DBMS & SQL- JAVATPOINT

1.DBMS Tutorial

DBMS Tutorial

What is Database open link

Types of Databases

What is RDBMS

DBMS vs RDBMS

DBMS vs File System

DBMS Architecture

Three schema Architecture

Data Models

Data model schema

Data Independence

DBMS Language

ACID Properties in DBMS

2.Data modelling

ER model concept

Notation for ER diagram

ER Design Issues

Mapping constraints

DBMS Keys

DBMS Generalization

DBMS Specialization

DBMS Aggregation

Convert ER into table

Relationship of Higher Degree

3.Relational data Model

Relational Model concept

Relational Algebra

Join Operation

Integrity Constraints

Relational Calculus

4.Normalization

Functional Dependency

Inference Rule

DBMS Normalization

DBMS 1NF

DBMS 2NF

DBMS 3NF

DBMS BCNF

DBMS 4NF

DBMS 5NF

Relational Decomposition

Multivalued Dependency

Join Dependency

Inclusion Dependence

Canonical Cover

5.Transaction Processing

Transaction

Transaction Property

States of Transaction

DBMS Schedule

Testing of Serializability

Conflict schedule

View Serializability

Recoverability of Schedule

Failure Classification

Log-Based Recovery

DBMS Checkpoint

Deadlock in DBMS

6.Concurrency Control

Concurrency Control

Lock based Protocol

Time stamping Protocol

Validation based Protocol

Thomas Write Rule

Multiple Granularity

Recovery Concurrent Transaction

7.File organization

File organization

Sequential File Organization

Heap File Organization

Hash File Organization

B+ File Organization

DBMS ISAM

Cluster File Organization

8.Indexing and B+ Tree

Indexing in DBMS

B+ Tree

9.Hashing

Hashing

Static Hashing

Dynamic Hashing

10.RAID

RAID

11.SQL Introduction

SQL Introduction

Characteristics of SQL

Advantage of SQL

SQL Datatype

SQL Command

SQL Operator

SQL Table

SQL SELECT Statement

SQL INSERT Statement

SQL Update Statement

SQL DELETE Statement

SQL View

SQL Index

SQL Sub Queries

SQL Clauses

SQL Aggregate Function

SQL JOIN

SQL Set Operation

DBMS Tutorial

DBMS Tutorial provides basic and advanced concepts of Database

. Our DBMS Tutorial is designed for beginners and professionals both.

Database management system is software that is used to manage the database.

Our DBMS

Tutorial includes all topics of DBMS such as introduction, ER model, keys, relational model, join operation, SQL
, functional dependency, transaction, concurrency control, etc.

What is Database

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System

- Database management system is a software which is used to manage the database. For example: MySQL

  , Oracle

  , etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.

- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware

  and software

  failures and restores the data if required.
- **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

Database

What is Data?

Data is a collection of a distinct small unit of information. It can be used in a variety of forms like text, numbers, media, bytes, etc. it can be stored in pieces of paper or electronic memory, etc.

Word 'Data' is originated from the word 'datum' that means 'single piece of information.' It is plural of the word datum.

In computing, Data is information that can be translated into a form for efficient movement and processing. Data is interchangeable.

What is Database?

A **database** is an organized collection of data, so that it can be easily accessed and managed.

You can organize data into tables, rows, columns, and index it to make it easier to find relevant information.

Database handlers create a database in such a way that only one set of software program provides access of data to all the users.

The **main purpose** of the database is to operate a large amount of information by storing, retrieving, and managing data.

There are many **dynamic websites** on the World Wide Web nowadays which are handled through databases. For example, a model that checks the availability of rooms in a hotel. It is an example of a dynamic website that uses a database.

There are many **databases available** like MySQL, Sybase, Oracle, MongoDB, Informix, PostgreSQL, SQL Server, etc.

Modern databases are managed by the database management system (DBMS).

SQL or Structured Query Language is used to operate on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.

A cylindrical structure is used to display the image of a database.

Evolution of Databases

The database has completed more than 50 years of journey of its evolution from flat-file system to relational and objects relational systems. It has gone through several generations.

The Evolution

File-Based

1968 was the year when File-Based database were introduced. In file-based databases, data was maintained in a flat file. Though files have many advantages, there are several limitations.

One of the major advantages is that the file system has various access methods, e.g., sequential, indexed, and random.

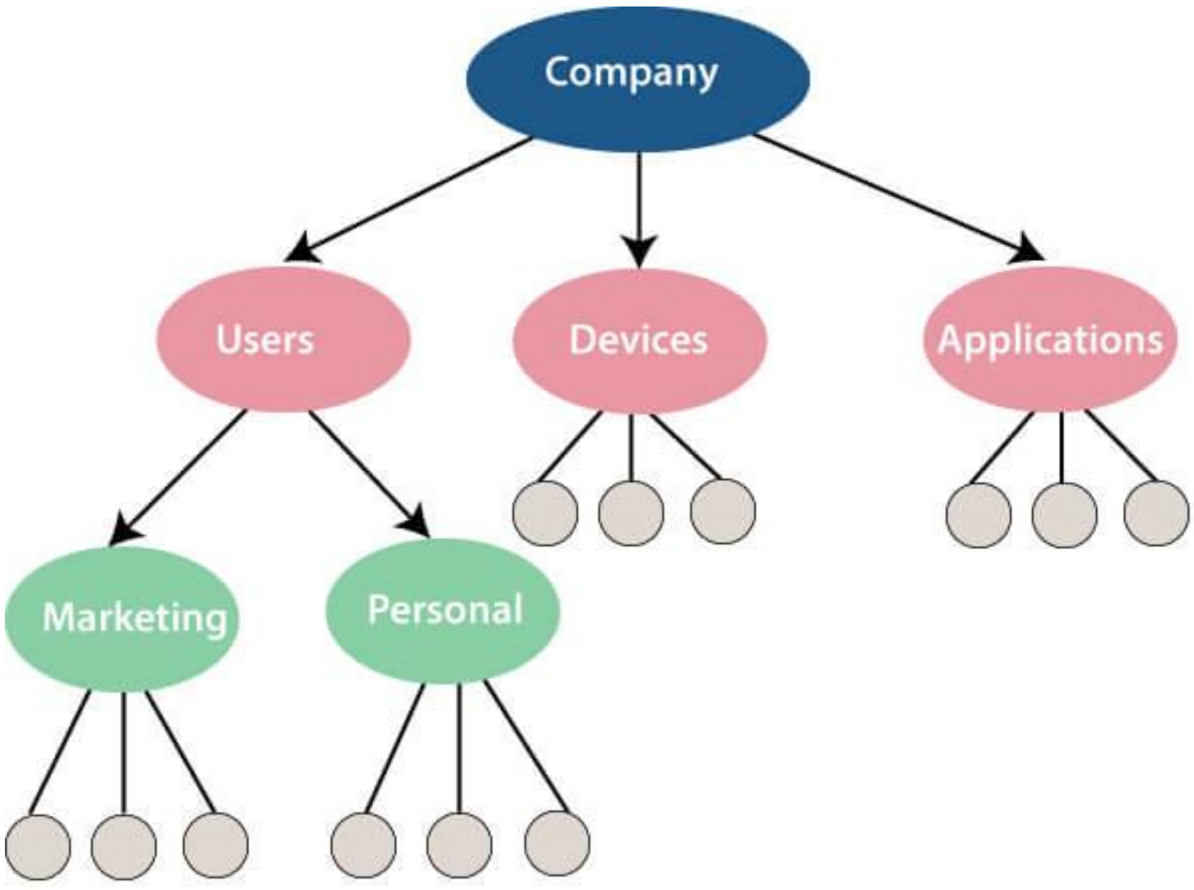
It requires extensive programming in a third-generation language such as COBOL, BASIC.

Hierarchical Data Model

1968-1980 was the era of the Hierarchical Database. Prominent hierarchical database model was IBM's first DBMS. It was called IMS (Information Management System).

In this model, files are related in a parent/child manner.

Below diagram represents Hierarchical Data Model. Small circle represents objects.



Like file system, this model also had some limitations like complex implementation, lack structural independence, can't easily handle a many-many relationship, etc.

Network data model

Charles Bachman developed the first DBMS at Honeywell called Integrated Data Store (IDS). It was developed in the early 1960s, but it was standardized in 1971 by the CODASYL group (Conference on Data Systems Languages).

