आईएफटीएम विश्वविद्यालय, मुरादाबाद, उत्तर प्रदेश

**IFTM University, Moradabad, Uttar Pradesh**

**NAAC ACCREDITED**

# E-Content

# IFTM University, Moradabad

## What is data processing?

Data in its raw form is not useful to any organization. Data processing is the method of collecting raw data and translating it into usable information. It is usually performed in a step-by-step process by a team of data scientists and data engineers in an organization. The raw data is collected, filtered, sorted, processed, analyzed, stored and then presented in a readable format.
Data processing is crucial for organizations to create better business strategies and increase their competitive edge. By converting the data into a readable format like graphs, charts and documents, employees throughout the organization can understand and use the data.

## Data Processing Cycle

The data processing cycle consists of a series of steps where raw data (input) is fed into a process (CPU) to produce actionable insights (output). Each step is taken in a specific order, but the entire process is repeated in a cyclic manner. The first data processing cycle's output can be stored and fed as the input for the next cycle.

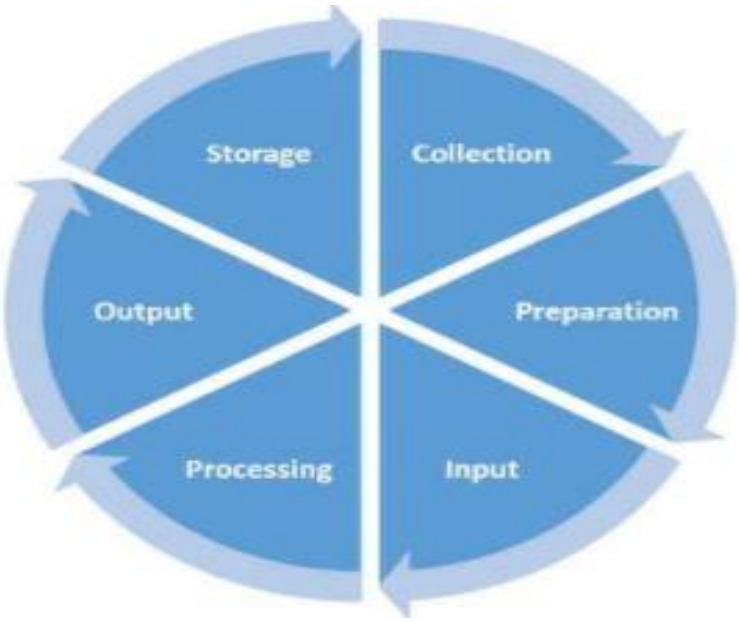



**Fig: Data processing cycle (source)**

Generally, there are six main steps in the data processing cycle:

### Step 1: Collection

The collection of raw data is the first step of the data processing cycle. The type of raw data collected has a huge impact on the output produced. Hence, raw data should be gathered from defined and accurate sources so that the subsequent findings are valid and usable. Raw data can

include monetary figures, website cookies, profit/loss statements of a company, user behavior, etc.

**Step 2: Preparation**

Data preparation or data cleaning is the process of sorting and filtering the raw data to remove unnecessary and inaccurate data. Raw data is checked for errors, duplication, miscalculations or missing data, and transformed into a suitable form for further analysis and processing. This is done to ensure that only the highest quality data is fed into the processing unit.

**Step 3: Input**

In this step, the raw data is converted into machine readable form and fed into the processing unit. This can be in the form of data entry through a keyboard, scanner or any other input source.

**Step 4: Data Processing**

In this step, the raw data is subjected to various data processing methods using machine learning and artificial intelligence algorithms to generate a desirable output. This step may vary slightly from process to process depending on the source of data being processed (data lakes, online databases, connected devices, etc.) and the intended use of the output.

**Step 5: Output (INFORMATION)**

The data is finally transmitted and displayed to the user in a readable form like graphs, tables, vector files, audio, video, documents, etc. This output can be stored and further processed in the next data processing cycle.

**Step 6: Storage**

The last step of the data processing cycle is storage, where data and **metadata** is stored for further use. This allows for quick access and retrieval of information whenever needed, and also allows it to be used as input in the next data processing cycle directly.

## Data Processing Methods

There are three main data processing methods - manual, mechanical and electronic.

- **Manual Data Processing**

In this data processing method, data is processed manually. The entire process of data collection, filtering, sorting, calculation and other logical operations are all done with human intervention without the use of any other electronic device or automation software. It is a low-cost method and requires little to no tools, but produces high errors, high labor costs and lots of time.

- **Mechanical Data Processing**

Data is processed mechanically through the use of devices and machines. These can include simple devices such as calculators, typewriters, printing press, etc. Simple data processing operations can be achieved with this method. It has much lesser errors than manual data processing, but the increase of data has made this method more complex and difficult.

- **Electronic Data Processing**

Data is processed with modern technologies using data processing software and programs. A set of instructions is given to the software to process the data and yield output. This method is the most expensive but provides the fastest processing speeds with the highest reliability and accuracy of output.

**Examples of Data Processing:-**

Data processing occurs in our daily lives whether we may be aware of it or not. Here are some real-life examples of data processing:

- A stock trading software that converts millions of stock data into a simple graph
- An e-commerce company uses the search history of customers to recommend similar products
- A self-driving car uses real-time data from sensors to detect if there are pedestrians and other cars on the road

**Types of Data Processing**

There are number of methods and techniques which can be adopted for processing of data depending upon the requirements, time availability, software and hardware capability of the technology being used for data processing. There are number of types of data processing methods.

- **Batch Processing**

This is one of the widely used type of data processing which is also known as Serial/Sequential, Tacked/Queued offline processing. The fundamental of this type of processing is that different jobs of different users are processed in the order received. Once the stacking of jobs is complete

they are provided/sent for processing while maintaining the same order. This processing of a large volume of data helps in reducing the processing cost thus making it data processing economical. Batch Processing is a method where the information to be organized is sorted into groups to allow for efficient and sequential processing.

- **Online Processing**

Online Processing is a method that utilizes Internet connections and equipment directly attached to a computer. It is used mainly for information recording and research. Real-Time Processing is a technique that can respond almost immediately to various signals to acquire and process information. ~~Distributed Processing is commonly utilized by remote workstations connected to one big central workstation or server. ATMs are good examples of this data processing method. Examples include: Examination, payroll and billing system.~~

- **Real time processing**

As the name suggests this method is used for carrying out real-time processing. This is required where the results are displayed immediately or in lowest time possible. The data fed to the software is used almost instantaneously for processing purpose. The nature of processing of this type of data processing requires use of internet connection and data is stored/used online. No lag is expected/acceptable in this type and receiving and processing of transaction is carried out simultaneously. This method is costly than batch processing as the hardware and software capabilities are better. Example includes banking system, tickets booking for flights, trains, movie tickets, rental agencies etc. This technique can respond almost immediately to various signals to acquire and process information. These involve high maintenance and upfront cost attributed to very advanced technology and computing power. Time saved is maximum in this case as the output is seen in real time. For example in banking transactions.

- **Distributed Processing**

This method is commonly utilized by remote workstations connected to one big central workstation or server. ATMs are good examples of this data processing method. All the end machines run on a fixed software located at a particular place and make use of exactly same information and sets of instruction.

- **Multiprocessing**

This type of processing perhaps the most widely used types of data processing. It is used almost everywhere and forms the basis of all computing devices relying on processors. Multi processing makes use of CPUs (more than one CPU). The task or sets of operations are divided between CPUs available simultaneously thus increasing efficiency and throughput. The breakdown of

jobs which needs be performed are sent to different CPUs working parallel within the mainframe. The result and benefit of this type of processing is the reduction in time required and increasing the output. Moreover CPUs work independently as they are not dependent on other CPU; failure of one CPU does not result in halting the complete process as the other CPUs continue to work. Examples include processing of data and instructions in computer, laptops, mobile phones etc.

- **Time sharing**

Time based used of CPU is the core of this data processing type. The single CPU is used by multiple users. All users share same CPU but the time allocated to all users might differ. The processing takes place at different intervals for different users as per allocated time. Since multiple users can use this type it is also referred as multi access system. This is done by providing a terminal for their link to main CPU and the time available is calculated by dividing the CPU time between all the available users as scheduled.

# Part -2

**Types of Data Processing**

There are different types of data processing based on the source of data and the steps taken by the processing unit to generate an output. There is no one-size-fits-all method that can be used for processing raw data.

| Type | Uses |
|---|---|
| Batch Processing | Data is collected and processed in batches. Used for large |

| | |
|---|---|
| | amounts of data.<br><br>Eg: payroll system |
| Real-time Processing | Data is processed within seconds when the input is given. Used for small amounts of data.<br><br>Eg: withdrawing money from ATM |
| Online Processing | Data is automatically fed into the CPU as soon as it becomes available. Used for continuous processing of data.<br><br>Eg: barcode scanning |
| Multiprocessing | Data is broken down into frames and processed using two or more CPUs within a single computer system. Also known as parallel processing.<br><br>Eg: weather forecasting |
| Time-sharing | Allocates computer resources and data in time slots to several users simultaneously. |

# DATA PROCESSING

- Data processing is the act of handling or manipulating data in some fashion. Regardless of the activities involved in it, processing tries to assign meaning to data. Data processing is the process through which facts and figures are collected, assigned meaning, communicated to others and retained for future use.
- Hence we can define data processing as a series of actions or operations that converts data into useful information

# STEPS IN DATA PROCESSING

- Collection
- Conversion
- Manipulation
- Storage
- Communication

# COLLECTION

- Data originates in the form of events transaction or some observations.
- This data is then recorded in some usable form. Data may be initially recorded on paper source documents and then converted into a machine usable form for processing.
- Alternatively, they may be recorded by a direct input device in a paperless, machine-readable form. Data collection is also termed as data capture.

# CONVERSION

- Once the data is collected, it is converted from its source documents to a form that is more suitable for processing. The data is first codified by assigning identification codes.
- A code comprises of numbers, letters, special characters, or a combination of these It is useful to codify data, when data requires classification.
- To classify means to categorize, i.e., data with similar characteristics are placed in similar categories or groups. For example, one may like to arrange accounts data according to account number or date. Hence a balance sheet can easily be prepared.

# MANIPULATION

- Once data is collected and converted, it is ready for the manipulation function which converts data into information. Manipulation consists of following activities:

1. Sorting
2. Calculating
3. Summarizing
4. Comparing

# MANAGING THE OUTPUT RESULTS

Once data has been captured and manipulated following activities may be carried out :

- **Storing**

To store is to hold data for continued or later use.

- **Retrieving**

To retrieve means to recover or find again the stored data or information. Retrieval techniques use data storage devices.

# COMMUNICATION

- Communication is the process of sharing information. Thus, communication involves the transfer of data and information produced by the data processing system to the prospective users of such information or to another data processing system.
- As a result, reports and documents are prepared and delivered to the users. In electronic data processing, results are communicated through display units or terminals.

# REPRODUCTION

- To reproduce is to copy or duplicate data or information.
- This reproduction activity may be done by hand or by machine.

# DATA ORGANISATION

Data can be arranged in a variety of ways, but a hierarchical approach to organisation is generally recommended.

- **Data Item**

A data item is the smallest unit of information stored in computer file.

- **Field**

Data items are physically arranged as fields in a computer file. Their length may be fixed or variable.

- **Record**

  A record is a collection of related data items or fields. Each record normally corresponds to a specific unit of information.

- **File**

  The collection of records is called a file. A file contains all the related records for an application.

- **Database**

  The collection of related files is called a database. A database contains all the related files for a particular application.

# What Is Data Processing: Cycle, Types, Methods, Steps and Examples

## What Is Data Processing?

Data in its raw form is not useful to any organization. Data processing is the method of collecting raw data and translating it into usable information. It is usually performed in a step-by-step process by a team of data scientists and data engineers in an organization. The raw data is collected, filtered, sorted, processed, analyzed, stored, and then presented in a readable format.

Data processing is crucial for organizations to create better business strategies and increase their competitive edge. By converting the data into a readable format like graphs, charts, and documents, employees throughout the organization can understand and use the data.

Now that we understood what is data processing, let's understand its cycle process.

## Data Processing Cycle

The data processing cycle consists of a series of steps where raw data (input) is fed into a process (CPU) to produce actionable insights (output). Each step is taken in a specific order, but the entire process is repeated in a cyclic manner. The first data processing cycle's output can be stored and fed as the input for the next cycle.

Generally, there are six main steps in the data processing cycle:

### Step 1: Collection

The collection of raw data is the first step of the data processing cycle. The type of raw data collected has a huge impact on the output produced. Hence, raw data should be gathered from defined and accurate sources so that the subsequent findings are valid and usable. Raw data can include monetary figures, website cookies, profit/loss statements of a company, user behavior, etc.

### Step 2: Preparation

Data preparation or data_cleaning is the process of sorting and filtering the raw data to remove unnecessary and inaccurate data. Raw data is checked for errors, duplication, miscalculations or missing data, and transformed into a suitable form for further analysis and processing. This is done to ensure that only the highest quality data is fed into the processing unit.

### Step 3: Input

In this step, the raw data is converted into machine readable form and fed into the processing unit. This can be in the form of data entry through a keyboard, scanner or any other input source.

### Step 4: Data Processing

In this step, the raw data is subjected to various data processing methods using machine_learning and_artificial_intelligence_algorithms to generate a desirable output. This step may vary slightly from process to process depending on the source of data being processed (data_lakes, online databases, connected devices, etc.) and the intended use of the output.

### Step 5: Output

The data is finally transmitted and displayed to the user in a readable form like graphs, tables, vector files, audio, video, documents, etc. This output can be stored and further processed in the next data processing cycle.

### Step 6: Storage

The last step of the data processing cycle is storage, where data and metadata are stored for further use. This allows for quick access and retrieval of information whenever needed, and also allows it to be used as input in the next data processing cycle directly.

Now that we have learned what is data processing and its cycle, now we can look at the types.

## Types of Data Processing

There are different types of data processing based on the source of data and the steps taken by the processing unit to generate an output. There is no one-size-fits-all method that can be used for processing raw data.

| Type | Uses |
| --- | --- |
| Batch Processing | Data is collected and processed in batches. Used for large amounts of data.<br><br>Eg: payroll system |
| Real-time Processing | Data is processed within seconds when the input is given. Used for small amounts of data. |

| | |
|---|---|
| | Eg: withdrawing money from ATM |
| Online Processing | Data is automatically fed into the CPU as soon as it becomes available. Used for continuous processing of data.<br><br>Eg: barcode scanning |
| Multiprocessing | Data is broken down into frames and processed using two or more CPUs within a single computer system. Also known as parallel processing.<br><br>Eg: weather forecasting |
| Time-sharing | Allocates computer resources and data in time slots to several users simultaneously. |

# Data Processing Methods

There are three main data processing methods - manual, mechanical and electronic.

## Manual Data Processing

In this data processing method, data is processed manually. The entire process of data collection, filtering, sorting, calculation and other logical operations are all done with human intervention without the use of any other electronic device or automation software. It is a low-cost method and requires little to no tools, but produces high errors, high labor costs and lots of time.

## Mechanical Data Processing

Data is processed mechanically through the use of devices and machines. These can include simple devices such as calculators, typewriters, printing press, etc. Simple data processing operations can be achieved with this method. It has much lesser errors than manual data processing, but the increase of data has made this method more complex and difficult.

## Electronic Data Processing

Data is processed with modern technologies using data processing software and programs. A set of instructions is given to the software to process the data and yield output. This method is the most expensive but provides the fastest processing speeds with the highest reliability and accuracy of output.

# Examples of Data Processing

Data processing occurs in our daily lives whether we may be aware of it or not. Here are some real-life examples of data processing:

- A stock trading software that converts millions of stock data into a simple graph
- An e-commerce company uses the search history of customers to recommend similar products
- A digital marketing company uses demographic data of people to strategize location-specific campaigns
- A self-driving car uses real-time data from sensors to detect if there are pedestrians and other cars on the road

That was all about the article what is data processing.

## 1. Real-time Processing

Real-time processing is similar to transaction processing, in that it is used in situations where output is expected in real-time. However, the two differ in terms of how they handle data loss. Real-time processing computes incoming data as quickly as possible. If it encounters an error in incoming data, it ignores the error and moves to the next chunk of data coming in. GPS-tracking applications are the most common example of real-time data processing.

Contrast this with transaction processing. In case of an error, such as a system failure, transaction processing aborts ongoing processing and reinitializes. Real-time processing is preferred over transaction processing in cases where approximate answers suffice.

In the world of data analytics, stream processing is a common application of real-time data processing. First popularized by Apache Storm, stream processing analyzes data as it comes in. Think data from IoT sensors, or tracking consumer activity in real-time. Google BigQuery and Snowflake are examples of cloud data platforms that employ real-time processing.

## 2. Batch Processing

As the name suggests, batch processing is when chunks of data, stored over a period of time, are analyzed together, or in batches. Batch processing is required when a large volume of data needs to be analyzed for detailed insights. For example, sales figures of a company over a period of time will typically undergo batch processing. Since there is a large volume of data involved, the system will take time to process it. By processing the data in batches, it saves on computational resources.

Batch processing is preferred over real-time processing when accuracy is more important than speed. Additionally, the efficiency of batch processing is also measured in terms of throughput. Throughput is the amount of data processed per unit time.

