstick for measuring the expressive power of any given language.	A language is said to be complete it it is least as powerful as the calculus that is, If any relation definable by some expression of the calculus is also definable by some expression the language in question.
The queries are domain independent.	The queries are domain dependent.

Notation

The operations have their own symbols. The symbols are hard to write in HTML that works with all browsers, so I'm writing **PROJECT** etc here. The real symbols:

Operation	My HTML	Symbol
Projection	**PROJECT**	π
Selection	**SELECT**	σ

Operation	My HTML	Symbol
Cartesian product	**X**	$\times$

Renaming	**RENAME**	ρ
Union	**UNION**	$\cup$
Intersection	**INTERSECTION**	$\cap$
Assignment	**<-**	$\leftarrow$

Join	**JOIN**	$\bowtie$
Left outer join	**LEFT OUTER JOIN**	⟕
Right outer join	**RIGHT OUTER JOIN**	⟖
Full outer join	**FULL OUTER JOIN**	⟗
Semijoin	**SEMIJOIN**	$\ltimes$

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

- Select
- Project
- Union
- Set different
- Cartesian product

- Rename

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like – =, ≠, ≥, < , >, ≤.

For example –

$\sigma_{subject = "database"}(Books)$

Output – Selects tuples from books where subject is 'database'.

$\sigma_{\text{subject = "database" and price = "450"}}(\text{Books})$

Output – Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject = "database" and price = "450" or year > "2010"}}(\text{Books})$

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation – $\prod_{A1, A2, An} (r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example –

$\prod_{\text{subject, author}} (\text{Books})$

Selects and projects columns named as subject and author from the relation Books.

SET OPERATIONS

Union Operation (∪)

It performs binary union between two given relations and is defined as

Notation – r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
Notation − r − s

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as

− r X s = { q t | q ∈ r and t ∈ s} $\sigma_{author = 'tutorialspoint'}$(Books X

Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − ρ_{x} (E)

Where the result of expression **E** is saved with name of **x**.

Additional operations are –

- Set intersection
- Assignment
- Natural join

JOIN OPERATION

- **Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.
- We will briefly describe various join types in the following sections.

Theta (θ) Join

- Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation

- $R1 \bowtie_{\theta} R2$
- R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is R1 ∩ R2 = Φ. Theta join can use all kinds of comparison operators.

Student

SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects

Class	Subject
10	Math
10	English
11	Music
11	Sports

Student_Detail –

- STUDENT $\bowtie_{Student.Std = Subject.Class}$ SUBJECT

Student_detail

SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

- When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.
- Natural Join (⋈)
- Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses

CID	Course	Dept
CS01	Database	CS

ME01	Mechanics	ME
EE01	Electronics	EE

HoD

Dept	Head
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD

Dept	CID	Course	Head
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

- Outer Joins
- Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.
- Left Outer Join(R ⟕ S)

Left

A	B
100	Database
101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex
102	Maya
104	Mira

Courses ⟕HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

- Right Outer Join: (R ⟖S)
- All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

- All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the Sattributes of the resulting relation are made NULL.

100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

- Full Outer Join: (R ⟗S)
- All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched

Courses HoD			
A	**B**	**C**	**D**

attributes are made NULL.

Courses ⟗HoD			
A	**B**	**C**	**D**
100	Database	100	Alex

101	Mechanics --- --- 102 Electronics 102 Maya		
---	---	104	Mira

ENTITY RELATIONSHIP DIAGRAM/ MODELLING

<u>ENTITY RELATIONSHIP DIAGRAM</u>

An **entity relationship model**, also called an **entity-relationship** (ER) **diagram**, is a graphical representation of **entities** and their **relationships** to each other, typically used in computing in regard to the organization of data within databases or information systems.

Entity

A definable thing—such as a person, object, concept or event—that can have data stored about it. Think of entities as nouns. Examples: a customer, student, car or product. Typically shown as a rectangle.