In this model, files are related as owners and members, like to the common network model.

Network data model identified the following components:

- Network schema (Database organization)
- Sub-schema (views of database per user)
- Data management language (procedural)

This model also had some limitations like system complexity and difficult to design and maintain.

Relational Database

1970 - Present: It is the era of Relational Database and Database Management. In 1970, the relational model was proposed by E.F. Codd.

Relational database model has two main terminologies called instance and schema.

The instance is a table with rows or columns

Schema specifies the structure like name of the relation, type of each column and name.

This model uses some mathematical concept like set theory and predicate logic.

The first internet database application had been created in 1995.

During the era of the relational database, many more models had introduced like object-oriented model, object-relational model, etc.

Cloud database

Cloud database facilitates you to store, manage, and retrieve their structured, unstructured data via a cloud platform. This data is accessible over the Internet. Cloud databases are also called a database as service (DBaaS) because they are offered as a managed service.

Some best cloud options are:

- AWS (Amazon Web Services)
- Snowflake Computing
- Oracle Database Cloud Services
- Microsoft SQL server
- Google cloud spanner

Advantages of cloud database

Lower costs

Generally, company provider does not have to invest in databases. It can maintain and support one or more data centers.

Automated

Cloud databases are enriched with a variety of automated processes such as recovery, failover, and auto-scaling.

Increased accessibility

You can access your cloud-based database from any location, anytime. All you need is just an internet connection.

NoSQL Database

A NoSQL database is an approach to design such databases that can accommodate a wide variety of data models. NoSQL stands for "not only SQL." It is an alternative to traditional relational databases in which data is placed in tables, and data schema is perfectly designed before the database is built.

NoSQL databases are useful for a large set of distributed data.

Some examples of NoSQL database system with their category are:

- MongoDB, CouchDB, Cloudant **(Document-based)**
- Memcached, Redis, Coherence **(key-value store)**
- HBase, Big Table, Accumulo **(Tabular)**

Advantage of NoSQL

High Scalability

NoSQL can handle an extensive amount of data because of scalability. If the data grows, NoSQL database scale it to handle that data in an efficient manner.

High Availability

NoSQL supports auto replication. Auto replication makes it highly available because, in case of any failure, data replicates itself to the previous consistent state.

Disadvantage of NoSQL

Open source

NoSQL is an open-source database, so there is no reliable standard for NoSQL yet.

Management challenge

Data management in NoSQL is much more complicated than relational databases. It is very challenging to install and even more hectic to manage daily.

GUI is not available

GUI tools for NoSQL database are not easily available in the market.

Backup

Backup is a great weak point for NoSQL databases. Some databases, like MongoDB, have no powerful approaches for data backup.

The Object-Oriented Databases

The object-oriented databases contain data in the form of object and classes. Objects are the real-world entity, and types are the collection of objects. An object-oriented database is a combination of relational model features with objects oriented principles. It is an alternative implementation to that of the relational model.

Object-oriented databases hold the rules of object-oriented programming. An object-oriented database management system is a hybrid application.

The object-oriented database model contains the following properties.

Object-oriented programming properties

- Objects
- Classes
- Inheritance
- Polymorphism
- Encapsulation

Relational database properties

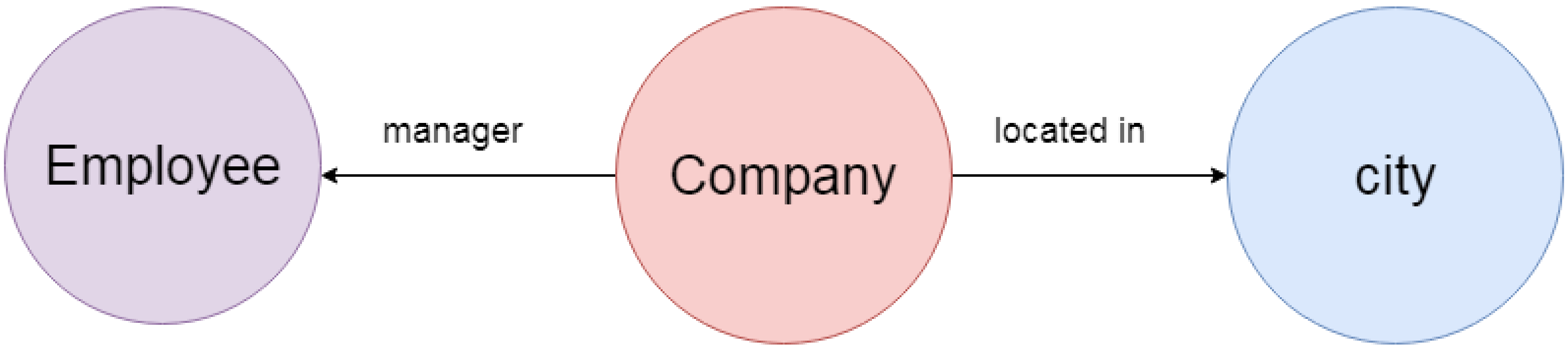
- Atomicity

- Consistency
- Integrity
- Durability
- Concurrency
- Query processing

Graph Databases

A graph database is a NoSQL database. It is a graphical representation of data. It contains nodes and edges. A node represents an entity, and each edge represents a relationship between two edges. Every node in a graph database represents a unique identifier.

Graph databases are beneficial for searching the relationship between data because they highlight the relationship between relevant data.



Graph databases are very useful when the database contains a complex relationship and dynamic schema.

It is mostly used in **supply chain management**, identifying the source of **IP telephony**.

DBMS (Data Base Management System)

Database management System is software which is used to store and retrieve the database. For example, Oracle, MySQL, etc.; these are some popular DBMS tools.

- DBMS provides the interface to perform the various operations like creation, deletion, modification, etc.
- DBMS allows the user to create their databases as per their requirement.
- DBMS accepts the request from the application and provides specific data through the operating system.
- DBMS contains the group of programs which acts according to the user instruction.
- It provides security to the database.

Advantage of DBMS

Controls redundancy

It stores all the data in a single database file, so it can control data redundancy.

Data sharing

An authorized user can share the data among multiple users.

Backup

It providesBackup and recovery subsystem. This recovery system creates automatic data from system failure and restores data if required.

Multiple user interfaces

It provides a different type of user interfaces like GUI, application interfaces.

Disadvantage of DBMS

Size

It occupies large disk space and large memory to run efficiently.

Cost

DBMS requires a high-speed data processor and larger memory to run DBMS software, so it is costly.

Complexity

DBMS creates additional complexity and requirements.

RDBMS (Relational Database Management System)

The word RDBMS is termed as 'Relational Database Management System.' It is represented as a table that contains rows and column.

RDBMS is based on the Relational model; it was introduced by E. F. Codd.

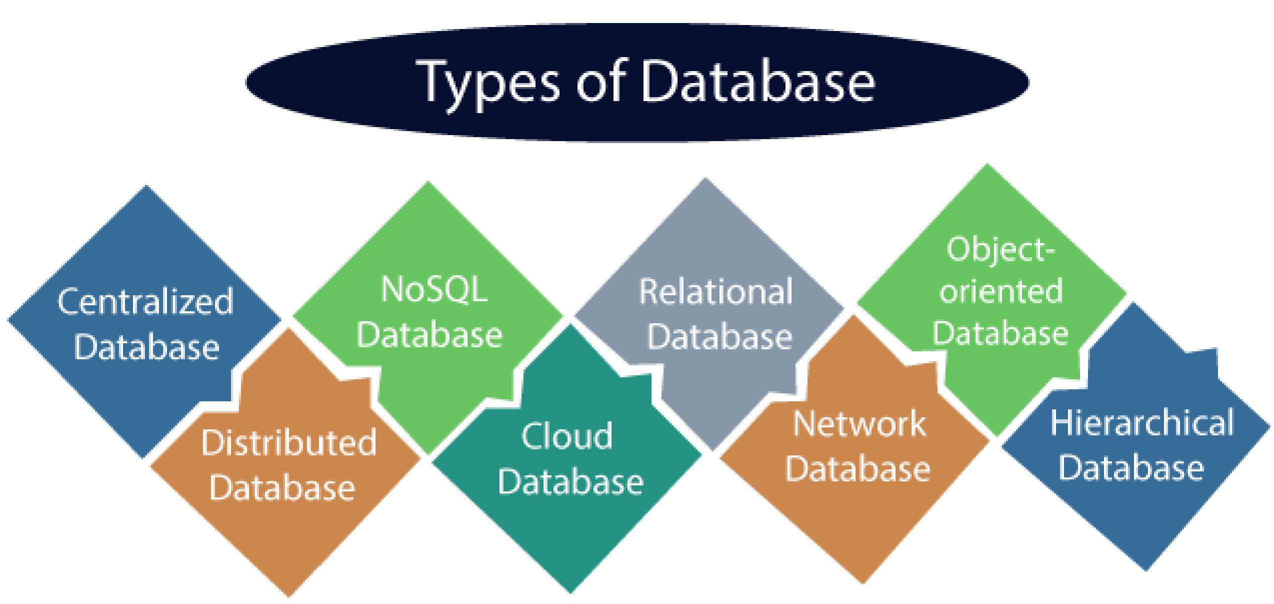
A relational database contains the following components:

- Table
- Record/ Tuple
- Field/Column name /Attribute
- Instance
- Schema
- Keys

An RDBMS is a tabular DBMS that maintains the security, integrity, accuracy, and consistency of the data.