## 3. Multiprocessing

Multiprocessing is the method of data processing where two or more than two processors work on the same dataset. It might sound exactly like distributed processing, but there is a difference. In multiprocessing, different processors reside within the same system. Thus, they are present in the same geographical location. If there is a component failure, it can reduce the speed of the system.

Distributed processing, on the other hand, uses servers that are independent of each other and can be present in different geographical locations. Since almost all systems today come with the ability to process data in parallel, almost every data processing system uses multiprocessing.However, in the context of this article, multiprocessing can be seen as having an on-premise data processing system. Typically, companies that handle very sensitive information might choose on-premise data processing as opposed to distributed processing. For example, pharmaceutical companies or businesses working in the oil and gas extraction industry.

# BCAGE 203

# Unit-2

## Data Storage Devices

### Data storage defined

There are two types of digital information: input and output data. Users provide the input data. Computers provide output data. But a computer's CPU can't compute anything or produce output data without the user's input.

Users can enter the input data directly into a computer. However, they have found early on in the computer-era that continually entering data manually is time- and energy-prohibitive. One short-term solution is computer memory, also known as random access memory (RAM). But its storage capacity and memory retention are limited. Read-only memory (ROM) is, as the name suggests, the data can only be read but not necessarily edited. They control a computer's basic functionality.

### How data storage works

In simple terms, modern computers, or terminals, connect to storage devices either directly or through a network. Users instruct computers to access data from and store data to these storage devices. However, at a fundamental level, there are two foundations to data storage: the form in which data takes and the devices data is recorded and stored on.

### Data storage devices

To store data, regardless of form, users need storage devices. Data storage devices come in two main categories: direct area storage and network-based storage.

**Direct area storage**, also known as direct-attached storage (DAS), is as the name implies. This storage is often in the immediate area and directly connected to the computing machine accessing it. Often, it's the only machine connected to it. DAS can provide decent local backup services, too, but sharing is limited. DAS devices include floppy disks, optical discs —compact discs (CDs) and digital video discs (DVDs)—hard disk drives (HDD), flash drives and solid-state drives (SSD).
**Network-based storage** allows more than one computer to access it through a network, making it better for data sharing and collaboration. Its off-site storage capability also makes it better suited for backups and data protection. Two common network-based storage setups are network-attached storage (NAS) and storage area network (SAN).

NAS is often a single device made up of redundant storage containers or a redundant array of independent disks (RAID). SAN storage can be a network of multiple devices of various types, including SSD and flash storage, hybrid storage, hybrid cloud storage, backup software and appliances, and cloud storage. Here are how NAS and SAN differ:

**NAS**

- Single storage device or RAI
- File storage system
- TCP/IP Ethernet network
- Limited users
- Limited speed
- Limited expansion options
- Lower cost and easy setup

**SAN**

- Network of multiple devices
- Block storage system
- Fibre Channel network
- Optimized for multiple users
- Faster performance
- Highly expandable
- Higher cost and complex setup

**10 Digital Data Storage Devices for Computers**

1. Hard Drive Disks
2. Floppy Disks
3. Tapes
4. Compact Discs (CDs)
5. DVD and Blu-ray Discs
6. USB Flash Drives
7. Secure Digital Cards (SD Card)s
8. Solid-State Drives (SSDs)
9. Cloud Storage
10. Punch Cards

**1. Hard Disk Drives**

A hard disk drive (also known as a hard drive, HD, or HDD) can be found installed in almost every desktop and laptop computer. It stores files for the operating system and software programs as well as user documents, such as photographs, text files, videos, and audio. The hard drive uses magnetic storage to record and retrieve digital information to and from one or more fast-spinning disks.

**2. Floppy Disks**

Also know as a diskette, floppy, or FD, the floppy disk is another type of storage medium that uses magnetic storage technology to store information. Floppy disks were once a common storage device for computers and were very common from the mid-1970s through to the start of the 21st century.

The earliest floppies were 8 inches (203 mm) in size, but these were replaced first by 5.25-inch (133 mm) disk drives and finally by 3.5-inch (90 mm) versions.

### 3. Tapes

In the past, magnetic tape was often used for digital data storage because of its low cost and ability to store large amounts of data. The technology essentially consisted of a thin, magnetically coated piece of plastic wrapped around wheels. Its relative slowness and unreliability compared to other data storage solutions have resulted in it now being largely abandoned as a storage medium.

### 4. Compact Discs (CDs)

The compact disc, (or CD for short) is a form of optical storage, a technology that employs lasers and lights to read and write data. Initially, compact discs were used purely for music, but in the late 1980s, they began to be used for computer data storage.

Initially, the compact discs that were introduced were CD-ROMs (read-only), but these were followed by CD-Rs (writable compact discs) and CD-RWs (rewritable compact discs).

### 5. DVD and Blu-ray Discs

The DVD (digital versatile disc) and Blu-ray disc (BD) are formats of digital optical disc data storage which have superseded compact discs, mainly because of their much greater storage capacity.

A Blu-ray disc, for example, can store 25 GB (gigabytes) of data on a single-layer disc and 50 GB on a dual-layer disc. In comparison, a standard CD is the same physical size, but only holds 700 MB (megabytes) of digital data.

USB flash drives are often used by students and professionals to save work from one computer and continue working on it on another.

### 6. USB Flash Drives

Also known as a thumb drive, pen drive, flash drive, memory stick, jump drive, and USB stick, the USB flash drive is a flash-memory data-storage device that incorporates an integrated USB interface. Flash memory is generally more efficient and reliable than optical media, being smaller, faster, and possessing much greater storage capacity. Flash drives are also more durable due to a lack of moving parts.

### 7. Secure Digital Cards (SD Cards)

SD cards are commonly used in multiple electronic devices, including digital cameras and mobile phones. Although there are different sizes, classes, and capacities available, they all use a rectangular design with one side "chipped off" to prevent the card from being inserted into a camera or computer the wrong way.

**8. Solid-State Drives (SSDs)**

A solid-state drive uses flash memory to store data and is sometimes used in devices such as netbooks, laptops, and desktop computers instead of a traditional hard disk drive.

The advantages of an SSD over an HDD include a faster read/write speed, noiseless operation, greater reliability, and lower power consumption. The biggest downside is cost, with an SSD offering lower capacity than an equivalently priced HDD.

**9. Cloud Storage**

With users increasingly operating multiple devices in multiple places, many are adopting online cloud-computing solutions. Cloud computing basically involves accessing services over a network via a collection of remote servers.

Although the idea of a "cloud of computers" may sound rather abstract to those unfamiliar with this metaphorical concept, in practice, it can provide powerful storage solutions for devices that are connected to the internet.

**10. Punch Cards**

Punch cards (or punched cards) were a common method of data storage used with early computers. Basically, they consisted of a paper card with punched or perforated holes created by hand or machine. The cards were entered into computers to enable the storage and accessing of information.

This data-storage medium pretty much disappeared as new and better technologies were developed.

What Does Tape Cartridge Mean?

A tape cartridge is a storage device that contains a spool of magnetic tape used to store different kinds of data, from corporate data to audio and video files. Each cartridge is designed to fit into a compatible audio/video recorder system or computer system. In the context of computing, however, a tape cartridge is the magnetic tape storage cartridge used in tape library units to store digital data on magnetic tape, which is packaged in cassettes and cartridges.

Tape cartridges are also known as data cartridges.

**6 Common Causes of Digital Data Loss**

There are a number of ways that digital data can be lost. I've listed six of the most common ways below. Generally speaking, the best way to protect data is to back it up in different places.

1. **Accidental deletions:** This is a very common problem and has happened to most people who deal with data, including myself. As well as deletion, reformatting a device can also result in the loss of stored information.
2. **Power failures:** Many electronic devices depend on electricity to function properly and maintain data. A loss of power can therefore be disruptive or destructive, especially in cases where the power loss is sudden. As well as power losses, power surges can also cause problems.
3. **Spills, drops, and other physical accidents:** Anything that causes physical damage to the storage device can corrupt data or prevent access to it. Even minor accidents, such as knocking over a cup of coffee, might be all it takes to cause the loss of large amounts of data.
4. **Viruses and other forms of malware:** Many modern forms of digital data storage are exposed to the internet. This means that the data risks being corrupted by malware, either directly, or via wider damage being caused to say, the operating system.
5. **Theft:** Whether through burglary, pickpocketing, mugging, or other forms of theft, you can lose the entire device and all the information that's on it.
6. **Fires, floods, explosions, and other catastrophic events:** These can all destroy vast amounts of data. This is one of the main reasons why data should never be backed up in the same building, but rather in a separate place.

**Fixed And Variable Length Records**

A **fixed length record** is one where the length of the fields in each record has been set to be a certain maximum number of characters long. Suppose a field that was going to contain a name was set to be 25 characters long. This means that the field could only ever contain up to 25 characters. If all the fields in the record have a fixed length like this then the record is said to be a fixed length record. The problem with fixed length records is that each field very rarely contains the maximum number of characters allowed. This means that a lot of space is needlessly set aside and wasted. Also, values sometimes cannot be entered because they are too large to fit inside the allowed space in a field. The advantage of fixed length records is that they make file processing much easier because the start and end of each record is always a fixed number of characters apart. This makes it much easier to locate both indicidual records and fields.

A **variable length record** is one where the length of a field can change to allow data of any size to fit. The advantage of variable length records is that space is not wasted, only the space

needed is ever used. The main problem with variable length records is that it is much more difficult to locate the start and end of individual records and fields. This is because they are not separated by a fixed amount of characters. To separate variable length recordseach field has a special character to mark where it ends- called an end- of- field marker. When records need to be located the computer must count through the end- of- field markers to locate individual records and fields.

A file can contain:

- Fixed-length records - all the records are exactly the same length
- Variable-length records - the length of each record varies

**Differentiate between fixed length records and variable length records.**

Fixed length records:-
1.All the records in the file are of same size.
2. Leads to memory wastage.
3. Access of the records is easier and faster.

Variable length records:-
1.Different records in the file have different sizes.
2. Memory efficient.
3. Access of the records is slow.

**PRIMARY KEY**

A primary key is a field that identifies each record in a database table admitting that the primary key must contain its UNIQUE values.

The primary key column cannot have NULL values.

A table can have only one primary key constraint which may consist of single and multiple fields. When multiple keys are used in a single primary key, these fields are called a composite key.

When we specify the primary key constraint for a table, the database engine executes the data uniqueness by automatically creating an index. this index permits fast access to data when the primary key is used.

Primary Key can be specified either when the table is created using CREATE table or by changing the existing table structure using ALTER.

```
1.  CREATE TABLE Customer
2.  (SID integer,
3.  Last_Name Varchar(20)
4.  First_Name Varchar(20)
5.  PRIMARY KEY (SID)
```

```
1.  ALTER TABLE Customer ADD PRIMARY KEY (SID)
```

**SECONDARY KEY**

A secondary key shows the secondary value that is unique for each record. It can be used to identify the record and it is usually indexed. It is also termed as **Alternate key.**

A table may have a primary key that is system generated and a secondary key that comes from the sources and other processes. that is the secondary key. it is made on a field to be indexed for faster searches. A table can have more than one secondary key.

## Difference between Primary key and Secondary key ?

The main difference between primary key and secondary key is, a key that is selected for identifying each tuple in a table uniquely is termed as primary key, whereas, a key that is not selected for identifying rows, even though it is capable of determining tuples uniquely in the table are termed as the secondary key.

| BASIS | PRIMARY KEY | SECONDARY KEY / ALTERNATE KEY |
|---|---|---|
| Definition | A key that is selected for identifying each record in a database table uniquely | A key that is not selected as primary key but capable of identifying each records uniquely |
| NULL Values | Not Allowed | Allowed |
| Number of Keys | A table can have only one primary key | A table can have any number of secondary or alternate keys |

**For Example ,**


| user_id | account_number | name | email |
|---|---|---|---|
| 1 | 2123234344 | Micky | xyz@gmail.com |
| 2 | 3126234344 | Micky | abz@gmail.com |
| 3 | 5466764432 | Shiv | klm@gmail.com |
| 4 | 4355672341 | Aniket | mno@gmail.com |

In the above table, we can identify candidate keys which are :

**user_id, account_number and email**

Now from them, we can select only one primary key, in this case, we selected **user_id** as the primary key. Other remaining keys such as **account_number** and **email** as Secondary or Alternate key.

**Disk Structure in Operating System:** The actual physical details of a modern hard disk may be quite complicated. Simply, there are one or more surfaces, each of which contains several tracks, each of which is divided into sectors.

There is one read/write head for every surface of the disk. Also, the same track on all surfaces is known as a **cylinder**, When talking about movement of the read/write head, the cylinder is a useful concept, because all the heads *(one for each surface)*, move in and out of the disk together.


**Fig. Disk structure.**

We say that the “read/write head is at cylinder #2", when we mean that the top read/write head is at track #2 of the top surface, the next head is at track #2 of the next surface, the third head is at track #2 of the third surface, etc.

The unit of information transfer is the sector *(though often whole tracks may be read and written, depending on the hardware)*. As far as most file-systems are concerned, though, the sectors are what matter. In fact, we usually talk about a '**block device**'. A block often corresponds to a sector, though it need not do, several sectors may be aggregated to form a single logical block.

File Structures: Physical Storage Media File Organization, Organization of records into Blocks, SequentialFiles, Indexing and Hashing, Primary indices, Secondary indices, B+ Tree index Files, B Tree index Files, Indexing and Hashing Techniques and their Comparisons.

## What is File?

File is a collection of records related to each other. The file size is limited by the size of memory and storage medium.

**There are two important features of file:**

**1.** File Activity
**2.** File Volatility

**File activity** specifies percent of actual records which proceed in a single run.

**File volatility** addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

**File Organization**

File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation.

For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

## Types of File Organization

**There are three types of organizing the file:**

1. Sequential access file organization
2. Direct access file organization
3. Indexed sequential access file organization

## 1. Sequential access file organization

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

**Advantages of sequential file**

- It is simple to program and easy to design.
- Sequential file is best use if storage space.

**Disadvantages of sequential file**

- Sequential file is time consuming process.
- It has high data redundancy.
- Random searching is not possible.

## 2. Direct access file organization

- Direct access file is also known as random access or relative file organization.
- In direct access file, all records are stored in direct access storage device (DASD), such as hard disk. The records are randomly placed throughout the file.

- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information. It is used in accessing large databases.
- It is also called as hashing.

**Advantages of direct access file organization**

- Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.
- In direct access file, sorting of the records are not required.
- It accesses the desired records immediately.
- It updates several files quickly.
- It has better control over record allocation.

**Disadvantages of direct access file organization**

- Direct access file does not provide back up facility.
- It is expensive.
- It has less storage space as compared to sequential file.

### 3. Indexed sequential access file organization

- Indexed sequential access file combines both sequential file and direct access file organization.
- In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
- This file have multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
- The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

**Advantages of Indexed sequential access file organization**

- In indexed sequential access file, sequential file and random file access is possible.
- It accesses the records very fast if the index table is properly organized.
- The records can be inserted in the middle of the file.
- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search.

**Disadvantages of Indexed sequential access file organization**

- Indexed sequential access file requires unique keys and periodic reorganization.
- Indexed sequential access file takes longer time to search the index for the data access or retrieval.
- It requires more storage space.
- It is expensive because it requires special software.
- It is less efficient in the use of storage space as compared to other file organizations.