Entity type: A group of definable things, such as students or athletes, whereas the entity would be the specific student or athlete. Other examples: customers, cars or products.

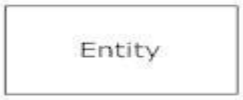
Entity set: Same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.

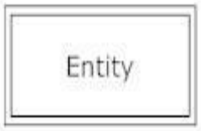
Entity categories: Entities are categorized as strong, weak or associative. A **strong entity** can be defined solely by its own attributes, while a **weak entity** cannot. An associative entity associates entities (or elements) within an entity set.

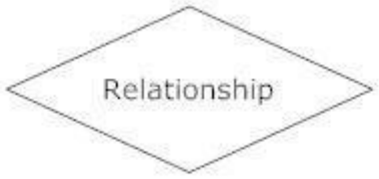
Entity keys: Refers to an attribute that uniquely defines an entity in an entity set. Entity keys can be super, candidate or primary. **Super key:** A set of attributes (one or more) that together define an entity in an entity set. **Candidate key:** A minimal super key, meaning it has the least possible number of attributes to still be a super key. An entity set may have more than one candidate key. **Primary key:** A candidate key chosen by the database designer to uniquely identify the entity set. **Foreign key:** Identifies the relationship between entities.

<u>Structure of an Entity Relationship Diagram with Common ERD Notations</u>

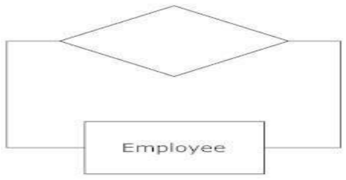
An entity relationship diagram is a means of visualizing how the information a system produces is related. There are five main components of an ERD:

Entities, which are represented by rectangles. An entity is an object or concept about which you want to store information. 

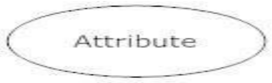
A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone. 

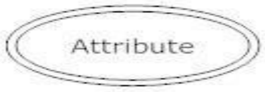
Actions, which are represented by diamond shapes, show how two entities share information in the database. 

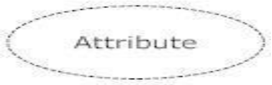
In some cases, entities can be self-linked. For example, employees can supervise other employees.



Attributes, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.

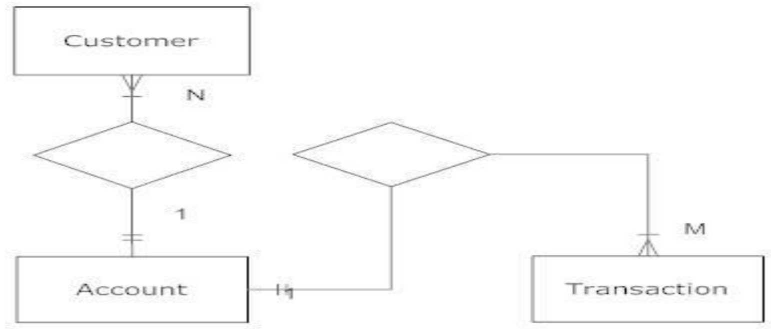


A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values. 

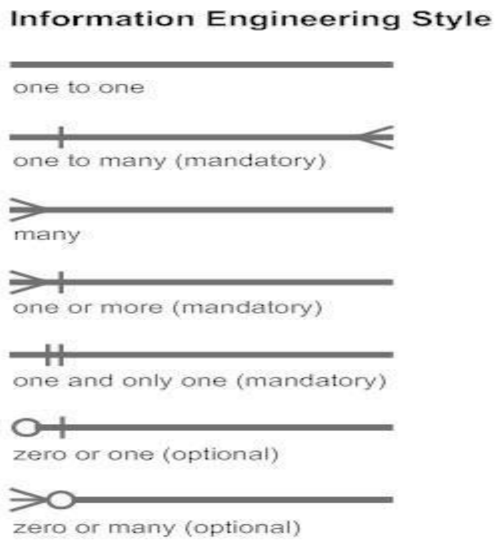
A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary. 