Types of Databases

There are various types of databases used for storing different varieties of data:



1) Centralized Database

It is the type of database that stores data at a centralized database system. It comforts the users to access the stored data from different locations through several applications. These applications contain the authentication process to let users access data securely. An example of a Centralized database can be Central Library that carries a central database of each library in a college/university.

Advantages of Centralized Database

- It has decreased the risk of data management, i.e., manipulation of data will not affect the core data.
- Data consistency is maintained as it manages data in a central repository.
- It provides better data quality, which enables organizations to establish data standards.
- It is less costly because fewer vendors are required to handle the data sets.

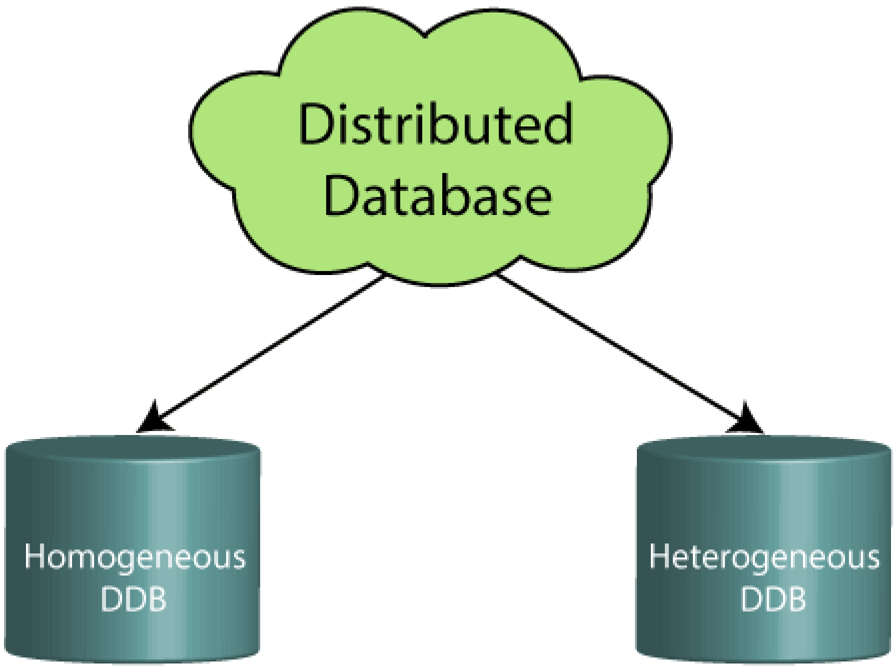
Disadvantages of Centralized Database

- The size of the centralized database is large, which increases the response time for fetching the data.
- It is not easy to update such an extensive database system.
- If any server failure occurs, entire data will be lost, which could be a huge loss.

2) Distributed Database

Unlike a centralized database system, in distributed systems, data is distributed among different database systems of an organization. These database systems are connected via communication links. Such links help the end-users to access the data easily. **Examples** of the Distributed database are Apache Cassandra, HBase, Ignite, etc.

We can further divide a distributed database system into:



- **Homogeneous DDB:** Those database systems which execute on the same operating system and use the same application process and carry the same hardware devices.
- **Heterogeneous DDB:** Those database systems which execute on different operating systems under different application procedures, and carries different hardware devices.

Advantages of Distributed Database

- Modular development is possible in a distributed database, i.e., the system can be expanded by including new computers and connecting them to the distributed system.
- One server failure will not affect the entire data set.

3) Relational Database

This database is based on the relational data model, which stores data in the form of rows(tuple) and columns(attributes), and together forms a table(relation). A relational database uses SQL for storing, manipulating, as well as maintaining the data. E.F. Codd invented the database in 1970. Each table in the database carries a key that makes the data unique from others. **Examples** of Relational databases are MySQL, Microsoft SQL Server, Oracle, etc.

Properties of Relational Database

There are following four commonly known properties of a relational model known as ACID properties, where:

A means Atomicity: This ensures the data operation will complete either with success or with failure. It follows the 'all or nothing' strategy. For example, a transaction will either be committed or will abort.

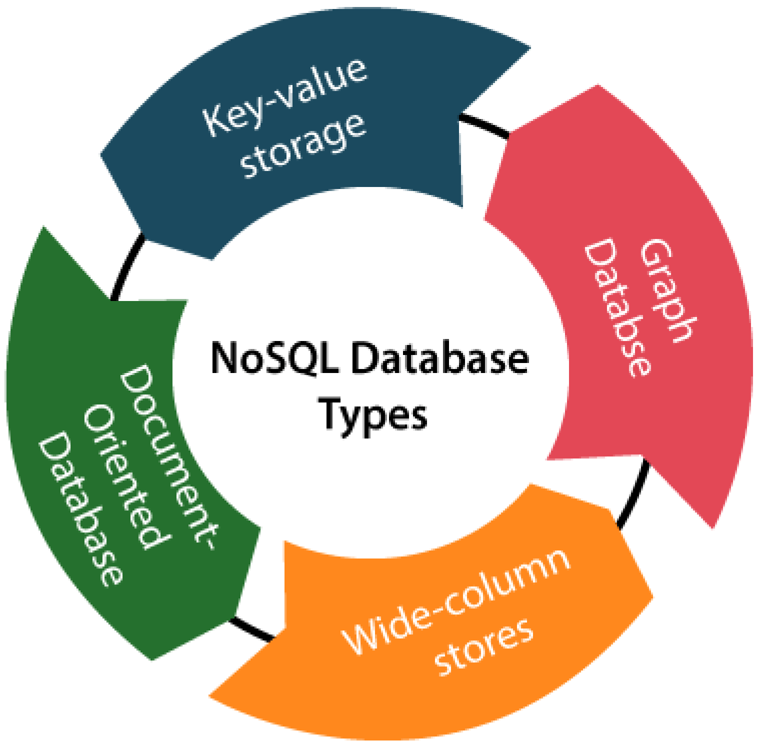
C means Consistency: If we perform any operation over the data, its value before and after the operation should be preserved. For example, the account balance before and after the transaction should be correct, i.e., it should remain conserved.

I means Isolation: There can be concurrent users for accessing data at the same time from the database. Thus, isolation between the data should remain isolated. For example, when multiple transactions occur at the same time, one transaction effects should not be visible to the other transactions in the database.

D means Durability: It ensures that once it completes the operation and commits the data, data changes should remain permanent.

4) NoSQL Database

Non-SQL/Not Only SQL is a type of database that is used for storing a wide range of data sets. It is not a relational database as it stores data not only in tabular form but in several different ways. It came into existence when the demand for building modern applications increased. Thus, NoSQL presented a wide variety of database technologies in response to the demands. We can further divide a NoSQL database into the following four types:



a. **Key-value storage:** It is the simplest type of database storage where it stores every single item as a key (or attribute name) holding its value, together.

b. **Document-oriented Database:** A type of database used to store data as JSON-like document. It helps developers in storing data by using the same document-model format as used in the application code.

c. **Graph Databases:** It is used for storing vast amounts of data in a graph-like structure. Most commonly, social networking websites use the graph database.

d. **Wide-column stores:** It is similar to the data represented in relational databases. Here, data is stored in large columns together, instead of storing in rows.