# File Organization

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

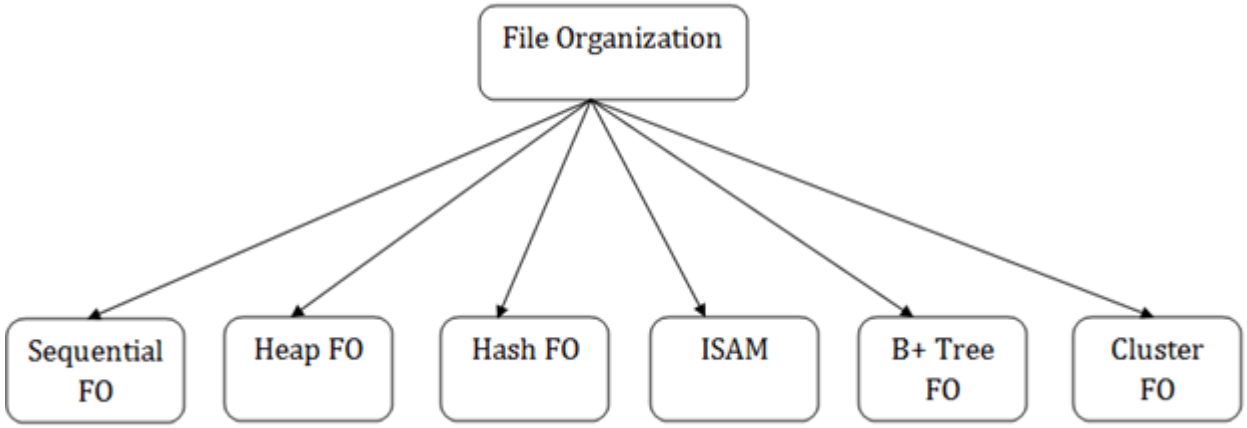
# Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

# Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:




- Sequential file organization
- Heap file organization

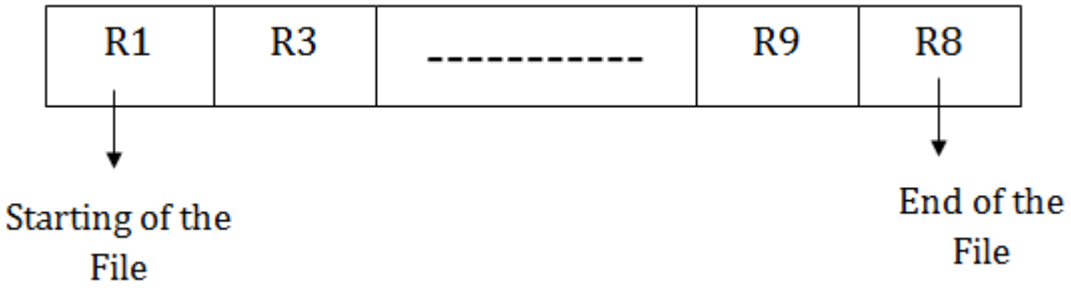
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

# Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

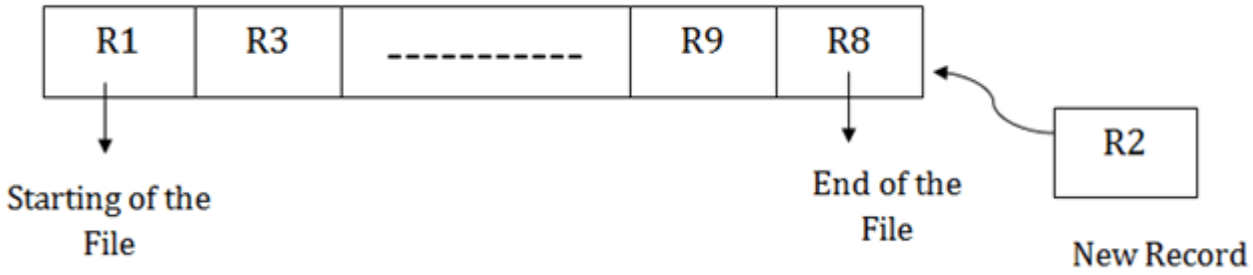
## 1. Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



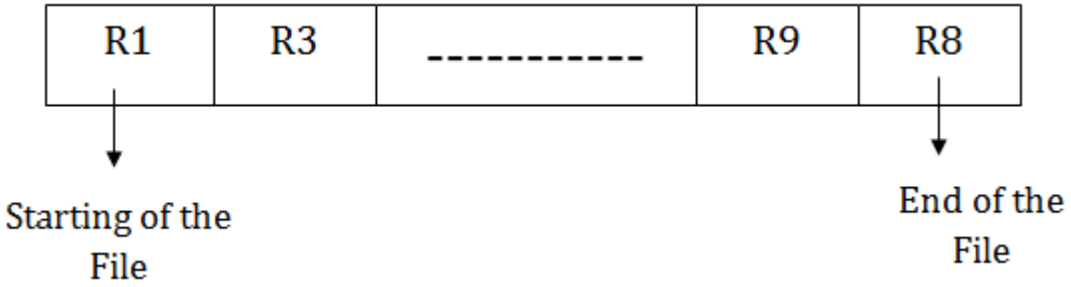

## Insertion of the new record:

Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



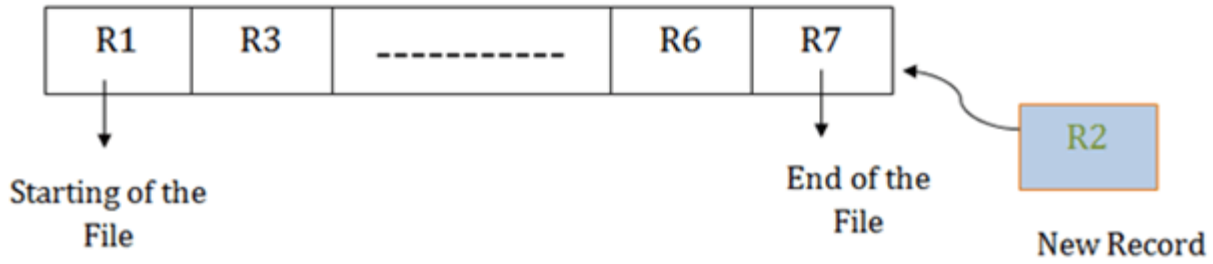

## 2. Sorted File Method:

- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.




## Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file, and then it will sort the sequence.





## Pros of sequential file organization

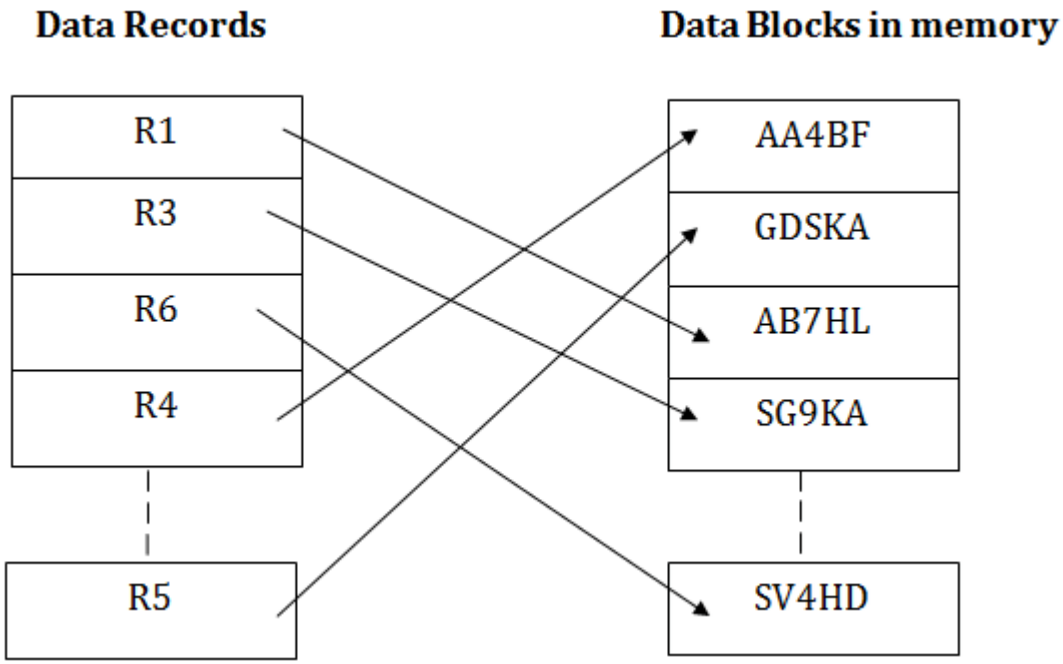
- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.

## Cons of sequential file organization

- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

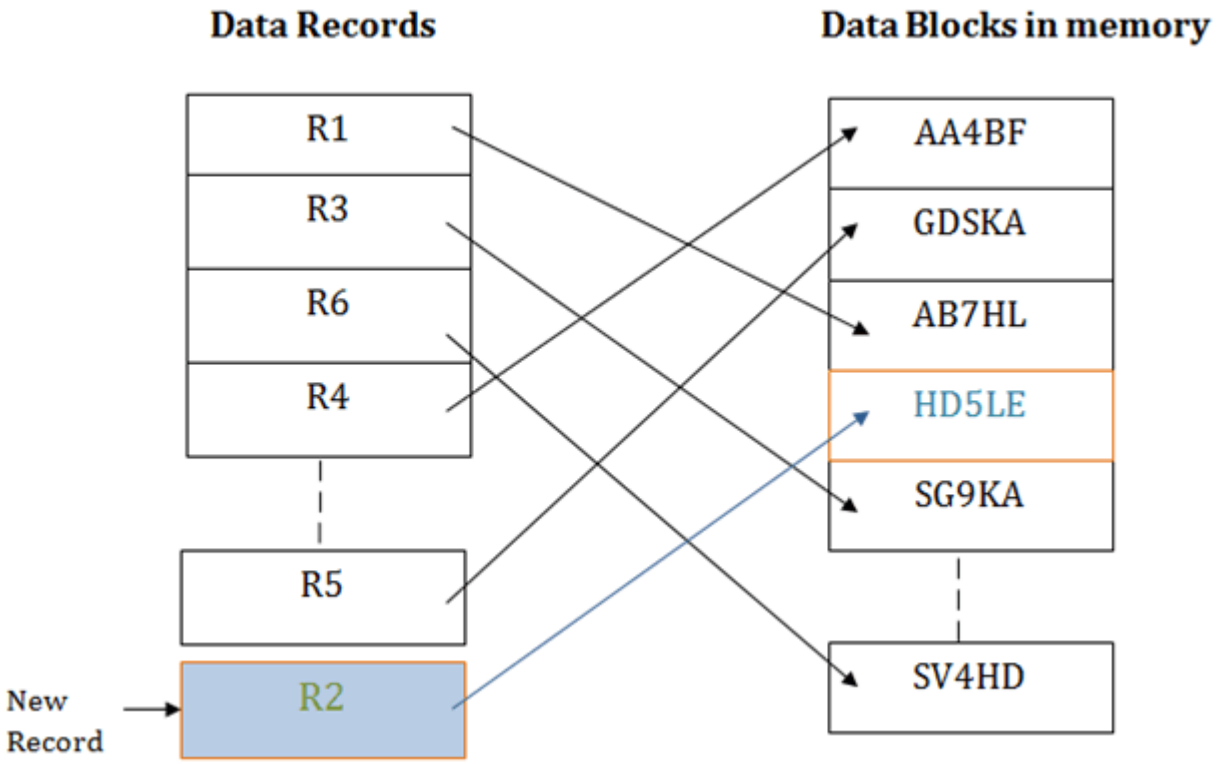
# Hash File Organization

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.



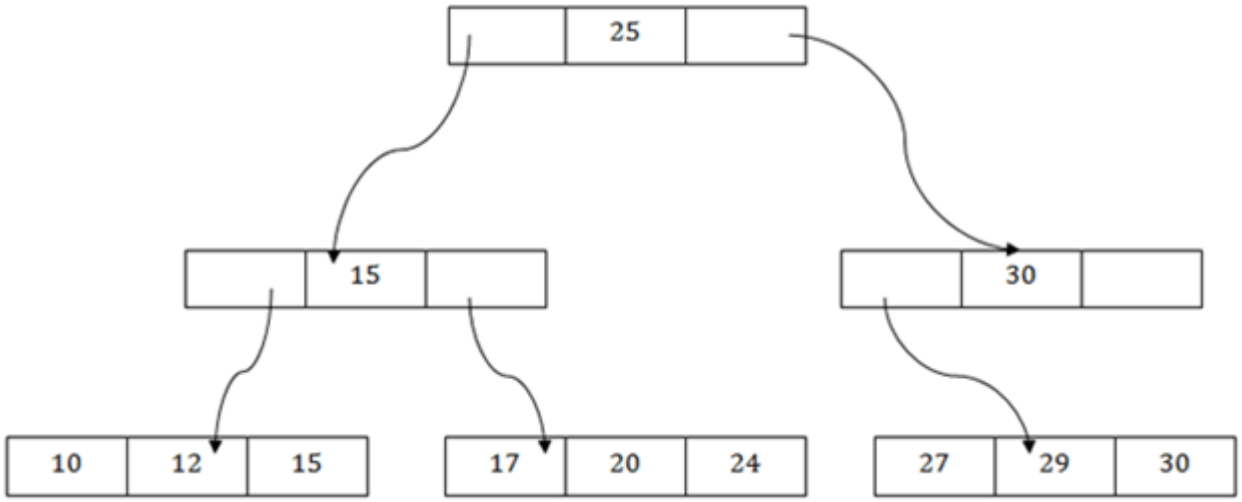
When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



# B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.




## The above B+ tree shows that:

- There is one root node of the tree, i.e., 25.
- There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.

- The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- Searching for any record is easier as all the leaf nodes are balanced.
- In this method, searching any record can be traversed through the single path and accessed easily.

## Pros of B+ tree file organization

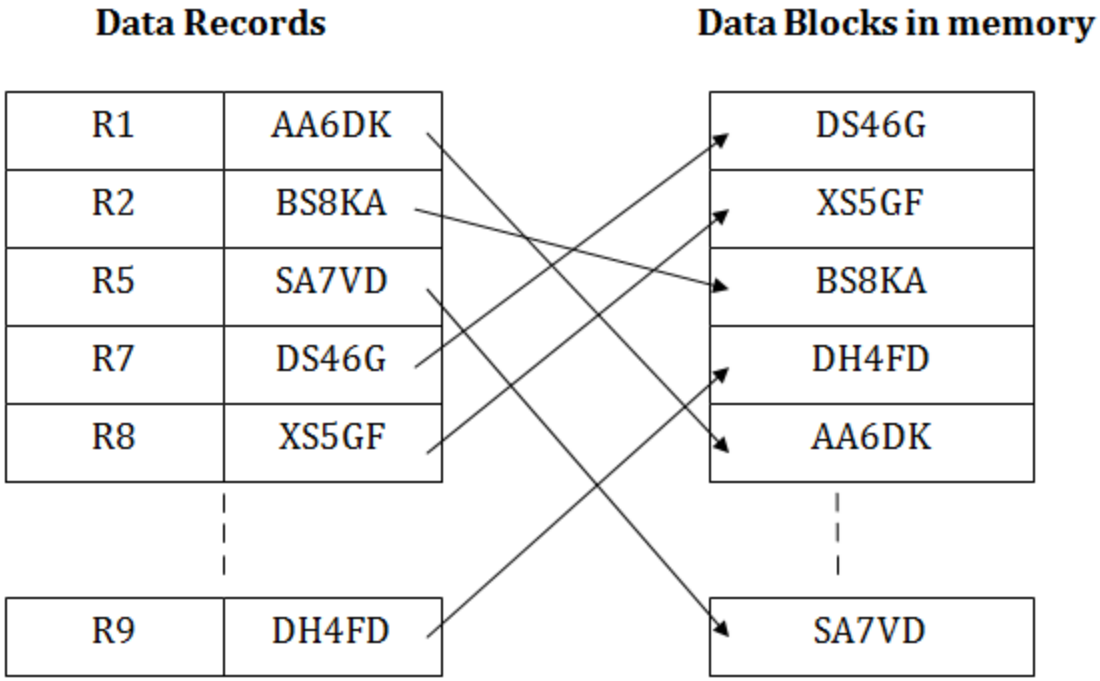
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

## Cons of B+ tree file organization

- This method is inefficient for the static method.

# Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.




If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

## Pros of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

## Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

## Cluster file organization

- When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

**EMPLOYEE**

| EMP_ID | EMP_NAME | ADDRESS | DEP_ID |
|---|---|---|---|
| 1 | John | Delhi | 14 |
| 2 | Robert | Gujarat | 12 |
| 3 | David | Mumbai | 15 |
| 4 | Amelia | Meerut | 11 |
| 5 | Kristen | Noida | 14 |
| 6 | Jackson | Delhi | 13 |
| 7 | Amy | Bihar | 10 |
| 8 | Sonoo | UP | 12 |

**DEPARTMENT**

| DEP_ID | DEP_NAME |
|---|---|
| 10 | Math |
| 11 | English |
| 12 | Java |
| 13 | Physics |
| 14 | Civil |
| 15 | Chemistry |

Cluster Key



| DEP_ID | DEP_NAME | EMP_ID | EMP_NAME | ADDRESS |
|---|---|---|---|---|
| 10 | Math | 7 | Amy | Bihar |
| 11 | English | 4 | Amelia | Meerut |
| 12 | Java | 2 | Robert | Gujarat |
| 12 | | 8 | Sonoo | UP |
| 13 | Physics | 6 | Jackson | Delhi |
| 14 | Civil | 1 | John | Delhi |
| 14 | | 5 | Kristen | Noida |
| 15 | Chemistry | 3 | David | Mumbai |

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

# Types of Cluster file organization:

Cluster file organization is of two types:

## 1. Indexed Clusters:

In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

## 2. Hash Clusters:

It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

# Pros of Cluster file organization

- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
- It provides the efficient result when there is a 1:M mapping between the tables.

# Cons of Cluster file organization

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.

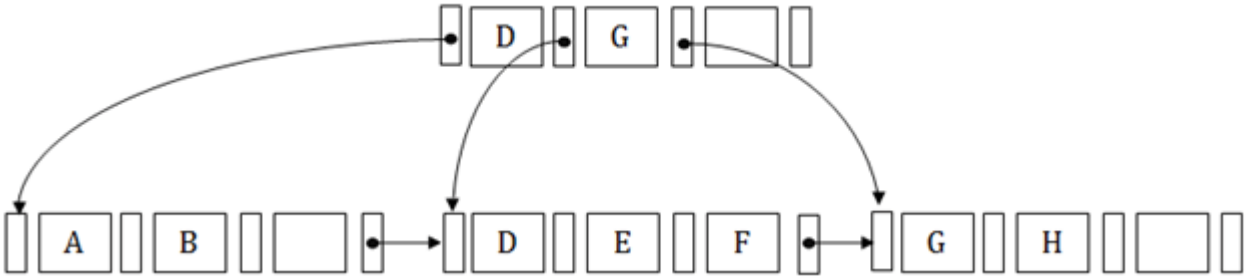
# B+ Tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.

- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

## Structure of B+ Tree

- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.




## Internal node

- An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
- At most, an internal node of the tree contains n pointers.