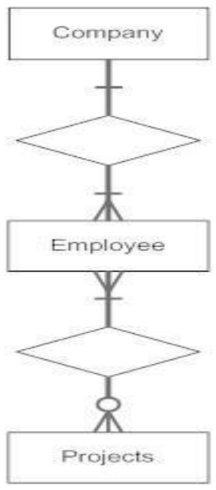
Connecting lines, solid lines that connect attributes to show the relationships of entities in the diagram.

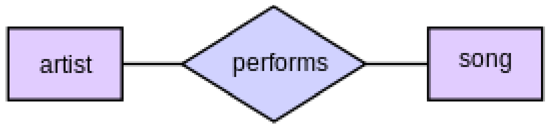
Cardinality specifies how many instances of an entity relate to one instance of another entity. Ordinality is also closely linked to cardinality. While cardinality specifies the occurrences of a relationship, ordinality describes the relationship as either mandatory or optional. In other words, cardinality specifies the maximum number of relationships and ordinality specifies the absolute minimum number of relationships.



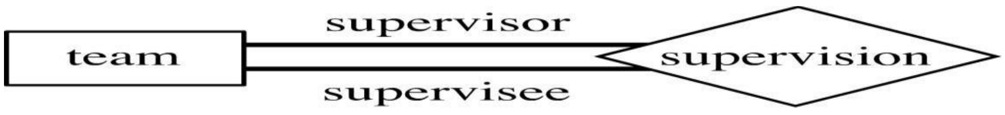
There are many notation styles that express cardinality.
Information Engineering Style







A **recursive relationship** is one in which the same entity participates more than once in the relationship.

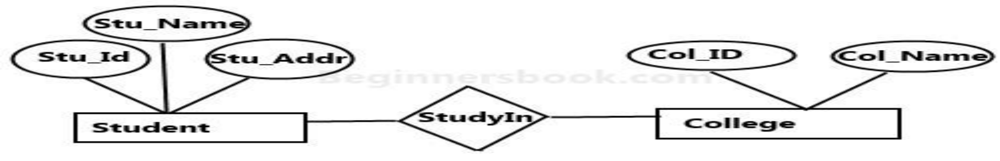


What is an Entity Relationship Diagram (ER Diagram)?

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

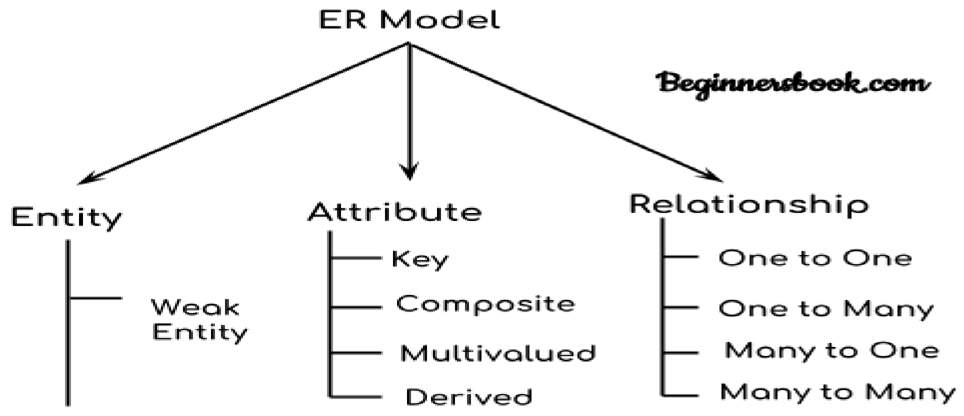
A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