Advantages of NoSQL Database

- It enables good productivity in the application development as it is not required to store data in a structured format.
- It is a better option for managing and handling large data sets.
- It provides high scalability.
- Users can quickly access data from the database through key-value.

5) Cloud Database

A type of database where data is stored in a virtual environment and executes over the cloud computing platform. It provides users with various cloud computing services (SaaS, PaaS, IaaS, etc.) for accessing the database. There are numerous cloud platforms, but the best options are:

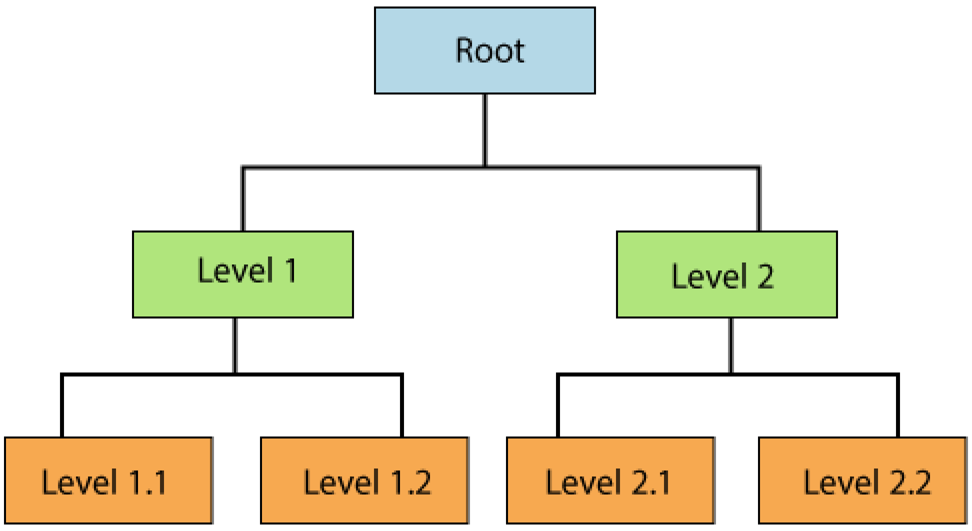
- Amazon Web Services(AWS)
- Microsoft Azure
- Kamatera
- PhonixNAP
- ScienceSoft
- Google Cloud SQL, etc.

6) Object-oriented Databases

The type of database that uses the object-based data model approach for storing data in the database system. The data is represented and stored as objects which are similar to the objects used in the object-oriented programming language.

7) Hierarchical Databases

It is the type of database that stores data in the form of parent-children relationship nodes. Here, it organizes data in a tree-like structure.



Hierarchical Database

Data get stored in the form of records that are connected via links. Each child record in the tree will contain only one parent. On the other hand, each parent record can have multiple child records.

8) Network Databases

It is the database that typically follows the network data model. Here, the representation of data is in the form of nodes connected via links between them. Unlike the hierarchical database, it allows each record to have multiple children and parent nodes to form a generalized graph structure.

9) Personal Database

Collecting and storing data on the user's system defines a Personal Database. This database is basically designed for a single user.

Advantage of Personal Database

- It is simple and easy to handle.
- It occupies less storage space as it is small in size.

10) Operational Database

The type of database which creates and updates the database in real-time. It is basically designed for executing and handling the daily data operations in several businesses. For example, An organization uses operational databases for managing per day transactions.

11) Enterprise Database

Large organizations or enterprises use this database for managing a massive amount of data. It helps organizations to increase and improve their efficiency. Such a database allows simultaneous access to users.

Advantages of Enterprise Database:

- Multi processes are supportable over the Enterprise database.
- It allows executing parallel queries on the system.

What is RDBMS

RDBMS stands for *Relational Database Management Systems.*.

All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS.

It is called Relational Data Base Management System (RDBMS) because it is based on relational model introduced by E.F. Codd.

How it works

Data is represented in terms of tuples (rows) in RDBMS.

Relational database is most commonly used database. It contains number of tables and each table has its own primary key.

Due to a collection of organized set of tables, data can be accessed easily in RDBMS.

Brief History of RDBMS

During 1970 to 1972, E.F. Codd published a paper to propose the use of relational database model.

RDBMS is originally based on that E.F. Codd's relational model invention.

What is table

The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data.

A table is the simplest example of data storage in RDBMS.

Let's see the example of student table.

ID	Name	AGE	COURSE
1	Ajeet	24	B.Tech
2	aryan	20	C.A
3	Mahesh	21	BCA
4	Ratan	22	MCA
5	Vimal	26	BSC

What is field

Field is a smaller entity of the table which contains specific information about every record in the table. In the above example, the field in the student table consist of id, name, age, course.

What is row or record

A row of a table is also called record. It contains the specific information of each individual entry in the table. It is a horizontal entity in the table. For example: The above table contains 5 records.

Let's see one record/row in the table.

1	Ajeet	24	B.Tech

What is column

A column is a vertical entity in the table which contains all information associated with a specific field in a table. For example: "name" is a column in the above table which contains all information about student's name.

Ajeet
Aryan
Mahesh

Ratan
Vimal

NULL Values

The NULL value of the table specifies that the field has been left blank during record creation. It is totally different from the value filled with zero or a field that contains space.

Data Integrity

There are the following categories of data integrity exist with each RDBMS:

Entity integrity: It specifies that there should be no duplicate rows in a table.

Domain integrity: It enforces valid entries for a given column by restricting the type, the format, or the range of values.

Referential integrity: It specifies that rows cannot be deleted, which are used by other records.

User-defined integrity: It enforces some specific business rules that are defined by users. These rules are different from entity, domain or referential integrity.

Difference between DBMS and RDBMS

Although DBMS and RDBMS both are used to store information in physical database but there are some remarkable differences between them.

The main differences between DBMS and RDBMS are given below:

No.	DBMS	RDBMS
1)	DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	**Normalization is not** present in DBMS.	**Normalization is** present in RDBMS.
4)	DBMS does **not apply any security** with regards to data manipulation.	RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be **no relation between the tables**.	in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS **does not support distributed database**.	RDBMS **supports distributed database**.
8)	DBMS is meant to be for small organization and **deal with small data**. it supports **single user**.	RDBMS is designed to **handle large amount of data**. it supports **multiple users**.

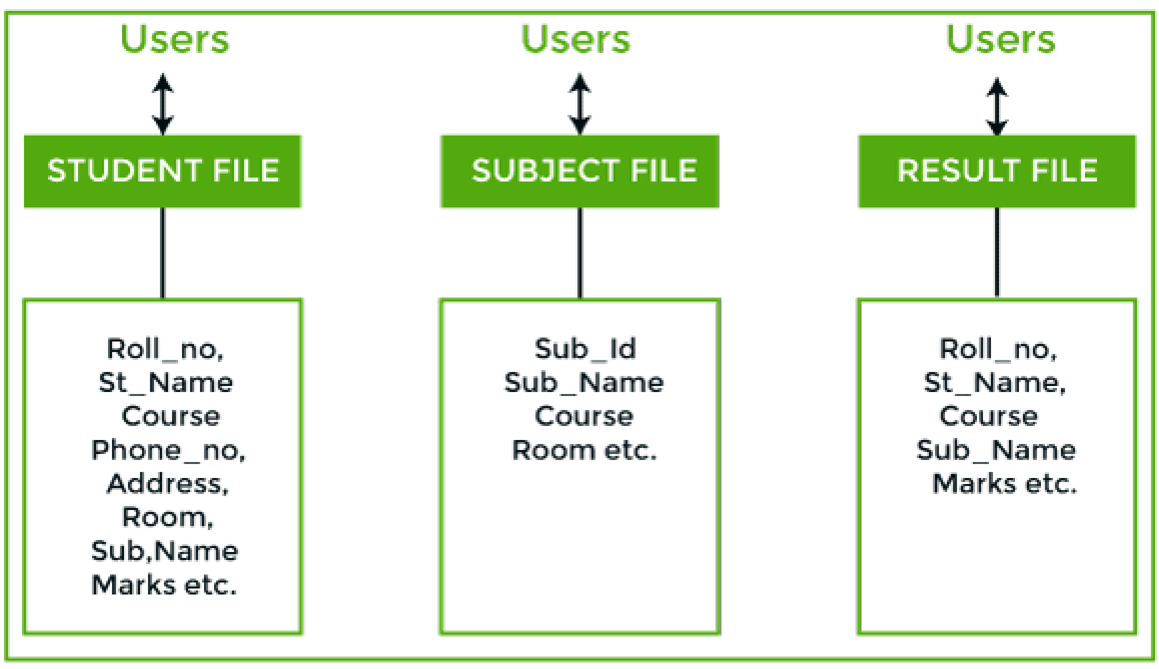
9)	Examples of DBMS are file systems, **xml** etc.	Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc.