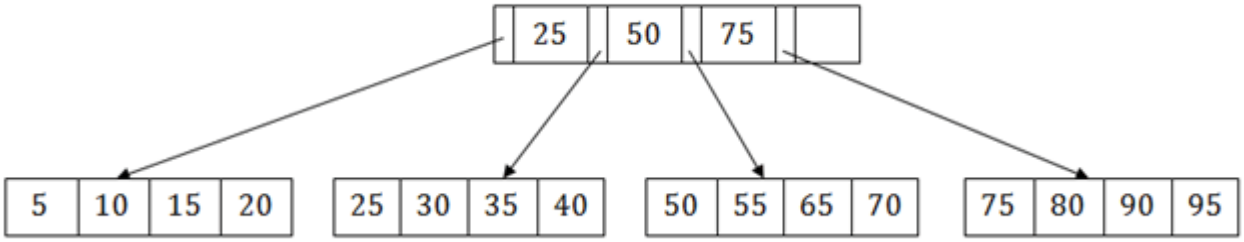
## Leaf node

- The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
- At most, a leaf node contains n record pointer and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

# Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

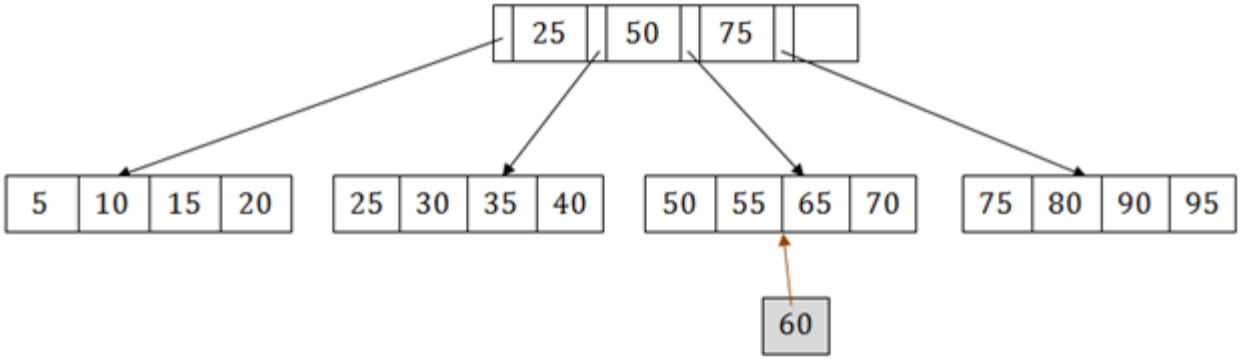
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.




# B+ Tree Insertion

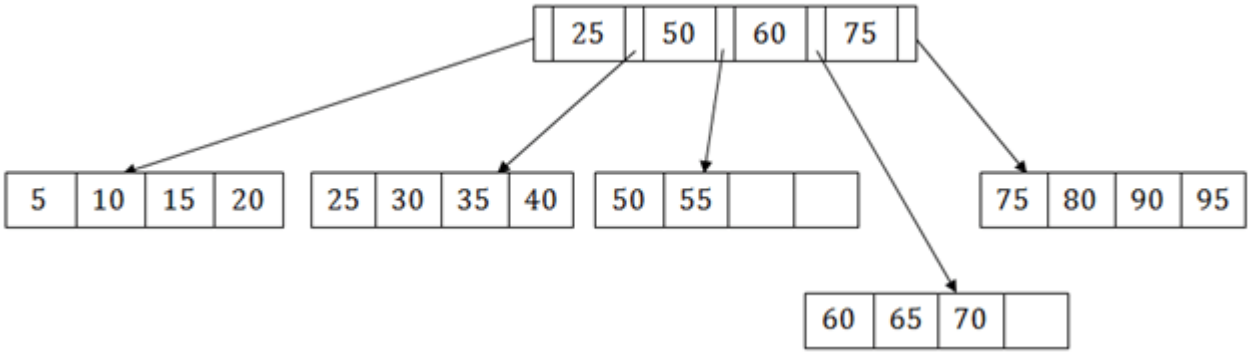
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.




The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

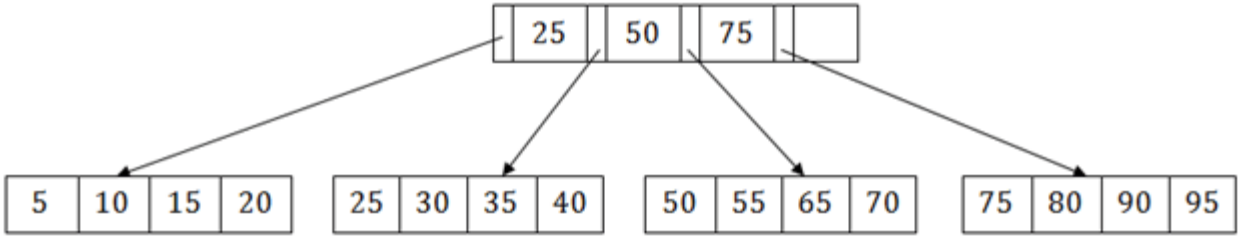



This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

# B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:

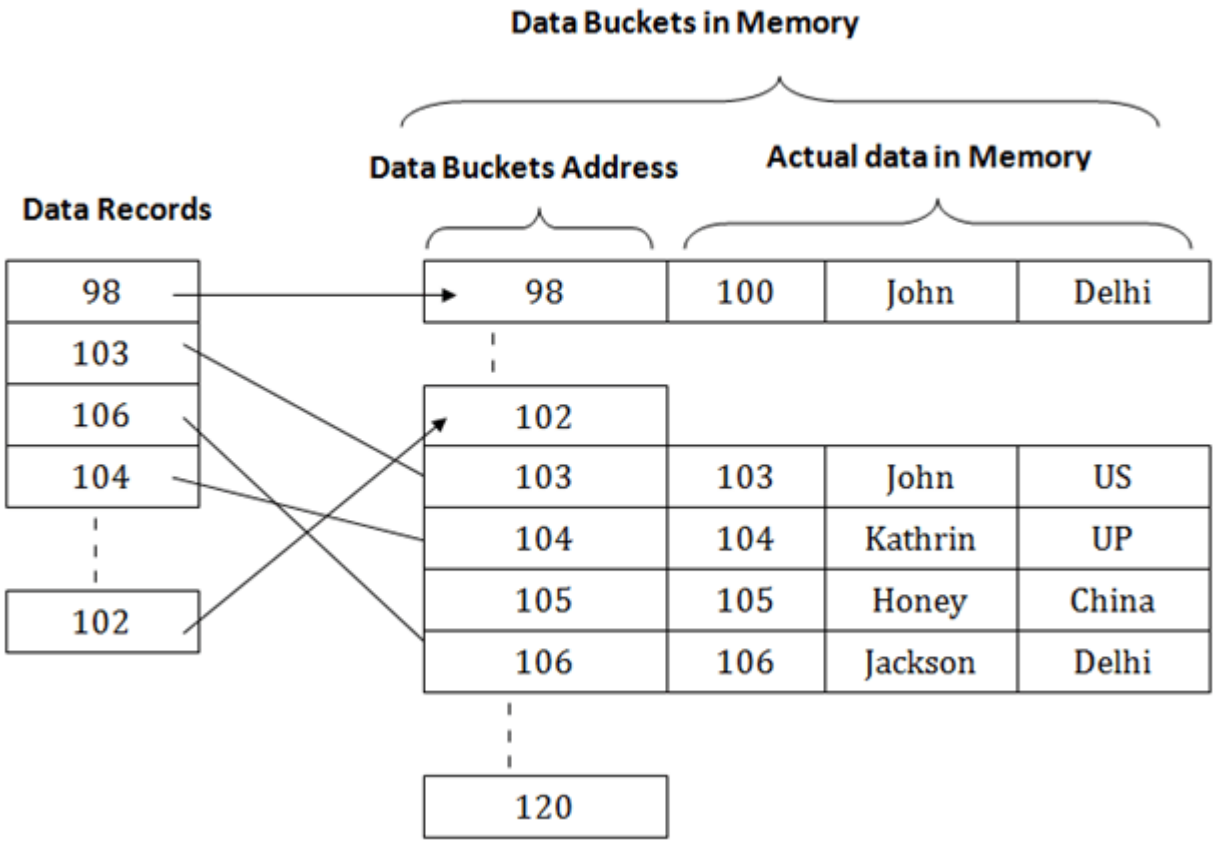



# Hashing

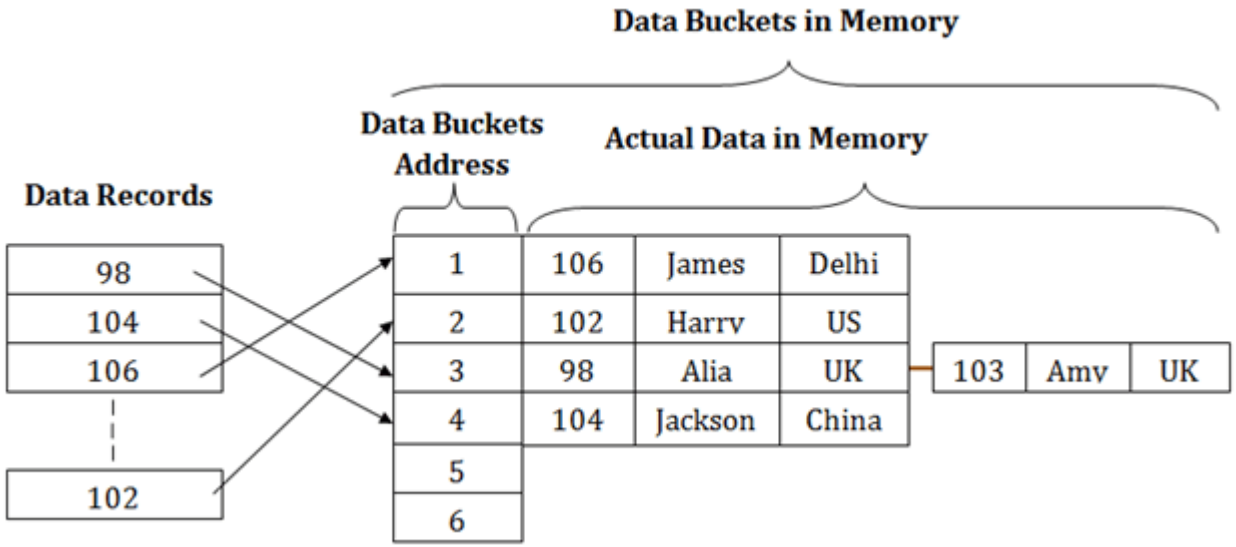
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

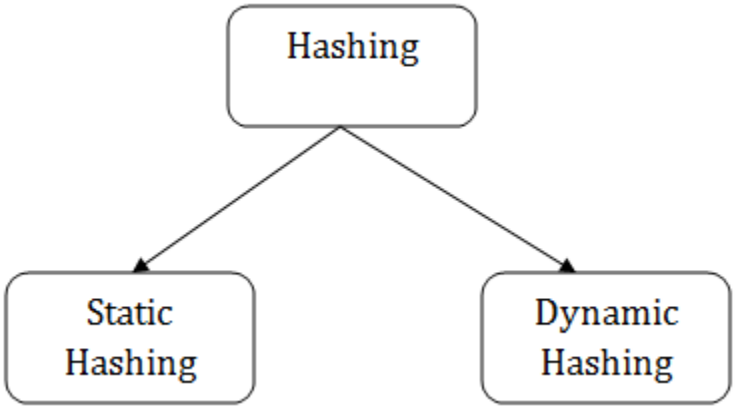
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.




The above diagram shows data block addresses same as primary key value. This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.




## Types of Hashing:



- Static Hashing
- Dynamic Hashing

## What is Data Structure?

- Data structure is an arrangement of data in computer's memory. It makes the data quickly available to the processor for required operations.
- It is a software artifact which allows data to be stored, organized and accessed.
- It is a structure program used to store ordered data, so that various operations can be performed on it easily.
  **For example,** if we have an employee's data like name 'ABC' and salary 10000. Here, 'ABC' is of String data type and 10000 is of Float data type.
  We can organize this data as a record like Employee record and collect & store employee's records in a file or database as a data structure like 'ABC' 10000, 'PQR' 15000, 'STU' 5000.
- Data structure is about providing data elements in terms of some relationship for better organization and storage.
- It is a specialized format for organizing and storing data that can be accessed within appropriate ways.

## Why is Data Structure important?

- Data structure is important because it is used in almost every program or software system.
- It helps to write efficient code, structures the code and solve problems.
- Data can be maintained more easily by encouraging a better design or implementation.

- Data structure is just a container for the data that is used to store, manipulate and arrange. It can be processed by algorithms.

  **For example,** while using a shopping website like Flipkart or Amazon, the users know their last orders and can track them. The orders are stored in a database as records.

  However, when the program needs them so that it can pass the data somewhere else (such as to a warehouse) or display it to the user, it loads the data in some form of data structure.

## Types of Data Structure

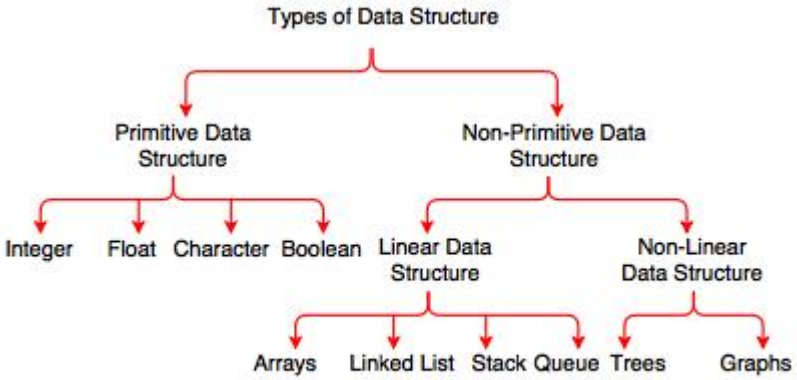



Fig. Types of Data Structure

### A. Primitive Data Type

- Primitive data types are the data types available in most of the programming languages.
- These data types are used to represent single value.
- It is a basic data type available in most of the programming language.

| Data type | Description |
|---|---|

| | |
|---|---|
| Integer | Used to represent a number without decimal point. |
| Float | Used to represent a number with decimal point. |
| Character | Used to represent single character. |
| Boolean | Used to represent logical values either true or false. |

## B. Non-Primitive Data Type

- Data type derived from primary data types are known as Non-Primitive data types.
- Non-Primitive data types are used to store group of values.

**It can be divided into two types:**

1. Linear Data Structure
2. Non-Linear Data Structure

### 1. Linear Data Structure

- Linear data structure traverses the data elements sequentially.
- In linear data structure, only one data element can directly be reached.
- It includes array, linked list, stack and queues.

| Types | Description |
|---|---|
| Arrays | Array is a collection of elements. It is used in mathematical problems like matrix, algebra etc. each element of an array is referenced by a subscripted variable or value, called subscript or index enclosed in parenthesis. |
| Linked list | Linked list is a collection of data elements. It consists of two parts: Info and Link. Info gives information and Link is an address of next node. Linked list can be implemented by using pointers. |

| | |
|---|---|
| Stack | Stack is a list of elements. In stack, an element may be inserted or deleted at one end which is known as Top of the stack. It performs two operations: Push and Pop. Push means adding an element in stack and Pop means removing an element in stack. It is also called Last-in-First-out (LIFO). |
| Queue | Queue is a linear list of element. In queue, elements are added at one end called rear and the existing elements are deleted from other end called front. It is also called as First-in-First-out (FIFO). |

### 2. Non-Linear Data Structure

- Non-Linear data structure is opposite to linear data structure.
- In non-linear data structure, the data values are not arranged in order and a data item is connected to several other data items.
- It uses memory efficiently. Free contiguous memory is not required for allocating data items.
- It includes trees and graphs.

| Type | Description |
|---|---|
| Tree | Tree is a flexible, versatile and powerful non-linear data structure. It is used to represent data items processing hierarchical relationship between the grandfather and his children & grandchildren. It is an ideal data structure for representing hierarchical data. |
| Graph | Graph is a non-linear data structure which consists of a finite set of ordered pairs called edges. Graph is a set of elements connected by edges. Each elements are called a vertex and node. |

## Abstract Data type (ADT)

### What is ADT?

- ADT stands for **Abstract Data Type.**
- It is an abstraction of a data structure.

- Abstract data type is a mathematical model of a data structure.
- It describes a container which holds a finite number of objects where the objects may be associated through a given binary relationship.
- It is a logical description of how we view the data and the operations allowed without regard to how they will be implemented.
- ADT concerns only with what the data is representing and not with how it will eventually be constructed.
- It is a set of objects and operations. For example, List, Insert, Delete, Search, Sort.

**It consists of following three parts:**

1. Data
2. Operation
3. Error

**1. Data** describes the structure of the data used in the ADT.

**2. Operation** describes valid operations for the ADT. It describes its interface.

**3. Error** describes how to deal with the errors that can occur.

## Advantages of ADT

- ADT is reusable and ensures robust data structure.
- It reduces coding efforts.
- Encapsulation ensures that data cannot be corrupted.
- ADT is based on principles of Object Oriented Programming (OOP) and Software Engineering (SE).
- It specifies error conditions associated with operations.

# Introduction of B+ Tree

In order, to implement dynamic multilevel indexing, B-tree and B+ tree are generally employed. The drawback of B-tree used for indexing, however is that it stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in the node of a B-tree. This technique, greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.
B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree. Thus, the structure of leaf nodes of a B+ tree is quite different from the structure of internal nodes of the B+ tree. It may be noted here that, since data pointers are present only at the leaf nodes, the leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, in order to access them. Moreover, the leaf nodes are linked to provide ordered access to the records. The leaf nodes, therefore form the first level of index, with the internal nodes forming the other levels of a multilevel index. Some of the key values of the leaf nodes also appear in the internal nodes, to simply act as a medium to control the searching of a record.