Sample E-R Diagram

Components of a ER Diagram

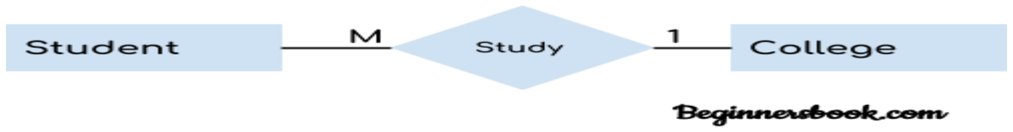


Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:
1. Entity
2. Attribute
3. Relationship

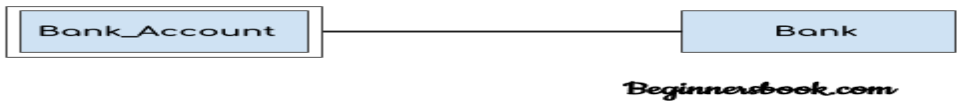
1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.



Weak Entity:

An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.



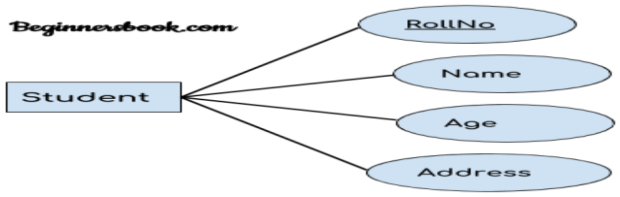
2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

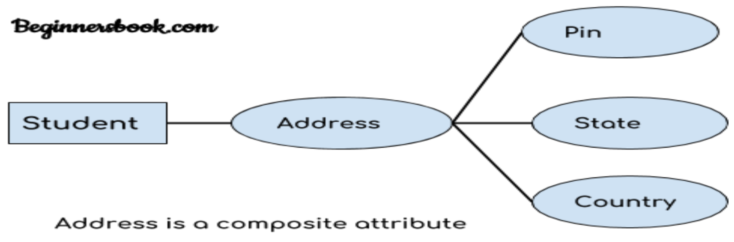
1. Key attribute:

A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined.**



2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



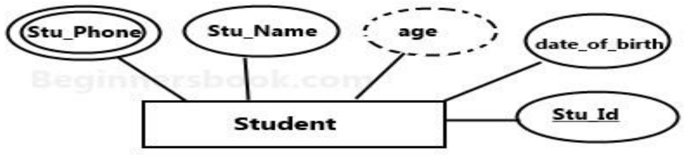
Address is a composite attribute

3. Multivalued attribute:
An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:



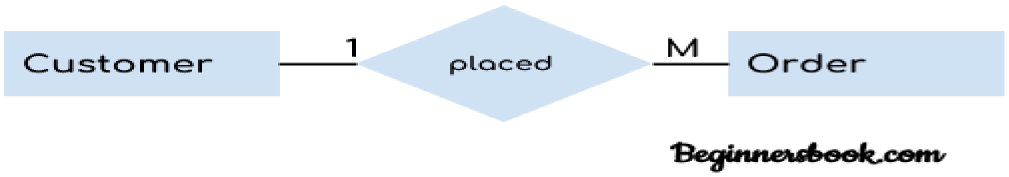
3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. **Recursive relationship:** The same entity participates more than once in the relationship.

There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

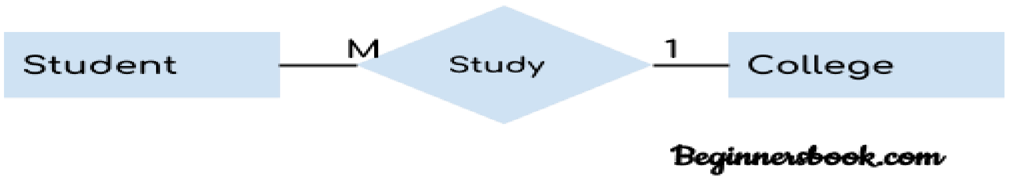
1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.