After observing the differences between DBMS and RDBMS, you can say that RDBMS is an extension of DBMS. There are many software products in the market today who are compatible for both DBMS and RDBMS. Means today a RDBMS application is DBMS application and vice-versa.

DBMS vs. File System

File System Approach

File based systems were an early attempt to computerize the manual system. It is also called a traditional based approach in which a decentralized approach was taken where each department stored and controlled its own data with the help of a data processing specialist. The main role of a data processing specialist was to create the necessary computer file structures, and also manage the data within structures and design some application programs that create reports based on file data.



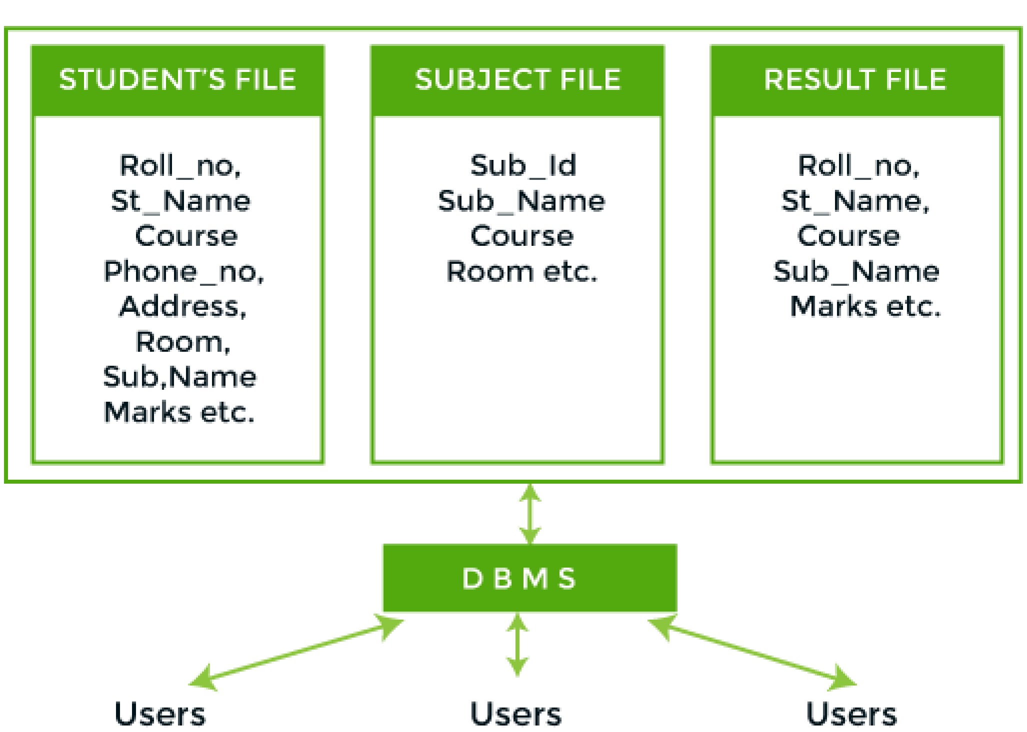
In the above figure:

Consider an example of a student's file system. The student file will contain information regarding the student (i.e. roll no, student name, course etc.). Similarly, we have a subject file that contains information about the subject and the result file which contains the information regarding the result.

Some fields are duplicated in more than one file, which leads to data redundancy. So to overcome this problem, we need to create a centralized system, i.e. DBMS approach.

DBMS:

A database approach is a well-organized collection of data that are related in a meaningful way which can be accessed by different users but stored only once in a system. The various operations performed by the DBMS system are: Insertion, deletion, selection, sorting etc.



In the above figure,

In the above figure, duplication of data is reduced due to centralization of data.

There are the following differences between DBMS and File systems:

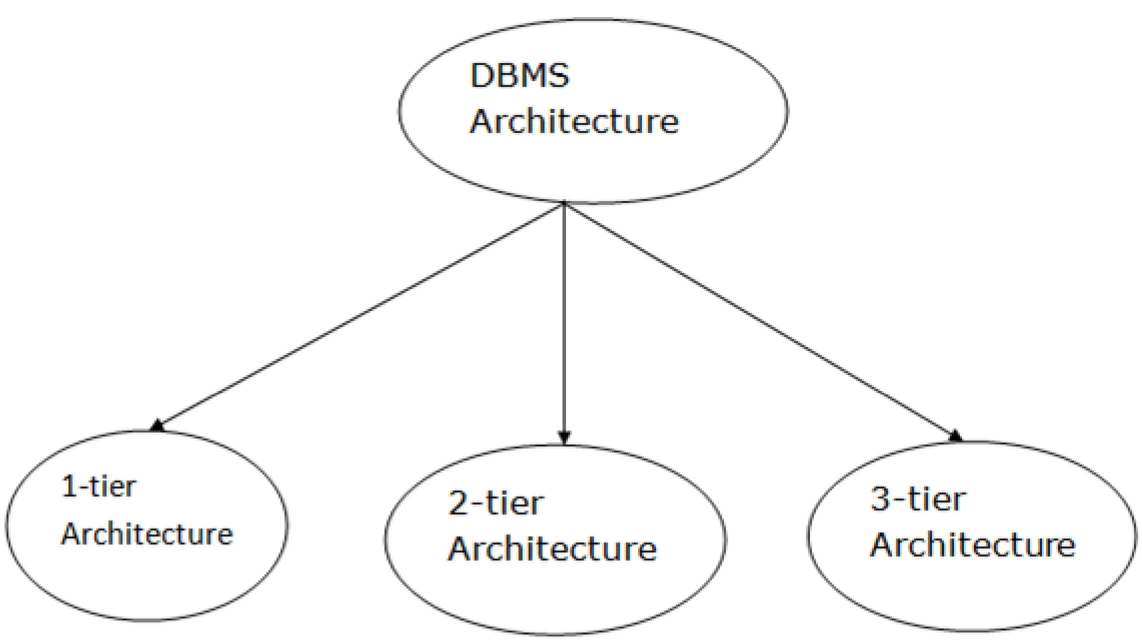
Basis	DBMS Approach	File System Approach
Meaning	DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	The file system is a collection of data. In this system, the user has to write the procedures for managing the database.
Sharing of data	Due to the centralized approach, data sharing is easy.	Data is distributed in many files, and it may be of different formats, so it isn't easy to share data.
Data Abstraction	DBMS gives an abstract view of data that hides the details.	The file system provides the detail of the data representation and storage of data.
Security and Protection	DBMS provides a good protection mechanism.	It isn't easy to protect a file under the file system.
Recovery Mechanism	DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from system failure.	The file system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will be lost.
Manipulation Techniques	DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	The file system can't efficiently store and retrieve the data.
Concurrency Problems	DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while deleting some information or updating some information.
Where to use	Database approach used in large systems which interrelate many files.	File system approach used in large systems which interrelate many files.
Cost	The database system is expensive to design.	The file system approach is cheaper to design.
Data Redundancy and Inconsistency	Due to the centralization of the database, the problems of data redundancy and inconsistency are controlled.	In this, the files and application programs are created by different programmers so that there exists a lot of duplication of data which may lead to inconsistency.
Structure	The database structure is complex to design.	The file system approach has a simple structure.
Data Independence	In this system, Data Independence exists, and it can be of two types. o Logical Data	In the File system approach, there exists no Data Independence.