From the above discussion it is apparent that a B+ tree, unlike a B-tree has two orders, ‘a’ and ‘b’, one for the internal nodes and the other for the external (or leaf) nodes.

**The structure of the internal nodes of a B+ tree of order ‘a’ is as follows:**

1. Each internal node is of the form :
   $<P_1, K_1, P_2, K_2, ....., P_{c-1}, K_{c-1}, P_c>$
   where c <= a and each **$P_i$ is a tree pointer (i.e points to another node of the tree)** and, each **$K_i$ is a key value** (see diagram-I for reference).
2. Every internal node has : $K_1 < K_2 < .... < K_{c-1}$
3. For each search field values ‘X’ in the sub-tree pointed at by $P_i$, the following condition holds :
   $K_{i-1} < X <= K_i$, for $1 < i < c$ and,

$K_{i-1} < X$, for i = c
(See diagram I for reference)

4. Each internal nodes has at most ‘a’ tree pointers.
5. The root node has, at least two tree pointers, while the other internal nodes have at least \ceil(a/2) tree pointers each.
6. If any internal node has ‘c’ pointers, c <= a, then it has 'c – 1' key values.

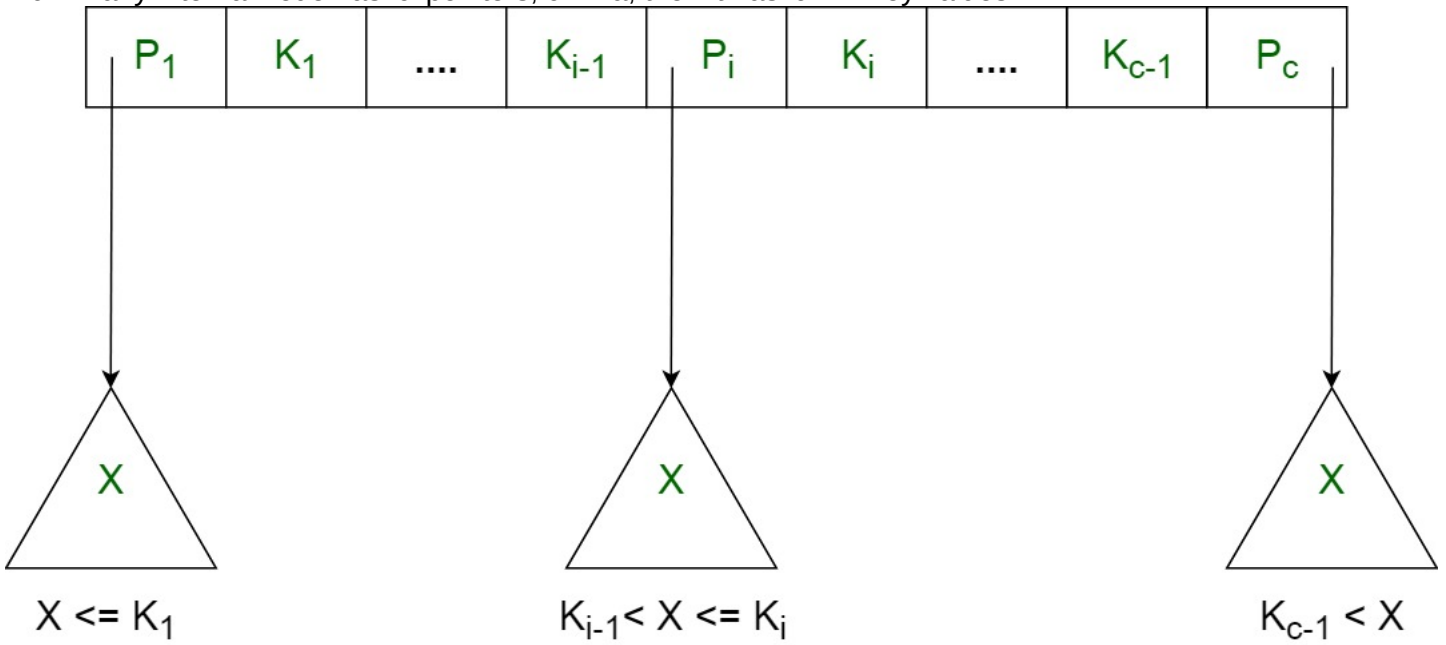



**Diagram-I**

**The structure of the leaf nodes of a B+ tree of order ‘b’ is as follows:**

1. Each leaf node is of the form :
   $<<K_1, D_1>, <K_2, D_2>, ....., <K_{c-1}, D_{c-1}>, P_{next}>$
   where c <= b and each **$D_i$ is a data pointer (i.e points to actual record in the disk whose key value is $K_i$ or to a disk file block containing that record)** and, each **$K_i$ is a key value** and, **$P_{next}$ points to next leaf node in the B+ tree** (see diagram II for reference).
2. Every leaf node has : $K_1 < K_2 < .... < K_{c-1}$, c <= b
3. Each leaf node has at least \ceil(b/2) values.
4. All leaf nodes are at same level.

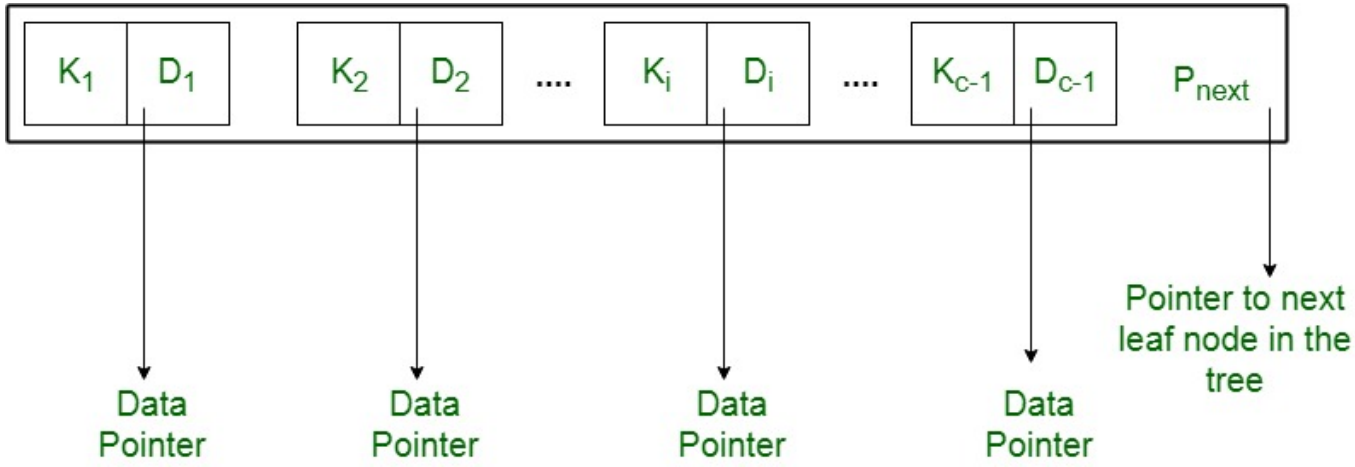
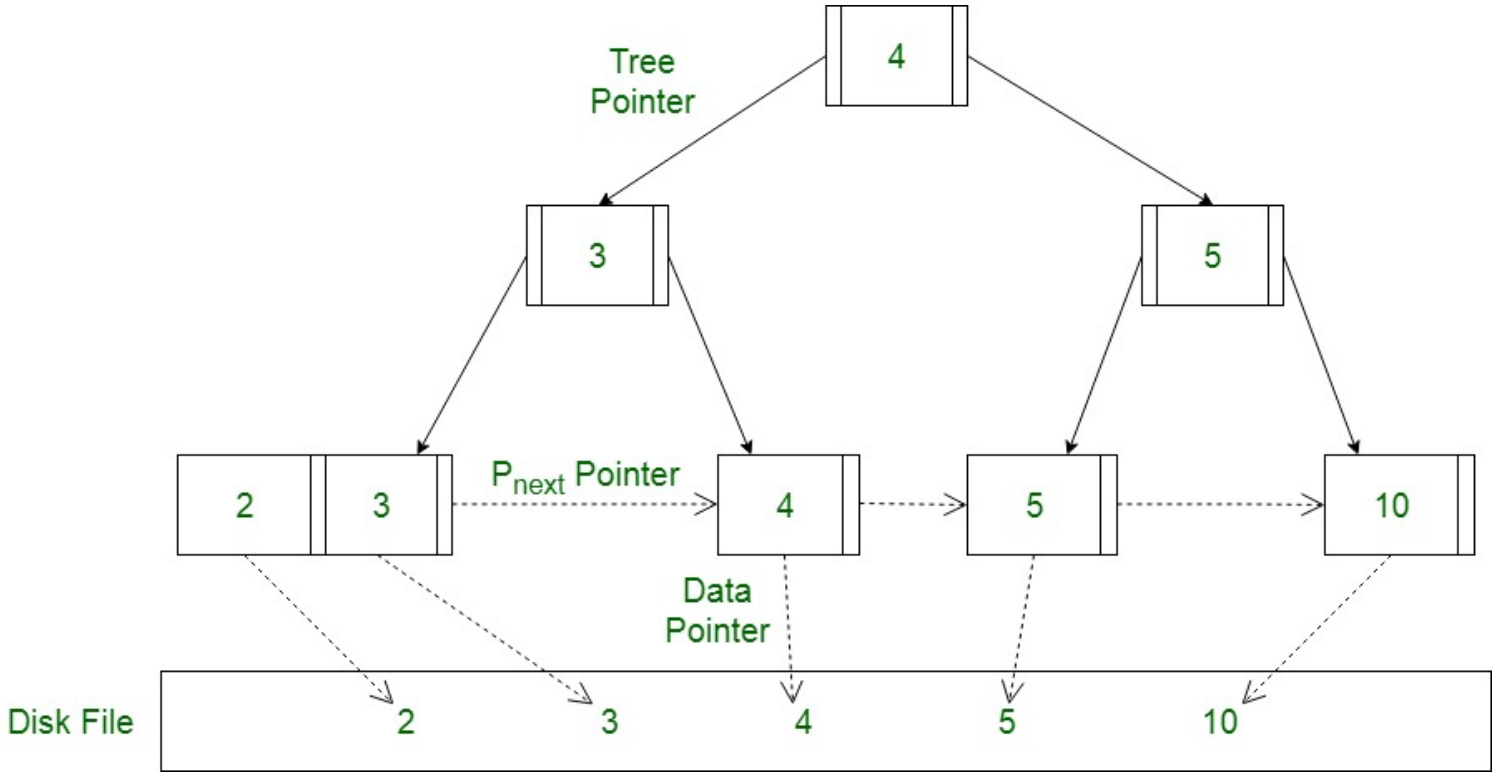



**Diagram-II**

Using the $P_{next}$ pointer it is viable to traverse all the leaf nodes, just like a linked list, thereby achieving ordered access to the records stored in the disk.

**A Diagram of B+ Tree –**




**Advantage –**
A B+ tree with 'l' levels can store more entries in its internal nodes compared to a B-tree having the same 'l' levels. This accentuates the significant improvement made to the search time for any given key. Having lesser levels and presence of $P_{next}$ pointers imply that B+ tree are very quick and efficient in accessing records from disks.

# BCAGE 203 (BDP)
# UNIT-04
# Programming Methodologies

Programming methodology deals with the analysis, design and implementation of programs.

When programs are developed to solve real-life problems like inventory management, payroll processing, student admissions, examination result processing, etc. they tend to be huge and complex. The approach to analyzing such complex problems, planning for software development and controlling the development process is called **programming methodology**.

### Types of Programming Methodologies

There are many types of programming methodologies prevalent among software developers –

### Procedural Programming

Problem is broken down into procedures, or blocks of code that perform one task each. All procedures taken together form the whole program. It is suitable only for small programs that have low level of complexity.

**Example** − For a calculator program that does addition, subtraction, multiplication, division, square root and comparison, each of these operations can be developed as separate procedures. In the main program each procedure would be invoked on the basis of user's choice.

### Object-oriented Programming

Here the solution revolves around entities or objects that are part of problem. The solution deals with how to store data related to the entities, how the entities behave and how they interact with each other to give a cohesive solution.

**Example** − If we have to develop a payroll management system, we will have entities like employees, salary structure, leave rules, etc. around which the solution must be built.

Some of the features of object oriented programming are:

• Emphasis is on data rather than procedure.

• Programs are divided into what are known as objects.

• Data structures are designed such that they characterize the objects.

• Functions that operate on the data of an object are ties together in the data

structure.

• Data is hidden and cannot be accessed by external function.

• Objects may communicate with each other through function.

• New data and functions can be easily added whenever necessary.

• Follows bottom up approach in program design.

**Functional Programming**

Here the problem, or the desired solution, is broken down into functional units. Each unit performs its own task and is self-sufficient. These units are then stitched together to form the complete solution.

**Example** − A payroll processing can have functional units like employee data maintenance, basic salary calculation, gross salary calculation, leave processing, loan repayment processing, etc.

**Logical Programming**

Here the problem is broken down into logical units rather than functional units. **Example:** In a school management system, users have very defined roles like class teacher, subject teacher, lab assistant, coordinator, academic in-charge, etc. So the software can be divided into units depending on user roles. Each user can have different interface, permissions, etc.

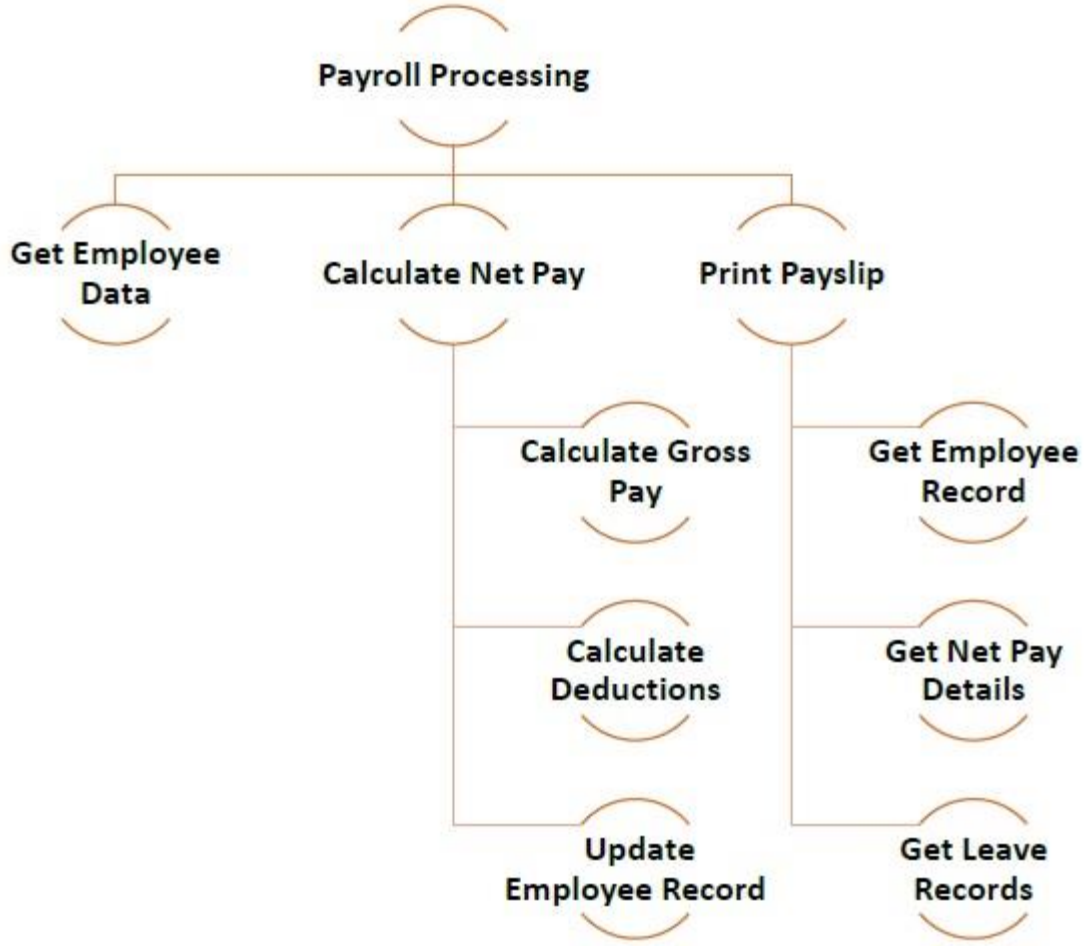
Software developers may choose one or a combination of more than one of these methodologies to develop a software. Note that in each of the methodologies discussed, problem has to be broken down into smaller units. To do this, developers use any of the following two approaches −

- Top-down approach
- Bottom-up approach

**Top-down or Modular Approach**

The problem is broken down into smaller units, which may be further broken down into even smaller units. Each unit is called a **module**. Each module is a self-sufficient unit that has everything necessary to perform its task.