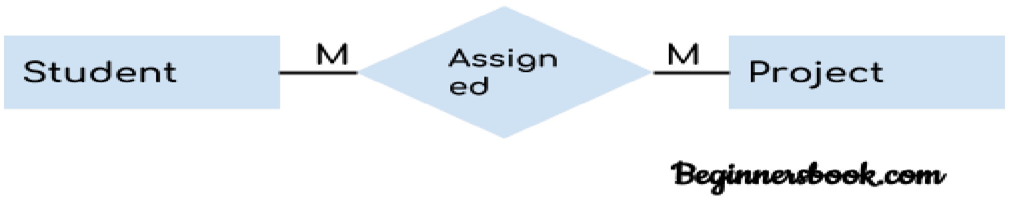
3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.



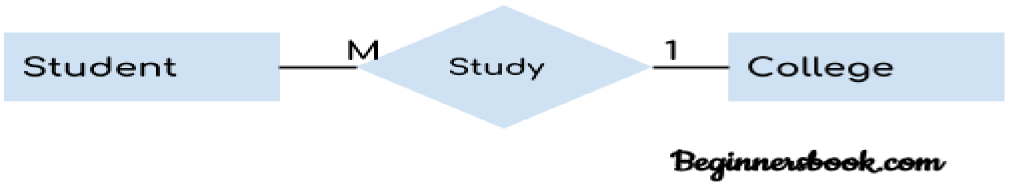
4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.



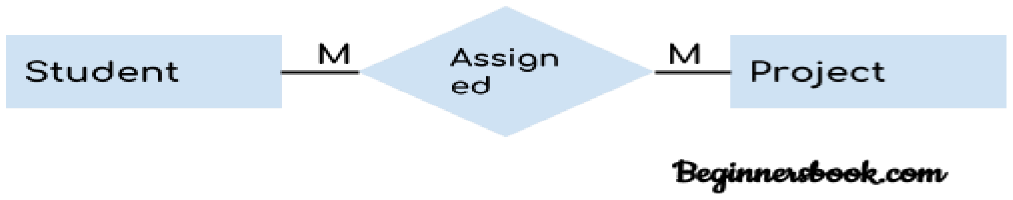
3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.



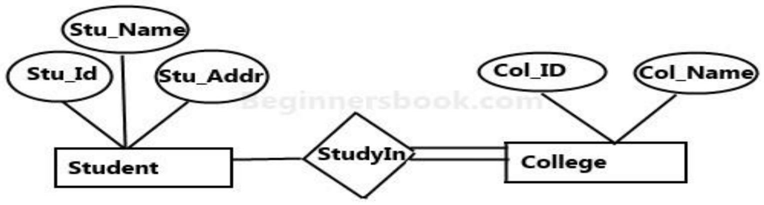
4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.



Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have at-least one associated Student.



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

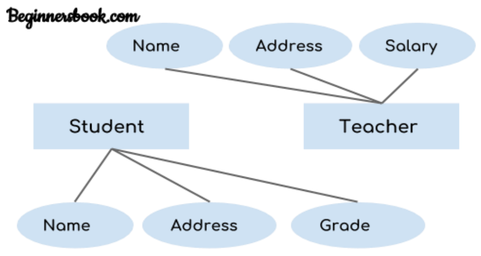
DBMS Generalization

Generalization is a process in which the common attributes of more than one entities form a new entity. This newly formed entity is called generalized entity.

Generalization Example

Lets say we have two entities Student and Teacher.
Attributes of Entity Student are: Name, Address & Grade
Attributes of Entity Teacher are: Name, Address & Salary

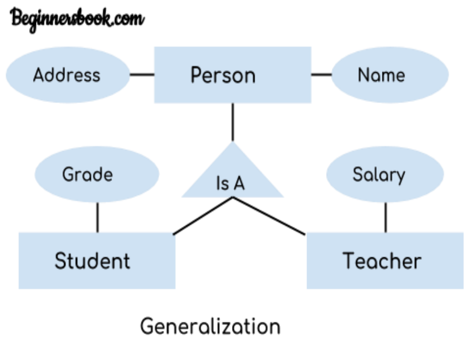
The ER diagram before generalization looks like this:



Before Generalization

These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes. Lets have a look at the ER model after generalization.