	Independence ○ Physical Data Independence	
Integrity Constraints	Integrity Constraints are easy to apply.	Integrity Constraints are difficult to implement in file system.
Data Models	In the database approach, 3 types of data models exist: ○ Hierarchal data models ○ Network data models ○ Relational data models	In the file system approach, there is no concept of data models exists.
Flexibility	Changes are often a necessity to the content of the data stored in any system, and these changes are more easily with a database approach.	The flexibility of the system is less as compared to the DBMS approach.
Examples	Oracle, SQL Server, Sybase etc.	Cobol, C++ etc.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

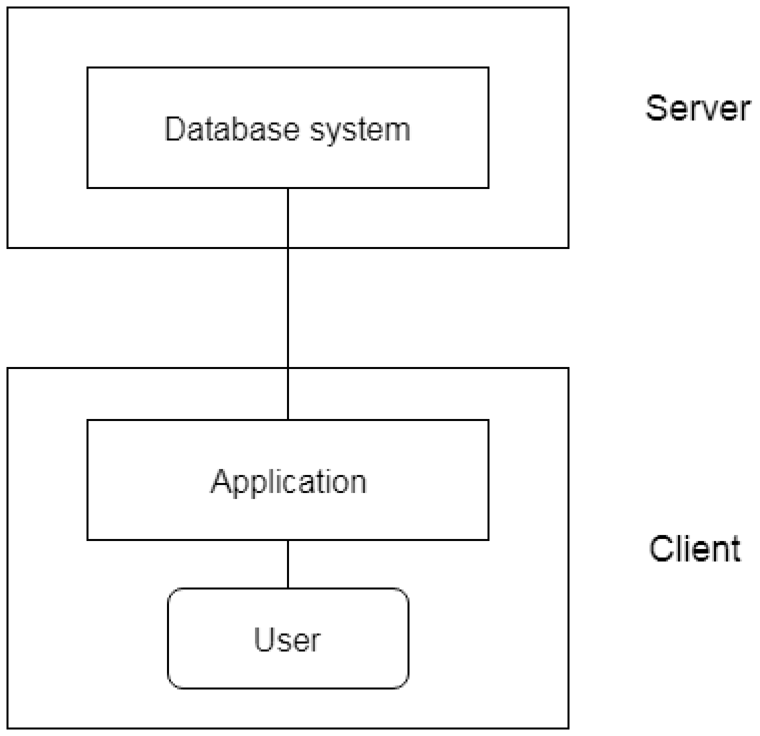


Fig: 2-tier Architecture

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

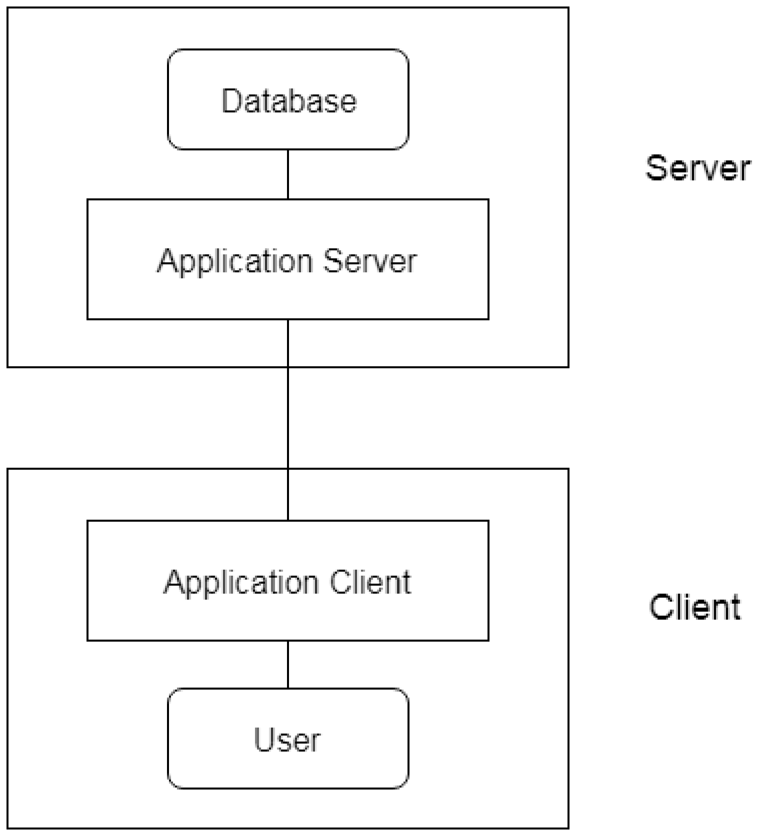


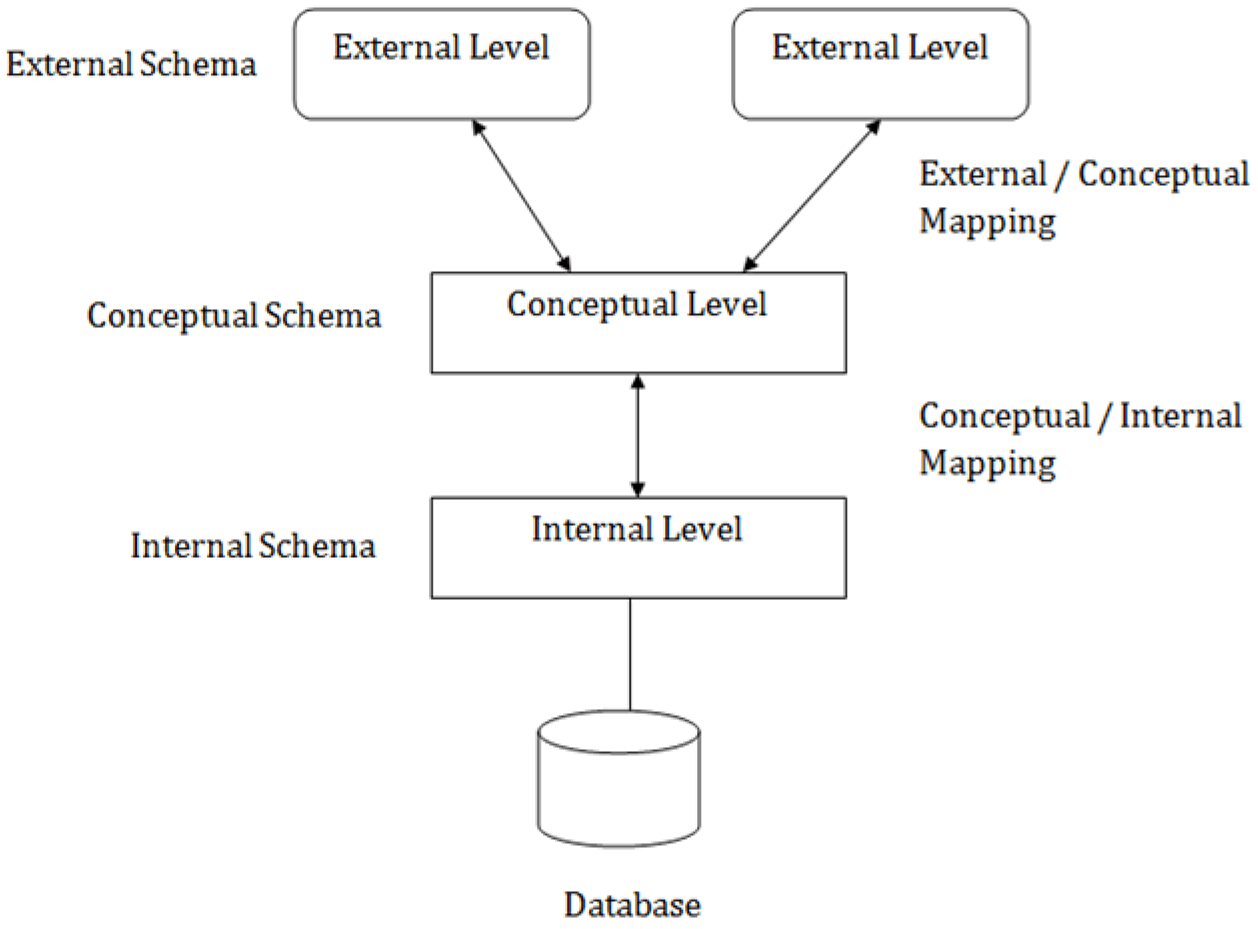
Fig: 3-tier Architecture

Three schema Architecture

- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.

- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database.
- The three schema architecture contains three-levels. It breaks the database down into three different categories.