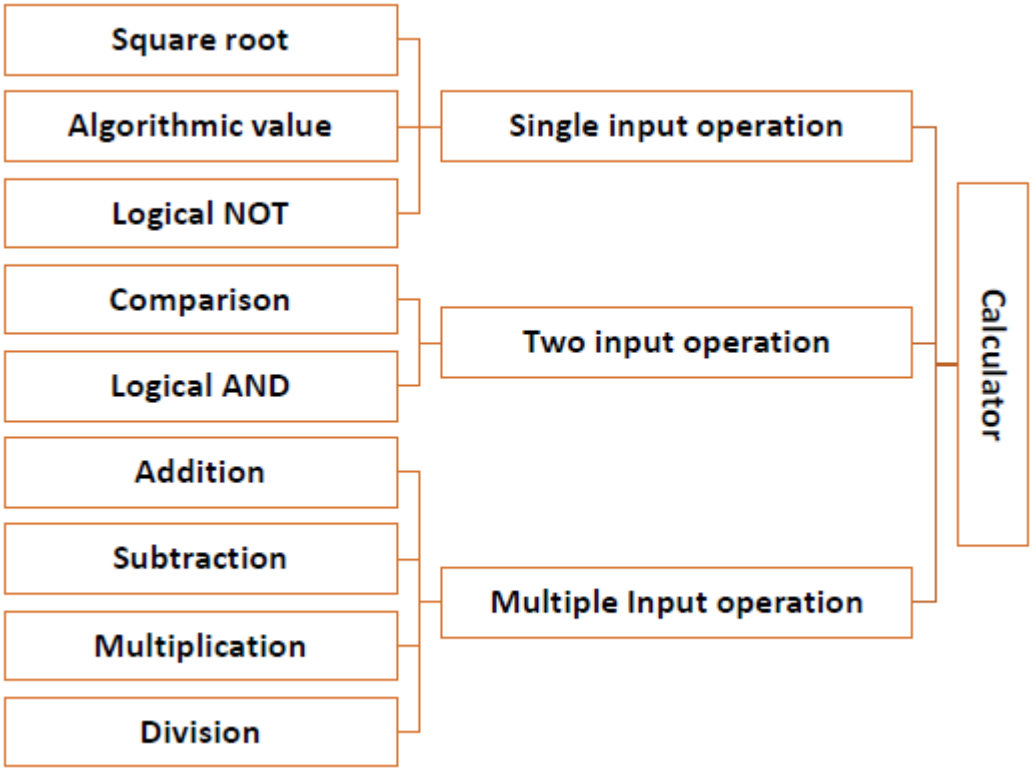
The following illustration shows an example of how you can follow modular approach to create different modules while developing a payroll processing program.




**Bottom-up Approach**

In bottom-up approach, system design starts with the lowest level of components, which are then interconnected to get higher level components. This process continues till a hierarchy of all system components is generated. However, in real-life scenario it is very difficult to know all lowest level components at the outset. So bottoms up approach is used only for very simple problems.

Let us look at the components of a calculator program.




**Structured Programming**

It is a programming style; and this style of programming is known by several names: Procedural decomposition, Structured programming, etc. Structured programming is not programming with structures but by using following types of code structures to write programs:

1. Sequence of sequentially executed statements

2. Conditional execution of statements (i.e., "if" statements)

3. Looping or iteration (i.e., "for, do...while, and while" statements)

4. Structured subroutine calls (i.e., functions)

In particular, the following language usage is forbidden:

• "GoTo" statements

• "Break" or "continue" out of the middle of loops

• Multiple exit points to a function/procedure/subroutine (i.e., multiple

"return" statements)

• Multiple entry points to a function/procedure/subroutine/method

In this style of programming there is a great risk that implementation details of many data structures have to be shared between functions, and thus globally exposed. This in turn tempts other functions to use these implementation details; thereby creating unwanted dependencies

between different parts of the program. The main disadvantage is that all decisions made from the start of the project depend directly or indirectly on the high-level specification of the application. It is a well known fact that this specification tends to change over a time. When that happens, there is a great risk that large parts of the application need to be rewritten.

**Advantages of structured programming**

1. clarity: structured programming has a clarity and logical pattern to their control structure and due to this tremendous increase in programming productivity

2. another key to structured programming is that each block of code has a single entry point and single exit point.so we can break up long sequence of code into modules

3. Maintenance: the clarity and modularity inherent in structured programming is of great help in finding an error and redesigning the required section of code.

**Programming Principles: -**

programming principles and using them in your code makes you a better developer. It improves the quality of code and later adding other functionality or making changes in it becomes easier for everyone. Let's discuss some basic principles of programming and the benefits of using it.

**7 Common Programming Principles**

**1. KISS:** Nobody in programming loves to debug, maintain, or make changes in complex code. *"**Keep It Simple, Stupid (KISS)**"* states that most systems work best if they are kept simple rather than making it complex, so when you are writing code your solution should not be complicated that takes a lot of time and effort to understand. If your code is simple then other developers won't face any problem understanding the code logic and they can easily proceed further with your code. So always try to simplify your code using different approaches like breaking a complex problem into smaller chunks or taking out some unnecessary code you have written.

**2. DRY:** Duplication of data, logic, or function in code not only makes your code lengthy but also wastes a lot of time when it comes to maintaining, debug or modify the code. If you need to make a small change in your code then you need to do it at several places. ***"Don't Repeat Yourself (DRY)"*** principal goal is to reduce the repetition of code. It states that a piece of code should be implemented in just one place in the source code. The opposite of the DRY principle is WET ("write everything twice" or "waste everyone's time") which breaks the DRY principle if you are writing the same logic at several places. You can create a common function or abstract your code to avoid the repetition in your code.

**3. YAGNI:** Your software or program can become larger and complex if you are writing some code which you may need in the future but not at the moment. ***"You Aren't Gonna Need It (YAGNI)"*** principle states that "don't implement something until it is necessary" because in most of the cases you are not going to use that piece of code in future. Most of the programmers while implementing software think about the future possibility and add some code or logic for some other features which they don't need at present. They add all the

unnecessary class and functionality which they might never use in the future. Doing this is completely wrong and you will eventually end up in writing bloated code also your project becomes complicated and difficult to maintain. We recommend all the programmers to avoid this mistake to save a lot of time and effort.

**4. SOLID:** The SOLID principle stands for five principles which are Single responsibility, Open-closed, Liskov substitution, Interface Segregation, and Dependency inversion. These principles are given by *Robert C. Martin* and you can check about these SOLID Principle in detail.

**5. Separation of Concerns (SoC):** Separation of Concerns Principle partition a complicated application into different sections or domains. Each section or domain addresses a separate concern or has a specific job. Each section is independent of each other and that's why each section can be tackled independently also it becomes easier to maintain, update, and reuse the code.

For example business logic (the content of the webpage) in an application is a different concern and user interface is a different concern in a web application program. One of the good examples of SoC is the MVC pattern where data ("model"), the logic ("controller"), and what the end-user sees ("view") divided into three different sections and each part is handled independently. Saving of data to a database has nothing to do with rendering the data on the web.

**6. Avoid Premature Optimization:** Optimization indeed helps in speeding up the program or algorithm but according to this principle you don't need to optimize your algorithm at an early stage of development. If you do premature optimization you won't be able to know where a program's bottlenecks will be and maintenance will become harder for you. If you optimize your code in the beginning and case if the requirement may change than your efforts will be wasted and your code will go to the garbage. So it's better to optimize the algorithm at the right time to get the right benefit of it.

**7. Law of Demeter:** This principle was first introduced by **Ian Holland** in 1987 at Northeastern University. It is also known as the *principle of least knowledge*. This principle divides the responsibility between classes or different units and it can be summarized in three points.

- Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.
- Each unit should only talk to its friends; don't talk to strangers.
- Only talk to your immediate friends.

The Law of Demeter helps in maintaining independent classes and makes your code less coupled which is very important in software development to make your application flexible, stable, maintainable and understandable.

**Coding Style**

Programmers spend an enormous amount of time reading and studying code when they are writing, testing, and debugging their programs. Using good programming style allows this process to proceed much more easily. When writing a program using iterative-enhancement, it is an excellent idea to beautify your code at the end of each enhancement, before proceding to the next one; each enhancement should be the best thatit can be, before continuing. Ultimately, this strategy will save you time compared to the strategy often used by students: ignore style until the program is completely written. This is a penny-wise, pound-foolish

strategy. It is much harder to "finish" a poorly-styled program, because it is harder to read and understand it; (software) engineers must learn to practice techniques that overcome human nature; this is one example.

In the real world, companies have their own style guidelines, which all their programmers must follow (see Vermuelen's book). In this way, code written by different programmers is consistent (and therefore more easily readable by other programmers). So, it is not unreasonable for me to ask you to write in a certain style, as consistently as you can.

We will use four general principles to discuss issues in programming style (backwards, they are the acronym CLAN).

- Names
- Alignment
- Locality
- Comments

**Good Names**

Programmers get to choose identifiers that name variables (and as we will see later in this course methods, parameters, classes, exceptions, etc). We should choose descriptive names. Yes, we should also try to choose short names, but descriptiveness is more important. A long descriptive name is better than a short unclear one. Of course, a short descriptive name is optimal.

Beginning programmers typically choose names that are too short: they abbreviate too much, or use just single letter names. Rather than declaring

**int qs; //qs means quarters**

declare a variable named **quarters** (and then, if necessary, comment on some other aspect of the name -like its units). Using longer names requires a bit more typing (which costs some time) and takes longer to read (ditto) but it makes it much easier for you to **understand** your program as you are enhancing/debugging it (which saves much much much more time). Examine the names that I use in my sample and solution programs and mimic them.

So far, we have learned the fllowing Java naming conventions.

- Names of Variable start with lower-case letters, and use upper-case letters at the start of each word in the name ("camel-style"): e.g., **dartsInCircle**.
- Names of classes start with upper-case letters,and use use other upper-case letters at the start of each word in the name: e.g., **StringTokenizer**.
- Names of **public static final fields are written in all upper-case, and use underscores to separate each word in the name: e.g., SPEED_OF_LIGHT.**

**Alignment (indenting)**

Generally, we use whitespace to present our programs (to humans, not computers) in an easy to read and understand form. Remember that adding extra whitespace doesn't affect the meaning of our programs (the sequence of tokens is still the same), but it does affect how a program is displayed in the editor while we are reading it.

Using extra whitespace will make the program "longer" but easier to read. In fact, in one early style of written English (scriptio continua), words were strung together with no intervening whitespace. Itwasstillreadablebutveryslowanddifficulttocomprehend. Sometimes smaller isn't simler.

Alignment involves mostly using horizontal whitespace. The most important use of alignment is showing which statements are controlled by which control structures, with the controlled statements indented to appear *inside* the statements that control them. This relationship is the essence of using control structures, so highlighting it is critical.

There is a pattern in how we write control structures. For example in the block after **main()**, all statements are indented at the same level.

```
{
  statement1
  statement2
  ...
  statementn
}
```

A typical indentation for these statements (and others inside control structures, illustrated below) is 2-4 spaces: one space is too little and more than four is too much (the Goldilocks principle again). In fact, the indent icons in the editor (red left-arrow followed by text or red right-arrow followed by text) make it easy to select multiple lines of text and indent (or outdent) them 2 spaces at a time.

Likewise, in an **if** statement we use the following forms (depending on whether or not the statement contolled by the **if** is a block)

```
if (test)
  statementT

if (test) {
  statementT1
  statementT2
  ...
  statementTn
}
```

For an **if/else** statement, there are four possiblities (based on the absence or presense of blocks). From simplest to most complicated, they are:

```
if (test)
  statementT
```

```
else
  statement_F


if (test)
  statement_T
else{
  statement_F1
  ...
  statement_Fn
}


if (test) {
  statement_T1
  ...
  statement_Tn
}else
  statement_F


if (test) {
  statement_T1
  ...
  statement_Tn
}else{
  statement_F1
  ...
  statement_Fn
}
```

I like to write **}else{** on the same line, but Vermeulen likes to write

```
 }
 else{
```

Many programmers adopt a style that ALWAYS use blocks in **if** statements (and loops), even if they contain just ONE statement. On the positive side, such an approach makes it very easy to add/remove statements (when debugging/enhancing programs), because the block is already there; otherwise going from one to more statements requires adding a block, and going from multiple to one statement requires removing the block. On the negative side, blocks, when they are unneccessary, make the program a harder to read. So, choose whichever of these options you think is better, but be consistent with your choice. I like "blocks where necessary" but Vermeulen likes "always blocks".

Finally, identically to **if** statements, we align a **for** loop by indenting the statement that is their body.

```
for (;;)
  statement
```

```
for (;;) {
  statement1
  statement2
  ...
  statementn
}
```

Almost all interesting loops use a block for their bodies. Exceptions are very simple loops and loops that have one **try-catch** statement in their bodies; such a statement has most of its code in a **try** block.

I cannot overemphasize how important it is to use proper alignment in control structures. A major source of programming errors for beginners is not understanding which statements are controlled by which control structures: these can get tricky with expression statements inside **if** statements inside loops. Proper alignment makes such relationships much simpler to see. I have seen students spend 2 hours trying to debug a program; at which point then finally spend 10 minutes aligning its statements (because I refuse to help them until they do), and then they solve their problem by themselves in 1 minute. If you expect to debug your programs, it is imperative that you use proper alignment whenever you add/remove code to/from them.

Another use of alignment occurs when declaring a sequence of variables; rather than doing so haphazardly, we can align the types, names, initial values, and comments.

```
int    game     = 0;   //Current game being played
int    maxGames = 10;  //Limit on games for one customer
int    winCount = 0;   //For statistics (see WL_Ratio too)
int    loseCount = 0;
double winLoseRatio;   //Calculated at the end of a session
```

Some programmers think that this kind of alignment is too much trouble, because if you add/remove declarations, you must realign them; I think the effort is worth it. So please examine all the alignment that I use in my sample and solution programs and mimic them.

**Locality (paragraphing)**

Locality is the most subjective of the style rules. It involves mostly adding extra vertical whitespace (blank lines). By grouping statements together and then placing blank lines between groups, we create the programming equivalent of paragraphs in prose writing (where each paragraph contains related sentences). In a written paper, students would never put all the sentences into one long paragraph; likewise, students would never make every sentence into its own paragraph. So, we should always use a more reasonable grouping (some number of related lines) for paragraphing in our programs.

Typically, each code group should contain a half-dozen statements. The magic number 7+/-2 is also used for psychological reasons: it represents the number of items typically usable in the brain's short-term memory.

Whenever a large number of statements appear in a block of code, use blank lines to group them into a smaller number of related sequences. We can write a preface comment (see below) that acts as a topic sentence for the paragraph of code.

A **for** loop and **try-catch** almost always start their own group; so do complicated **if** statements. Locality is more art than rules; I encourage you to examine the groupings that I use in my sample and solution programs and try to critique and ultimately emulate them.

---

**Comments**

We document our programs with comments. While we try to express ourselves as well as we can with Java code, there is always other useful information about a program that we would like communicate to programmers reading our code (including ourself, while debugging it, or at some future date when we are enhancing our code). Such information is for programmers, not the computer: not the instructions saying HOW the code works (that is for the programmer and computer), but WHAT the program does and WHY it does it (that way). We supply this information in comments.

There are a few different categories of comments that frequently reappear.

- Preface comments act as a topic-sentence, describing a group of related statements that directly follow the comment. Use the locality principle with such comments: there should be more blank lines separating the comment from the code before it (which it doesn't describe) than blank lines separating the comment from the code after it (which it does describe). Taken together, and indented appropriately, these commments provide an outline of the program. Every loop should have a preface comment; for other statements, comment them as necessary.
- Sidebar comments appear on the same line, after some statement. They help explain that statement; sometimes a series of sidebar comments will also help outline the computation. Use alignment so that all the sidebar comments are aligned: that makes it very easy to have the code separated from the comments (more use of the locality rule).
- "Sandwhich comments" directly preface and suffix some statement (with no blank lines lines. Use a sandwhich comment to make the **if/break;** statements terminating a long loop easy to locate.
- ```
  //////////////////////
  if (index == maxIndex)
    break;
  //////////////////////
  ```

- Avoid mingling comments within code; separate them for clarity. In the following example use the FORMER side-bar comment, not the latter, code.
- ```
  d = v*t;  //distance = velocity times time
  ```

- **d /*distance*/ = v /*velocity*/ * t /*time*/;**

Like the other rules of good style, comments are best included while the program is being written, not after it is working. I find and correct many errors while writing comments, because I am focusing on the code while writing about it. Again, many students approach writing comments as something to do AFTER the program is complete, which ultimately slows them down. Examine the comments that I use in my sample and solution programs and try to critique and ultimately emulate them.

**Miscellaneous Style Rules**

Finally, here are some miscellaneous style rules

- Use local variables whenever they clarify the code, keeping expression sizes managable; use the goldilocks principle
- Don't reuse variable names for more than one purpose.
- Choose the types for variables carefully. If a variable stores only integral values, declare it to be an **int**; use explicit conversion if you need to use it as a **double** in some expression(s).
- Initialize variables when they are declared; but don't initialize them at all if the next use of the variable is to store something into it.
- Use about 80 characters per line; remember that a carriage return is whitespace, so don't write huge lines of code.
- Good style is cumulative: each style improvement may marginally improve a program; but many can dramatically improve it.

Write code to be easily readable and understandable. Don't obfuscate code because you think it will make the code run faster. Compilers do amazing optimizations.

## DBMS Concept:-

A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.

Database management systems are set up on specific data handling concepts, as the practice of administrating a database evolves. The earliest databases only handled individual single pieces of specially formatted data. Today’s more evolved systems can handle different kinds of less formatted data and tie them together in more elaborate ways.