The ER diagram after generalization:
We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see in the following ER diagram that after the generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).

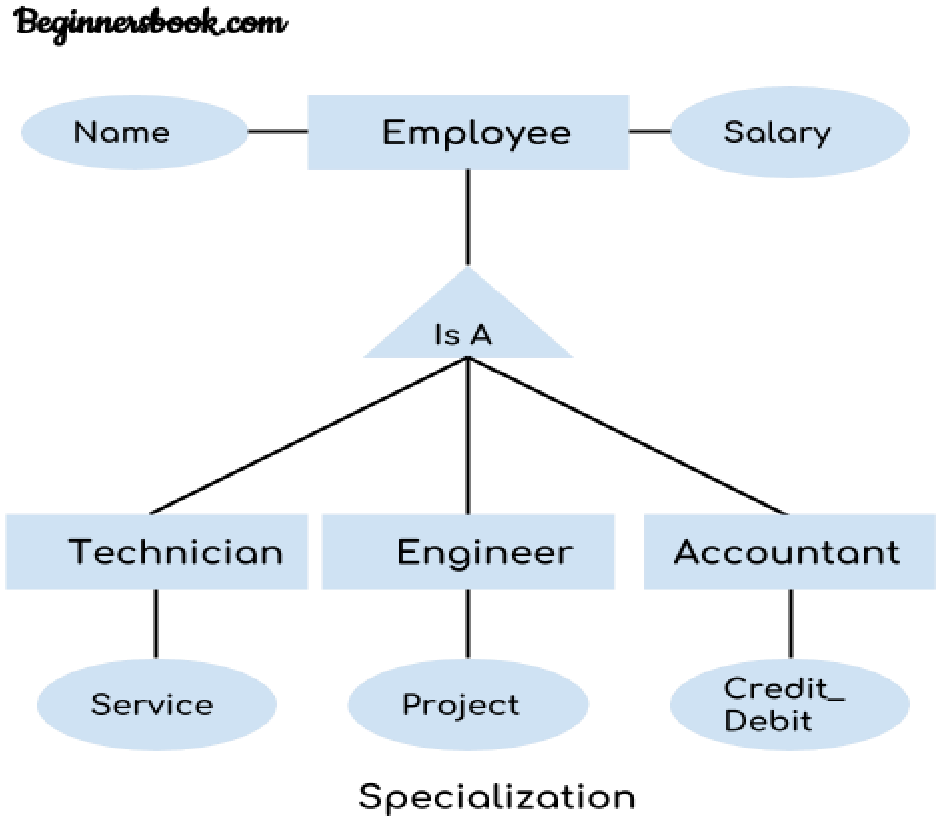


Note:
1. Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher level new entity.
2. The new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.

Specialization is a process in which an entity is divided into sub-entities. You can think of it as a reverse process of generalization, in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process.

The idea behind Specialization is to find the subsets of entities that have few distinguish attributes. For example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish attributes.

Specialization Example

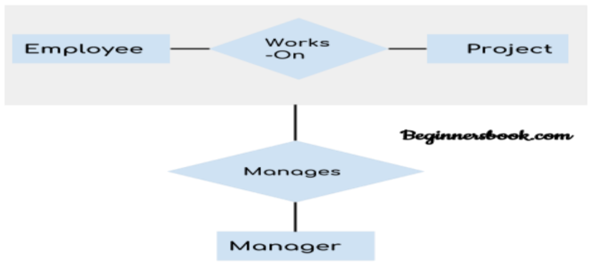


Specialization

In the above diagram, we can see that we have a higher level entity "Employee" which we have divided in sub entities "Technician", "Engineer" & "Accountant". All of these are just an employee of a company, however their role is completely different and they have few different attributes. Just for the example, I have shown that Technician handles service requests, Engineer works on a project and Accountant handles the credit & debit details. All of these three employee types have few attributes common such as name & salary which we had left associated with the parent entity "Employee" as shown in the above diagram.