The three-schema architecture is as follows:



In the above diagram:

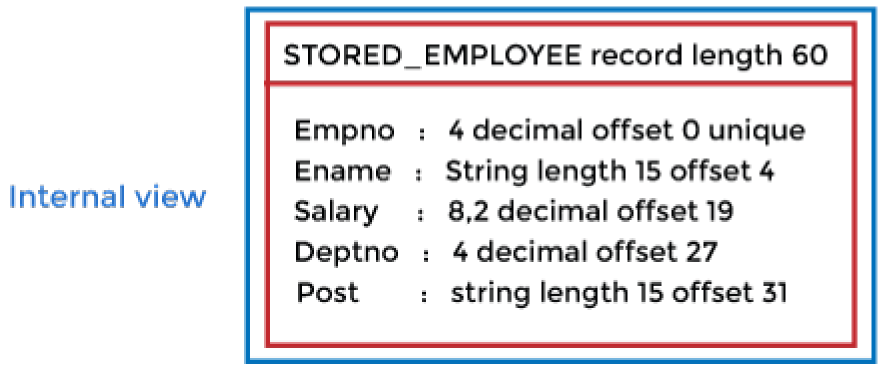
- It shows the DBMS architecture.
- Mapping is used to transform the request and response between various database levels of architecture.
- Mapping is not good for small DBMS because it takes more time.
- In External / Conceptual mapping, it is necessary to transform the request from external level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

Objectives of Three schema Architecture

The main objective of three level architecture is to enable multiple users to access the same data with a personalized view while storing the underlying data only once. Thus it separates the user's view from the physical structure of the database. This separation is desirable for the following reasons:

- Different users need different views of the same data.
- The approach in which a particular user needs to see the data may change over time.
- The users of the database should not worry about the physical implementation and internal workings of the database such as data compression and encryption techniques, hashing, optimization of the internal structures etc.
- All users should be able to access the same data according to their requirements.
- DBA should be able to change the conceptual structure of the database without affecting the user's
- Internal structure of the database should be unaffected by changes to physical aspects of the storage.

1. Internal Level



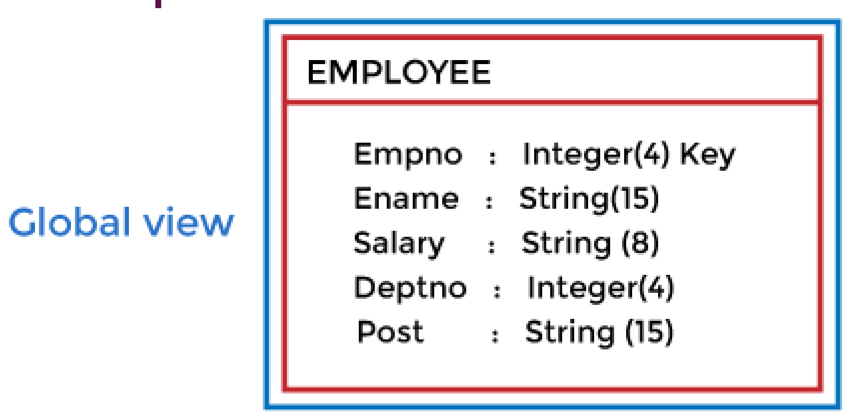
- The internal level has an internal schema which describes the physical storage structure of the database.

- The internal schema is also known as a physical schema.
- It uses the physical data model. It is used to define that how the data will be stored in a block.
- The physical level is used to describe complex low-level data structures in detail.

The internal level is generally is concerned with the following activities:

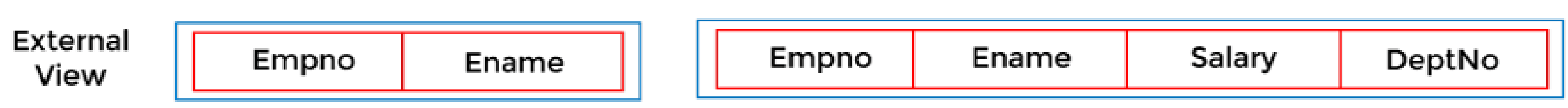
- Storage space allocations.
  For Example: B-Trees, Hashing etc.
- Access paths.
  For Example: Specification of primary and secondary keys, indexes, pointers and sequencing.
- Data compression and encryption techniques.
- Optimization of internal structures.
- Representation of stored fields.

2. Conceptual Level



- The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
- The conceptual schema describes the structure of the whole database.
- The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
- In the conceptual level, internal details such as an implementation of the data structure are hidden.
- Programmers and database administrators work at this level.

3. External Level



- At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
- An external schema is also known as view schema.
- Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
- The view schema describes the end user interaction with database systems.

Mapping between Views

The three levels of DBMS architecture don't exist independently of each other. There must be correspondence between the three levels i.e. how they actually correspond with each other. DBMS is responsible for correspondence between the three types of schema. This correspondence is called Mapping.

There are basically two types of mapping in the database architecture:

- Conceptual/ Internal Mapping
- External / Conceptual Mapping

Conceptual/ Internal Mapping

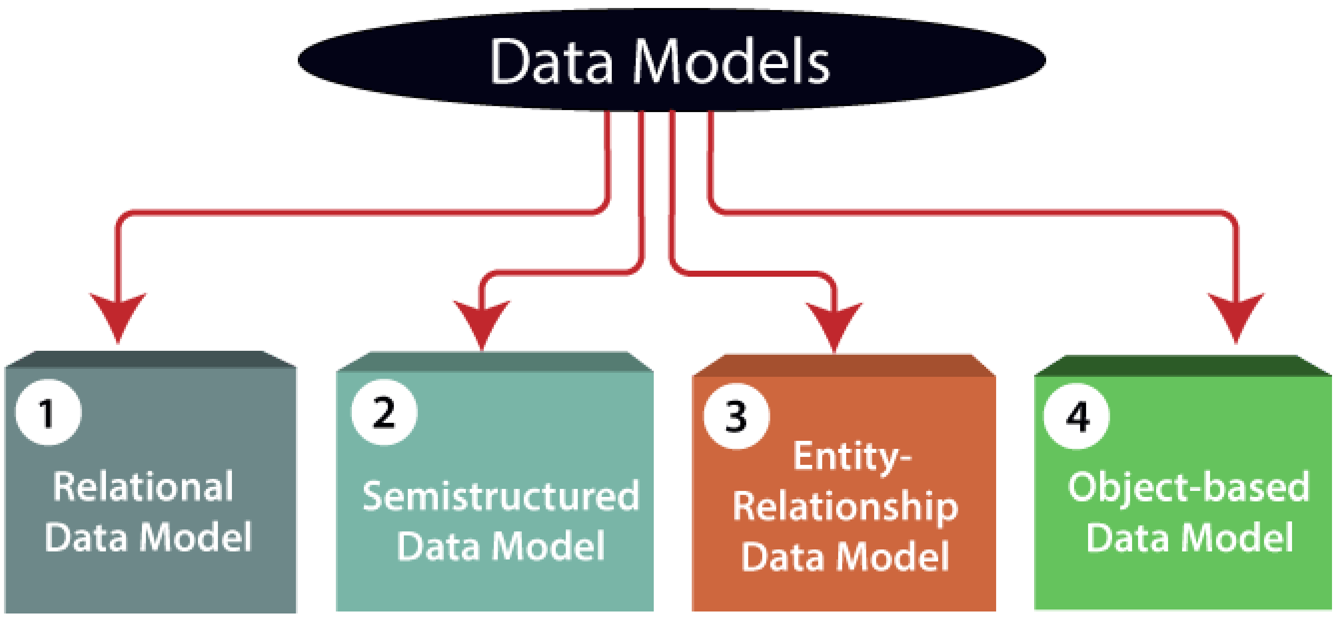
The Conceptual/ Internal Mapping lies between the conceptual level and the internal level. Its role is to define the correspondence between the records and fields of the conceptual level and files and data structures of the internal level.

External/ Conceptual Mapping

The external/Conceptual Mapping lies between the external level and the Conceptual level. Its role is to define the correspondence between a particular external and the conceptual view.

Data Models

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:



1) Relational Data Model: This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

2) Entity-Relationship Data Model: An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

3) Object-based Data Model: An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

4) Semistructured Data Model: This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

Data model Schema and Instance

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
- A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modeling.

A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram. For example, the given figure neither show the data type of each data item nor the relationship among various files.