**Database Management System** or **DBMS** in short refers to the technology of storing and retrieving usersí data with utmost efficiency along with appropriate security measures. This tutorial explains the basics of DBMS such as its architecture, data models, data schemas, data independence, E-R model, relation model, relational database design, and storage and file structure and much more.

## Why to Learn DBMS?

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics −

- **Real-world entity** − A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables** − DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Isolation of data and application** − A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- **Less redundancy** − DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.
- **Consistency** − Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.
- **Query Language** − DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.

## Applications of DBMS

**Database** is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. Following are the important characteristics and applications of DBMS.

- **ACID Properties** − DBMS follows the concepts of **A**tomicity, **C**onsistency, **I**solation, and **D**urability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- **Multiuser and Concurrent Access** − DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.

- **Multiple views** − DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security** − Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code

**What is a Relational Database (RDBMS)?**

A relational database is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

**A relational database example:-**

Here's a simple example of two tables a small business might use to process orders for its products. The first table is a customer info table, so each record includes a customer's name, address, shipping and billing information, phone number, and other contact information. Each bit of information (each attribute) is in its own column, and the database assigns a unique ID (a key) to each row. In the second table—a customer order table—each record includes the ID of the customer that placed the order, the product ordered, the quantity, the selected size and color, and so on—but not the customer's name or contact information.

These two tables have only one thing in common: the ID column (the key). But because of that common column, the relational database can create a relationship between the two tables. Then, when the company's order processing application submits an order to the database, the database can go to the customer order table, pull the correct information about the product order, and use the customer ID from that table to look up the customer's billing and shipping information in the customer info table. The warehouse can then pull the correct product, the customer can receive timely delivery of the order, and the company can get paid.

**Differences Between RDBMS and DBMS**

There are some contrasting differences between RDBMS vs. DBMS. An RDBMS is an advanced version of a DBMS. Unlike a DBMS that manages databases on a computer network and hard disks, an RDBMS database helps maintain relationships between its tables.

Here are some of the main differences between an RDBMS and a DBMS:

- **Number of operators:** A DBMS allows only a single operator simultaneously, whereas multiple users can operate an RDBMS concurrently. An RDBMS uses intricate algorithms that enable several users to access the database while preserving data integrity simultaneously, significantly reducing response time.
- **Hardware and software need:** A DBMS utilizes fewer data storage and retrieval resources than an RDBMS. The latter is more complex due to its multi-table structure and cross-referencing capability, making it costlier than a DBMS. RDBMSs are also generally used for enterprise-class applications, while DBMSs are more commonly utilized for smaller, purpose-specific applications.
- **Data modification:** Altering data in a DBMS is quite difficult, whereas you can easily modify data in an RDBMS using an SQL query. Thus, programmers can change/access multiple data elements simultaneously. This is one of the reasons why an RDBMS is more efficient than a DBMS.
- **Data volume:** A DBMS is more suitable for handling low data volume, whereas an RDBMS can handle even large data volumes.
- **Keys and Indexes:** A DBMS doesn't involve keys and indexes, whereas an RDBMS specifies a relationship between data elements via keys and indexes.
- **Data consistency:** As a DBMS does not follow the ACID (Atomicity, Consistency, Isolation, and Durability) model, the data stored can have inconsistencies. In contrast, an RDBMS follows the ACID model, making it structured and consistent.
- **Database structure:** A DBMS works by storing data in a hierarchical structure, while an RDBMS stores data in tables.
- **Data fetching speed:** In a DBMS, the process is relatively slow, especially when data is complex and extensive. This is because each of the data elements must be fetched individually. In an RDBMS, data is fetched faster because of the relational approach. Plus, SQL facilitates quicker data retrieval in an RDBMS.
- **Distributed databases:** A DBMS doesn't support distributed databases, whereas an RDBMS offers full support for distributed databases.
- **Client-server architecture:** Unlike a DBMS, an RDBMS supports client-server architecture.

**BCAGE 203**
**Unit - 05**

# Data Models:-

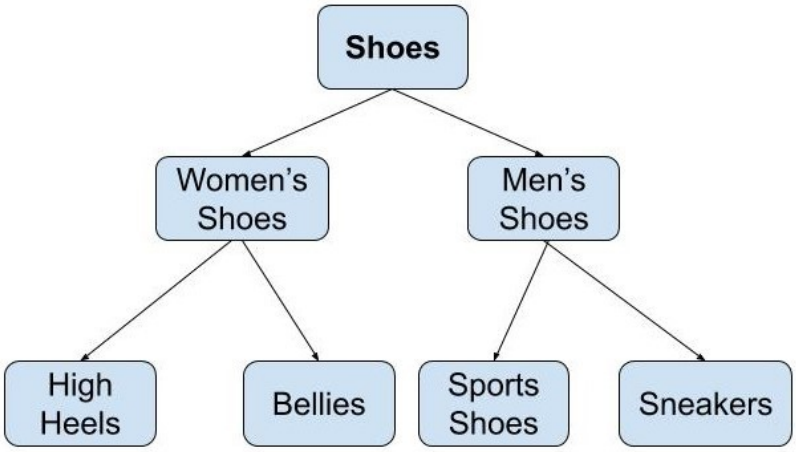
Data Model gives us an idea that how the final system will look like after its complete implementation. It defines the data elements and the relationships between the data elements. Data Models are used to show how data is stored, connected, accessed and updated in the database management system. Here, we use a set of symbols and text to represent the information so that members of the organisation can communicate and understand it. Though there are many data models being used nowadays but the Relational model is the most widely used model. Apart from the Relational model, there are many other types of data models about which we will study in details in this blog. Some of the Data Models in DBMS are:

1. Hierarchical Model
2. Network Model
3. Entity-Relationship Model
4. Relational Model
5. Object-Oriented Data Model
6. Object-Relational Data Model
7. Flat Data Model
8. Semi-Structured Data Model
9. Associative Data Model
10. Context Data Model

## 1. Hierarchical Model

Hierarchical Model was the first DBMS model. This model organises the data in the hierarchical tree structure. The hierarchy starts from the root which has root data and then it expands in the form of a tree adding child node to the parent node. This model easily

represents some of the real-world relationships like food recipes, sitemap of a website etc. ***Example:*** We can represent the relationship between the shoes present on a shopping website in the following way:




*Hierarchical Model*

**Features of a Hierarchical Model**

1. **One-to-many relationship:** The data here is organised in a tree-like structure where the one-to-many relationship is between the datatypes. Also, there can be only one path from parent to any node. **Example:** In the above example, if we want to go to the node sneakers we only have one path to reach there i.e through men's shoes node.
2. **Parent-Child Relationship:** Each child node has a parent node but a parent node can have more than one child node. Multiple parents are not allowed.
3. **Deletion Problem:** If a parent node is deleted then the child node is automatically deleted.
4. **Pointers:** Pointers are used to link the parent node with the child node and are used to navigate between the stored data. Example: In the above example the 'shoes' node points to the two other nodes 'women shoes' node and 'men's shoes' node.

**Advantages of Hierarchical Model**

- It is very simple and fast to traverse through a tree-like structure.

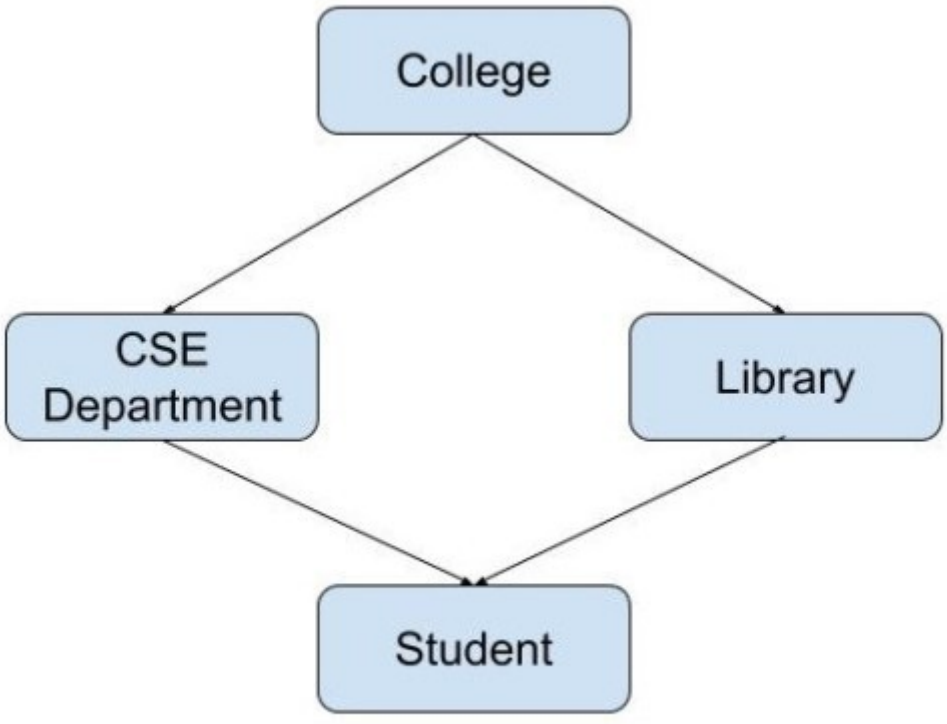
- Any change in the parent node is automatically reflected in the child node so, the integrity of data is maintained.

**Disadvantages of Hierarchical Model**

- Complex relationships are not supported.
- As it does not support more than one parent of the child node so if we have some complex relationship where a child node needs to have two parent node then that can't be represented using this model.
- If a parent node is deleted then the child node is automatically deleted.

2. **Network Model**

This model is an extension of the hierarchical model. It was the most popular model before the relational model. This model is the same as the hierarchical model, the only difference is that a record can have more than one parent. It replaces the hierarchical tree with a graph. **Example:** In the example below we can see that node student has two parents i.e. CSE Department and Library. This was earlier not possible in the hierarchical model.




*Network Model*

**Features of a Network Model**

1. **Ability to Merge more Relationships:** In this model, as there are more relationships so data is more related. This model has the ability to manage one-to-one relationships as well as many-to-many relationships.
2. **Many paths:** As there are more relationships so there can be more than one path to the same record. This makes data access fast and simple.
3. **Circular Linked List:** The operations on the network model are done with the help of the circular linked list. The current position is maintained with the help of a program and this position navigates through the records according to the relationship.

**Advantages of Network Model**

- The data can be accessed faster as compared to the hierarchical model. This is because the data is more related in the network model and there can be more than one path to reach a particular node. So the data can be accessed in many ways.
- As there is a parent-child relationship so data integrity is present. Any change in parent record is reflected in the child record.

D**isadvantages of Network Model**

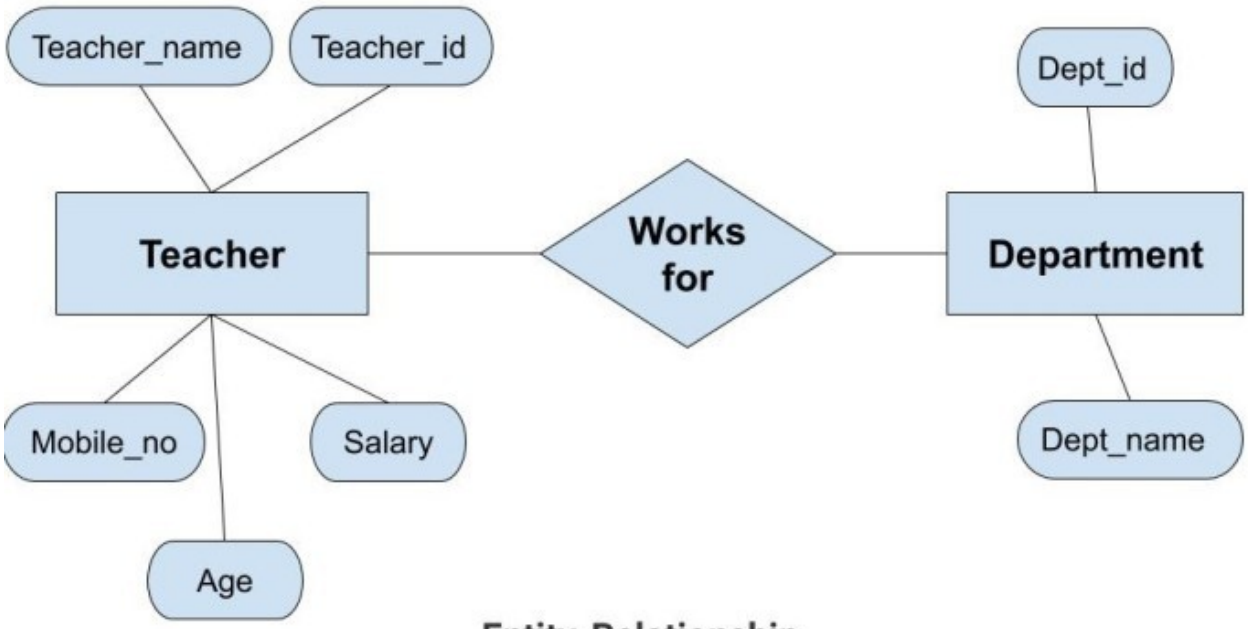
- As more and more relationships need to be handled the system might get complex. So, a user must be having detailed knowledge of the model to work with the model.
- Any change like updation, deletion, insertion is very complex.

**3. Entity-Relationship Model**

Entity-Relationship Model or simply ER Model is a high-level data model diagram. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:

- **Entities:** Entity is a real-world thing. It can be a person, place, or even a concept. Example: Teachers, Students, Course, Building, Department, etc are some of the entities of a School Management System.
- **Attributes:** An entity contains a real-world property called attribute. This is the characteristics of that attribute. Example: The entity teacher has the property like teacher id, salary, age, etc.
- **Relationship:** Relationship tells how two attributes are related. Example: Teacher works for a department.

**Example:**




Entity-Relationship Model

In the above diagram, the entities are Teacher and Department. The attributes of ***Teacher*** entity are Teacher_Name, Teacher_id, Age, Salary, Mobile_Number. The attributes of entity ***Department*** entity are Dept_id, Dept_name. The two entities are connected using the relationship. Here, each teacher works for a department.

*Features of ER Model*

- ***Graphical Representation for Better Understanding:*** It is very easy and simple to understand so it can be used by the developers to communicate with the stakeholders.
- ***ER Diagram:*** ER diagram is used as a visual tool for representing the model.
- ***Database Design:*** This model helps the database designers to build the database and is widely used in database design.

*Advantages of ER Model*

- ***Simple:*** Conceptually ER Model is very easy to build. If we know the relationship between the attributes and the entities we can easily build the ER Diagram for the model.
- ***Effective Communication Tool***: This model is used widely by the database designers for communicating their ideas.
- ***Easy Conversion to any Model***: This model maps well to the relational model and can be easily converted relational model by converting the ER model to the table. This model can also be converted to any other model like network model, hierarchical model etc.

*Disadvatages of ER Model*

- ***No industry standard for notation:*** There is no industry standard for developing an ER model. So one developer might use notations which are not understood by other developers.
- ***Hidden information:*** Some information might be lost or hidden in the ER model. As it is a high-level view so there are chances that some details of information might be hidden.

4. **Relational Model**

Relational Model is the most widely used model. In this model, the data is maintained in the form of a two-dimensional table. All the information is stored in the form of row and columns. The basic structure of a relational model is tables. So, the tables are also

called *relations* in the relational model. ***Example:*** In this example, we have an Employee table.

| Emp_id | Emp_name | Job_name | Salary | Mobile_no | Dep_id | Project_id |
|---|---|---|---|---|---|---|
| AfterA001 | John | Engineer | 100000 | 9111037890 | 2 | 99 |
| AfterA002 | Adam | Analyst | 50000 | 9587569214 | 3 | 100 |
| AfterA003 | Kande | Manager | 890000 | 7895212355 | 2 | 65 |

**EMPLOYEE TABLE**

**Features of Relational Model**

- **Tuples**: Each row in the table is called tuple. A row contains all the information about any instance of the object. In the above example, each row has all the information about any specific individual like the first row has information about John.
- **Attribute or field:** Attributes are the property which defines the table or relation. The values of the attribute should be from the same domain. In the above example, we have different attributes of the employee like Salary, Mobile_no, etc.

**Advantages of Relational Model**

- **Simple:** This model is more simple as compared to the network and hierarchical model.
- **Scalable:** This model can be easily scaled as we can add as many rows and columns we want.
- **Structural Independence:** We can make changes in database structure without changing the way to access the data. When we can make changes to the database structure without affecting the capability to DBMS to access the data we can say that structural independence has been achieved.