DBMS Aggregration

Aggregation is a process in which a single entity alone is not able to make sense in a relationship so the relationship of two entities acts as one entity. I know it sounds confusing but don't worry the example we will take, will clear all the doubts.
Aggregration Example



In real world, we know that a manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity "Manager" makes a "manages" relationship with either "Employee" or "Project" entity alone then it will not make any sense because he has to manage both. In these cases the relationship of two entities acts as one entity. In our example, the relationship "Works-On" between "Employee" & "Project" acts as one entity that has a relationship "Manages" with the entity "Manager"

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Key constraints force that −

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints :Attributes have specific values in **real-world scenario**. For example, age can only be a **positive integer**. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Referential integrity constraints work on the concept of Foreign Keys. **A foreign key** is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

The primary key is also used to reference other tables (to be elaborated later).

You have to decide which column(s) is to be used for primary key. The decision may not be straight forward but the primary key shall have these properties:

- The values of primary key shall be unique (i.e., no duplicate value). For example, customerName may not be appropriate to be used as the primary key for the Customers table, as there could be two customers with the same name.
- The primary key shall always have a value. In other words, it shall not contain NULL.

Consider the followings in choosing the primary key:

- The primary key shall be simple and familiar, e.g., employeeID for employees table and isbn for books table.
- The value of the primary key should not change. Primary key is used to reference other tables. If you change its value, you have to change all its references; otherwise, the references will be lost. For example, phoneNumber may not be appropriate to be used as primary key for table Customers, because it might change.
- Primary key often uses integer (or number) type. But it could also be other types, such as texts. However, it is best to use numeric column as primary key for efficiency.
- Primary key could take an arbitrary number. Most RDBMSs support so-called *autoincrement* (or AutoNumber type) for integer primary key, where (current maximum value + 1) is assigned to the new record. This arbitrary number is *factless*, as it contains no factual information. Unlike factual information such as phone number, fact-less number is ideal for primary key, as it does not change.
- Primary key is usually a single column (e.g., customerID or productCode). But it could also make up of several columns. You should use as few columns as possible.

Integrity Rules

You should also apply the integrity rules to check the integrity of your design:

Entity Integrity Rule: The primary key cannot contain NULL. Otherwise, it cannot uniquely identify the row. For composite key made up of several columns, none of the column can contain NULL. Most of the RDBMS check and enforce this rule.

Referential Integrity Rule: Each foreign key value must be matched to a primary key value in the table referenced (or parent table).

-You can insert a row with a foreign key in the child table only if the value exists in the parent table.

-If the value of the key changes in the parent table (e.g., the row updated or deleted), all rows with this foreign key in the child table(s) must be handled accordingly. You could either (a) disallow the changes; (b) cascade the change (or delete the records) in the child tables accordingly; (c) set the key value in the child tables to NULL

DATABASE VIEWS

A database management system provides **three views** of the database data:

- The **external level** defines how each group of end-users sees the organization of data in the database. A single database can have any number of views at the external level.
- The **conceptual level** unifies the various external views into a compatible global view. It provides the synthesis of all the external views. It is out of the scope of the various

database end-users, and is rather of interest to database application developers and database administrators.

- The **internal level** (or *physical level*) is the internal organization of data inside a DBMS (see Implementation section below). It is concerned with cost, performance, scalability and other operational matters. It deals with storage layout of the data, using storage structures such as indexes to enhance performance. Occasionally it stores data of individual views (materialized views), computed from generic data, if performance justification exists for such redundancy. It balances all the external views' performance requirements, possibly conflicting, in an attempt to optimize overall performance across all activities