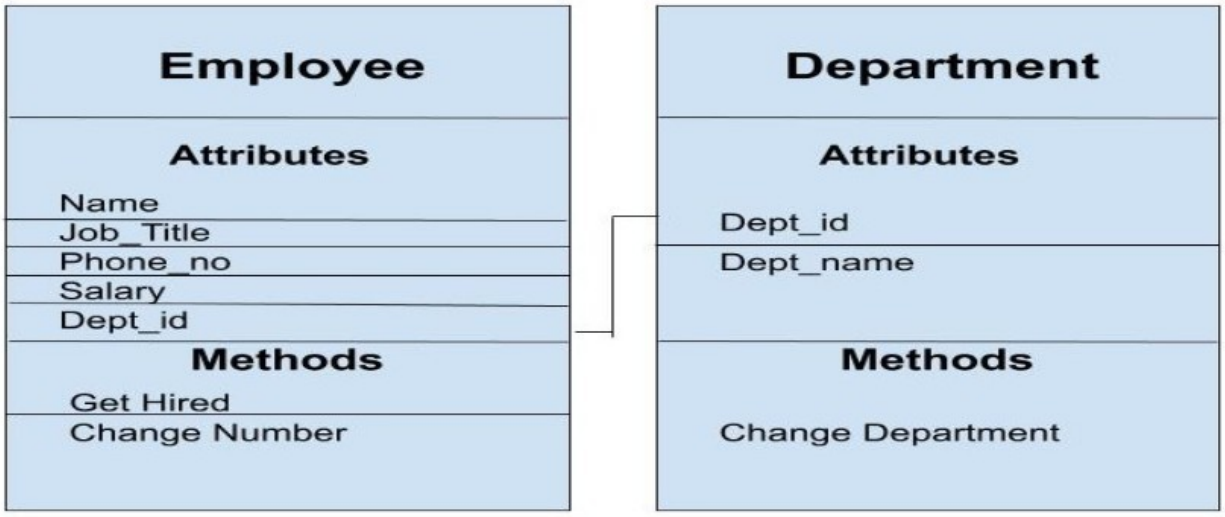
**Disadvantages of Relational Model**

- **Hardware Overheads:** For hiding the complexities and making things easier for the user this model requires more powerful hardware computers and data storage devices.
- **Bad Design:** As the relational model is very easy to design and use. So the users don't need to know how the data is stored in order to access it. This ease of design can lead to the development of a poor database which would slow down if the database grows.

But all these disadvantages are minor as compared to the advantages of the relational model. These problems can be avoided with the help of proper implementation and organisation.

## 5. Object-Oriented Data Model

The real-world problems are more closely represented through the object-oriented data model. In this model, both the data and relationship are present in a single structure known as an object. We can store audio, video, images, etc in the database which was not possible in the relational model(although you can store audio and video in relational database, it is adviced not to store in the relational database). In this model, two are more objects are connected through links. We use this link to relate one object to other objects. This can be understood by the example given below.




**Object_Oriented_Model**

6. **Object-Relational Model**

As the name suggests it is a combination of both the relational model and the object-oriented model. This model was built to fill the gap between object-oriented model and the relational model. We can have many advanced features like we can make complex data types according to our requirements using the existing data types. The problem with this model is that this can get complex and difficult to handle. So, proper understanding of this model is required.

7. **Flat Data Model**

It is a simple model in which the database is represented as a table consisting of rows and columns. To access any data, the computer has to read the entire table. This makes the modes slow and inefficient.

8. **Semi-Structured Model**

Semi-structured model is an evolved form of the relational model. We cannot differentiate between data and schema in this model. **Example:** Web-Based data sources which we can't differentiate between the schema and data of the website. In this model, some entities may have missing attributes while others may have an extra attribute. This model gives flexibility in storing the data. It also gives flexibility to the attributes. **Example:** If we are storing any value in any attribute then that value can be either atomic value or a collection of values.

9. **Associative Data Model**

Associative Data Model is a model in which the data is divided into two parts. Everything which has independent existence is called as an entity and the relationship among these entities are called association. The data divided into two parts are called items and links.

- **Item**: Items contain the name and the identifier(some numeric value).
- **Links:** Links contain the identifier, source, verb and subject.

**Multidimensional model**

This is a variation of the relational model designed to facilitate improved analytical processing. While the relational model is optimized for online transaction processing (OLTP), this model is designed for online analytical processing (OLAP).

Each cell in a dimensional database contains data about the dimensions tracked by the database. Visually, it's like a collection of cubes, rather than two-dimensional tables.

**Semistructured model**

In this model, the structural data usually contained in the database schema is embedded with the data itself. Here the distinction between data and schema is vague at best. This model is useful for describing systems, such as certain Web-based data sources, which we treat as databases but cannot constrain with a schema. It's also useful for describing interactions between databases that don't adhere to the same schema.

**Context model**

This model can incorporate elements from other database models as needed. It cobbles together elements from object-oriented, semistructured, and network models.

**Associative model**

This model divides all the data points based on whether they describe an entity or an association. In this model, an entity is anything that exists independently, whereas an association is something that only exists in relation to something else.

The associative model structures the data into two sets:

- A set of items, each with a unique identifier, a name, and a type
- A set of links, each with a unique identifier and the unique identifiers of a source, verb, and target. The stored fact has to do with the source, and each of the three identifiers may refer either to a link or an item.

Other, less common database models include:

- Semantic model, which includes information about how the stored data relates to the real world
- XML database, which allows data to be specified and even stored in XML format
- Named graph
- Triplestore

**NoSQL database models**

In addition to the object database model, other non-SQL models have emerged in contrast to the relational model:

The **graph database model**, which is even more flexible than a network model, allowing any node to connect with any other.

The **multivalue model**, which breaks from the relational model by allowing attributes to contain a list of data rather than a single data point.

The **document model**, which is designed for storing and managing documents or semi-structured data, rather than atomic data.

**1. RDBMS :**

RDBMS stands for Relational Database Management System.

In this database management, the data is organized into the related tables. To access the database it uses Structured Query Language (SQL). This model is based on the mathematical theory of relational algebra and calculus. The original concept for the model is proposed by Dr. E.F. Codd in a 1970. After some time the model was classified by defining twelve rules which are known as **Codd's rule**. For any database to be relational database it must satisfy atleast 6 out of 12 Codd's rules. These 12 Codd's rule are as follows :

- Information Rule
- Guaranteed Access Rule
- Systematic Treatment Of Null Values
- Database Description Rule
- Comprehensive Data Sub-Language Rule
- View Updating Rule
- High Level Insert, Update and Delete
- Physical Data Independence
- Logical Data Independence
- Integrity Independence
- Distribution Independence
- Non Subversion Rule

**2. ORDBMS :**

ORDBMS stands for Object-Relational Database Management System.

It provides all the facilities of RDBMS with the additional support of object oriented concepts. The concept of classes, objects and inheritance are supported in this database. It is present in the ground level between the RDBMS and OODBMS. In this data can be manipulated by using any **query language.** It is complex because it has to take care of both Relational database concepts and as well as Object Oriented concepts. Some of the object related DBMS available in the market are as follows :

- IBM'S DB2 Universal Database system
- Informix's Universal server

**Difference between RDBMS and ORDBMS :**

| S.No. | RDBMS | ORDBMS |
|---|---|---|
| 1. | RDBMS is a Relational Database | ORDBMS is a Object Oriented Relational |

| S.No. | RDBMS | ORDBMS |
|---|---|---|
| | Management System based on the Relational model of data. | Database Management System based on the Relational as well as Object Oriented database model. |
| 2. | It follows table structure, it is simple to use and easy to understand. | It is same as RDBMS but it has some extra confusing extensions because of the Object Oriented concepts. |
| 3. | It has no extensibility and content. | It is only limited to the new data-types. |
| 4. | Since RDBMS is old so, it is very mature. | It is developing so it is immature in nature. |
| 5. | In this, there is extensive supply of tools and trained developers. | It can take the advances of RDBMS tools and developers. |
| 6. | It has poor support for Object-Oriented programming. | It supports the features of object-oriented programming. |
| 7. | It supports Structured Query Language (SQL). | It supports Object Query Language (OQL). |
| 8. | RDMS is used for traditional applications tasks such as data administration and data processing. | ORDMS is used for applications with complex objects. |
| 9. | It is capable of handling only simple data. | It is also capable of handling the complex data. |
| 10 | MS SQL server, MySQL, SQLite, MariaDB are examples of RDBMS. | PostgreSQL is an example of ORDBMS. |

The SQL **CREATE DATABASE** statement is used to create a new SQL database.

Syntax

The basic syntax of this CREATE DATABASE statement is as follows −

```
CREATE DATABASE DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

If you want to create a new database <testDB>, then the CREATE DATABASE statement would be as shown below −

```
SQL> CREATE DATABASE testDB;
```

Make sure you have the admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows −

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
| testDB             |
+--------------------+
7 rows in set (0.00 sec)
```

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL **CREATE TABLE** statement is used to create a new table.

Syntax

The basic syntax of the CREATE TABLE statement is as follows −

```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
   PRIMARY KEY( one or more columns )
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with the following example.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check the complete details at Create Table Using another Table.

Example

The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table −

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC** command as follows −

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Now, you have CUSTOMERS table available in your database which you can use to store the required information related to customers.

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows −

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example

The following statements would create six records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in the CUSTOMERS table as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Populate one table using another table

You can populate the data into a table through the select statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax −

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
```

```
+----+----------+-----+-----------+----------+
```

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View

A view can be updated under certain conditions which are given below −

- The SELECT clause may not contain the keyword DISTINCT.

- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW
   SET AGE = 35
   WHERE name = 'Ramesh';
```

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
   WHERE age = 22;
```

This would ultimately delete a row from the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

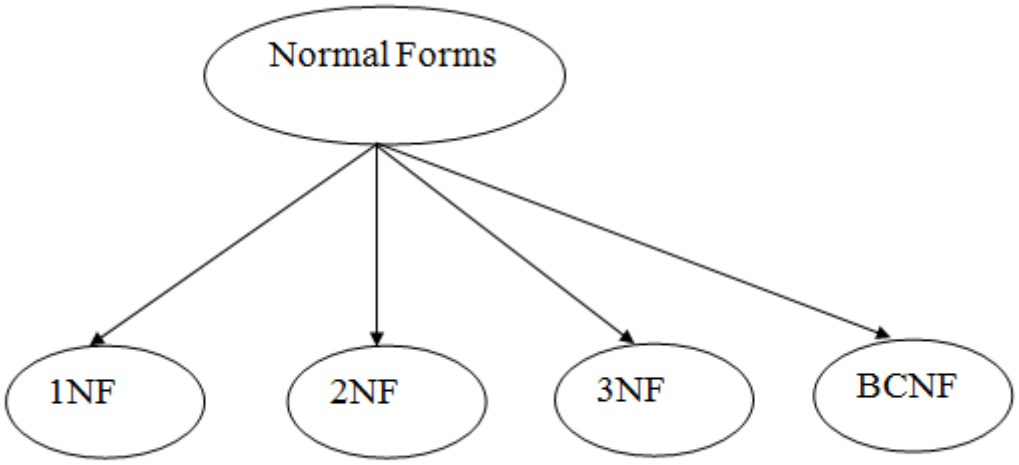
```
DROP VIEW CUSTOMERS_VIEW;
```

# Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

## Types of Normal Forms

There are the four types of normal forms:




| Normal Form | Description |
|---|---|
| 1NF | A relation is in 1NF if it contains an atomic (single) value. |
| 2NF | • The table should be in the 1NF.<br>• There should be no Partial Dependency. |
| 3NF | • The table should be in the 2NF.<br>• There should be no transitive Dependency. |
| BCNF<br>(3.5NF) | • It should be in the 3NF.<br>• And, for any dependency A → B, A should be a super key. |

**First Normal Form**

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

**EMPLOYEE table:**

| EMP_ID | EMP_NAME | EMP_PHONE | EMP_STATE |
|---|---|---|---|
| 14 | Mohan | 7272826385, 9064738238 | UP |
| 20 | Harish | 8574783832 | Bihar |
| 12 | Shyam | 7390372389, 8589830302 | Punjab |

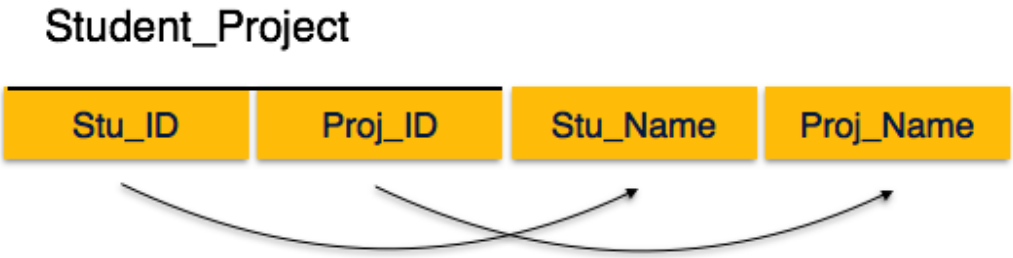
The decomposition of the EMPLOYEE table into 1NF has been shown below:

| EMP_ID | EMP_NAME | EMP_PHONE | EMP_STATE |
|---|---|---|---|
| 14 | Mohan | 7272826385 | UP |
| 14 | Mohan | 9064738238 | UP |
| 20 | Harish | 8574783832 | Bihar |
| 12 | Shyam | 7390372389 | Punjab |
| 12 | Shyam | 8589830302 | Punjab |

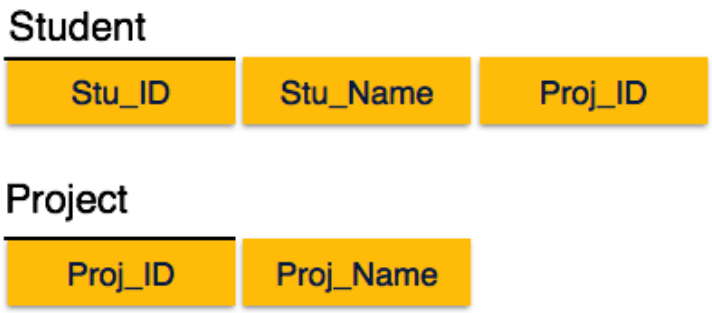
**Second Normal Form (2NF)**

- In the 2NF, relational must be in 1NF.
- In the second normal form, there should be no Partial Dependency. All non-key attributes should be fully functional dependent on the primary key

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.




We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.



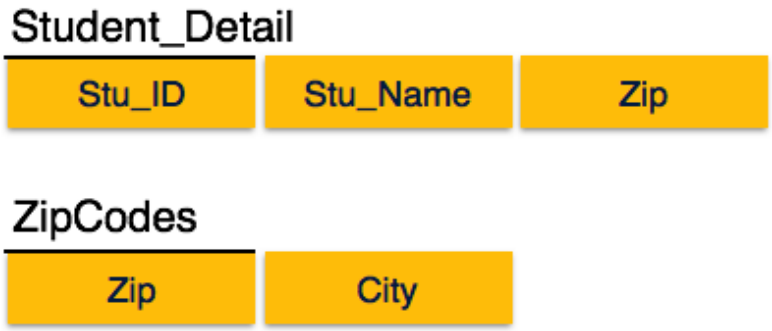

We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

## Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.


We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that Zip can be identified by Stu_ID and as well as city can be identified by Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists transitive dependency.

To bring this relation into third normal form, we break the relation into two relations as follows-




## Boyce Codd normal form (BCNF) (3.5NF)

BCNF stands for Boyce-Codd normal form and was made by R.F Boyce and E.F Codd in 1947. BCNF is the advance version of 3NF. It is stricter than 3NF.A functional dependency is said to be in BCNF if these properties hold:

- It should already be in 3NF.
- For a functional dependency say P→ Q, P should be a super key.

**Example:**

Below we have a college enrolment table with columns student_id, subject and professor.

As we can see, we have also added some sample data to the table.

| student_id | student_name | Subject | Professor |
|---|---|---|---|
| 101 | Ram | Java | P.Java1 |
| 101 | Ram | C++ | P.Cpp |
| 102 | Mohan | Java | P.Java2 |
| 103 | Suresh | C# | P.Chash |
| 104 | Hitesh | Java | P.Java |

In the table above:

- One student can enrol for multiple subjects. For example, student with **student_id** 101 **student_name** Ram, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between `subject` and `professor` here, where `subject` depends on the professor name.

This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the **2nd Normal Form** as there is no **Partial Dependency**.

And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.

But this table is not in **Boyce-Codd Normal Form**.

## Normalization

Apply the so-called *normalization rules* to check whether your database is structurally correct and optimal.

**First Normal Form (1NF)**: A table is 1NF if every cell contains a single value, not a list of values. This properties is known as *atomic*. 1NF also prohibits repeating group of columns such as item1, item2,.., itemN. Instead, you should create another table using one-to-many relationship.

**Second Normal Form (2NF)**: A table is 2NF, if it is 1NF and every non-key column is fully dependent on the primary key. Furthermore, if the primary key is made up of several columns, every non-key column shall depend on the entire set and not part of it.

For example, the primary key of the OrderDetails table comprising orderID and productID. If unitPrice is dependent only on productID, it shall not be kept in the OrderDetails table (but in the Products table). On the other hand, if the unitPrice is dependent on the product as well as the particular order, then it shall be kept in the OrderDetails table.

**Third Normal Form (3NF)**: A table is 3NF, if it is 2NF and the non-key columns are independent of each others. In other words, the non-key columns are dependent on primary key, only on the primary key and nothing else. For example, suppose that we have a Products table with columns productID (primary key), name and unitPrice. The column discountRate shall not belong to Products table if it is also dependent on the unitPrice, which is not part of the primary key.

**Higher Normal Form**: 3NF has its inadequacies, which leads to higher Normal form, such as Boyce/Codd Normal form, Fourth Normal Form (4NF) and Fifth Normal Form (5NF), which is beyond the scope of this tutorial.

At times, you may decide to break some of the normalization rules, for performance reason (e.g., create a column called totalPrice in Orders table which can be derived from the orderDetails records); or because the end-user requested for it. Make sure that you fully aware of it, develop programming logic to handle it, and properly document the decision.