In the database, actual data changes quite frequently. For example, in the given figure, the database changes whenever we add a new grade or add a student. The data at a particular moment of time is called the instance of the database.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

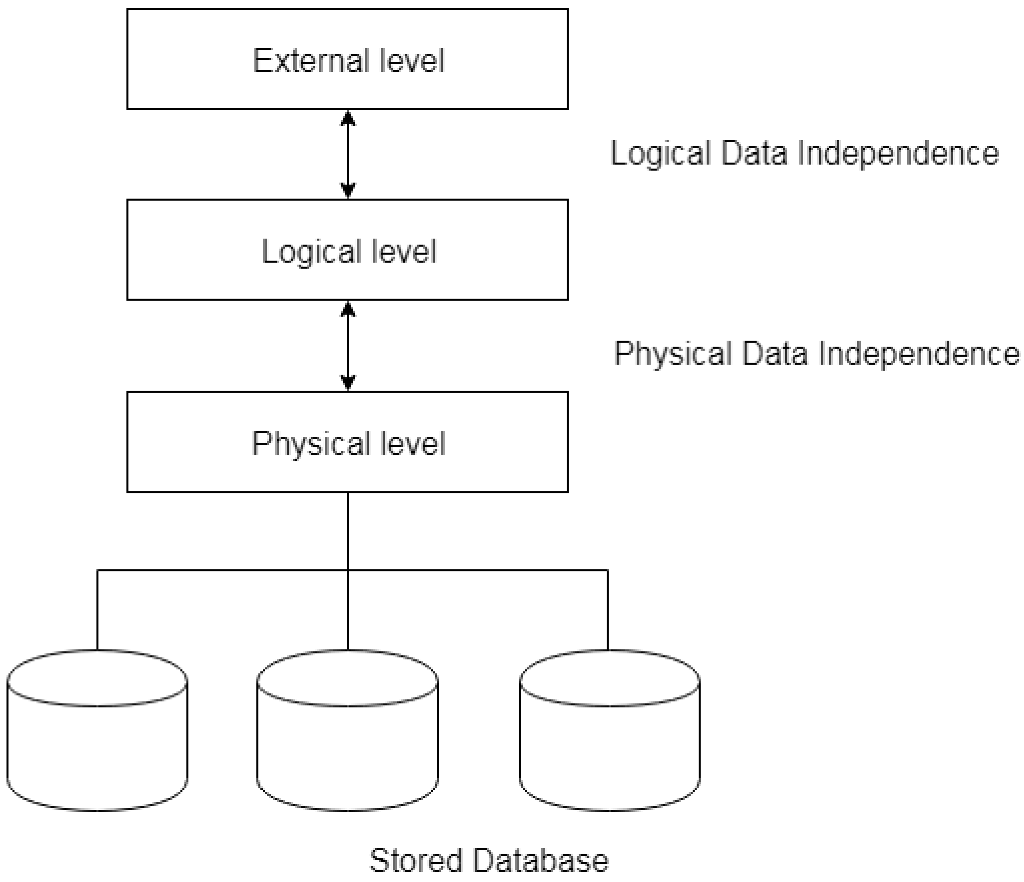
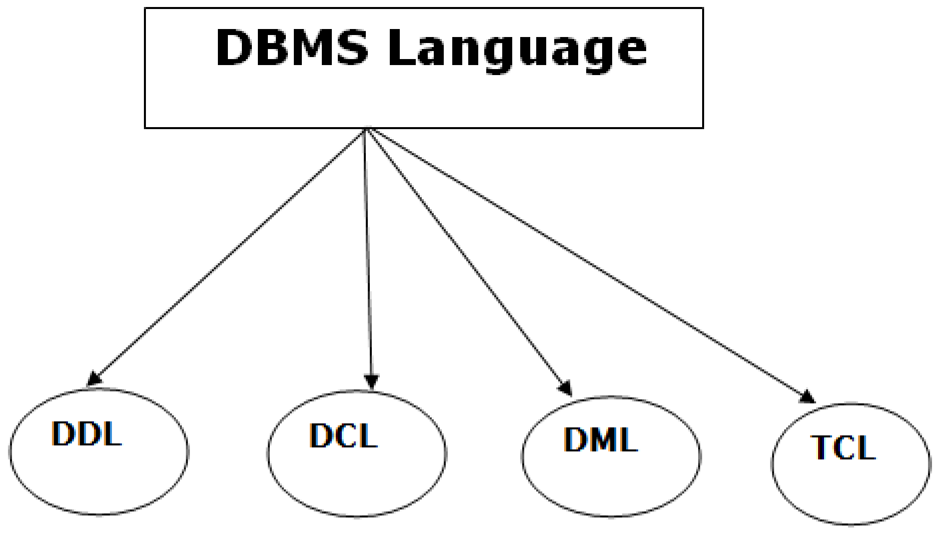


Fig: Data Independence

Database Language

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language



1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.

- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

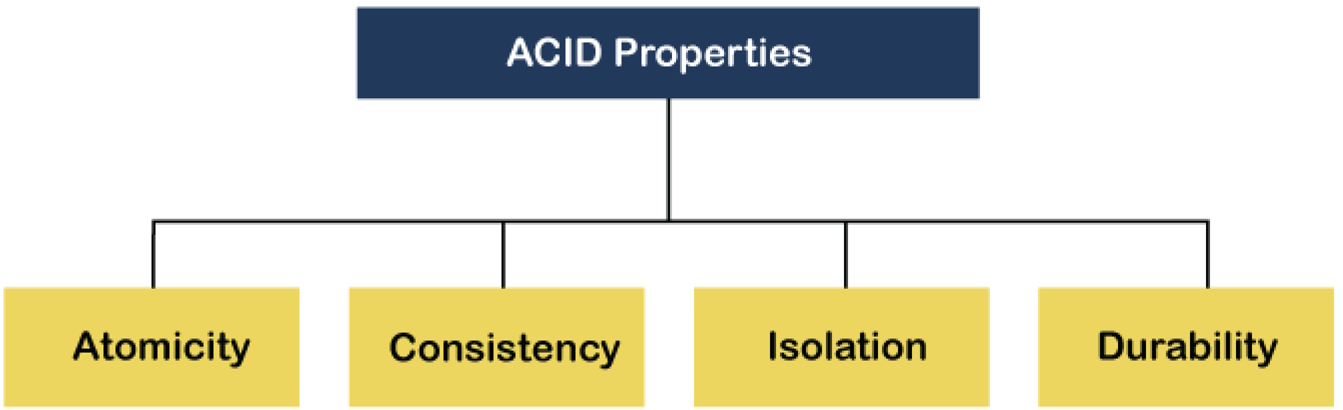
ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

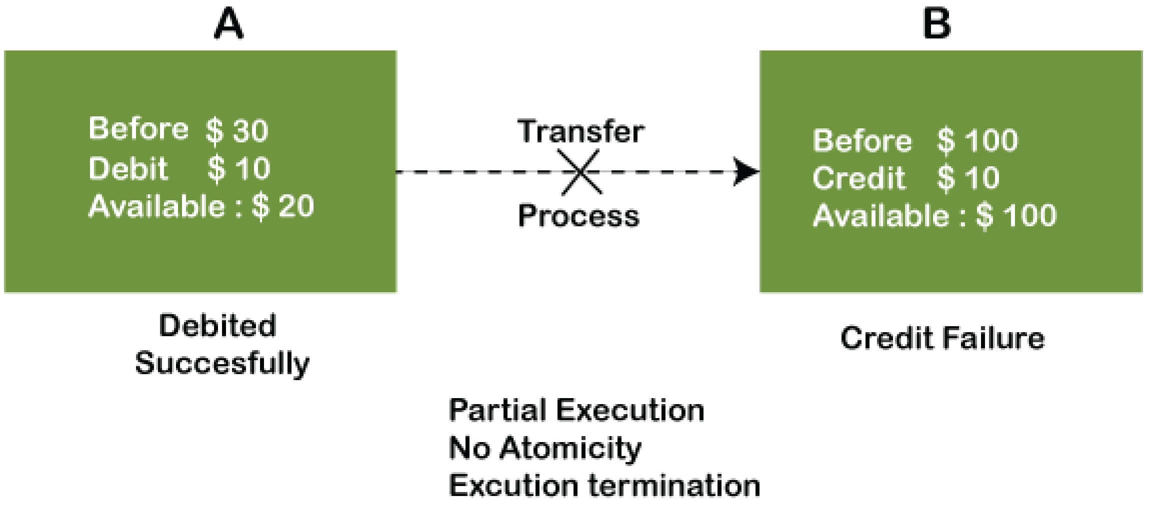
ACID Properties

The expansion of the term ACID defines for:



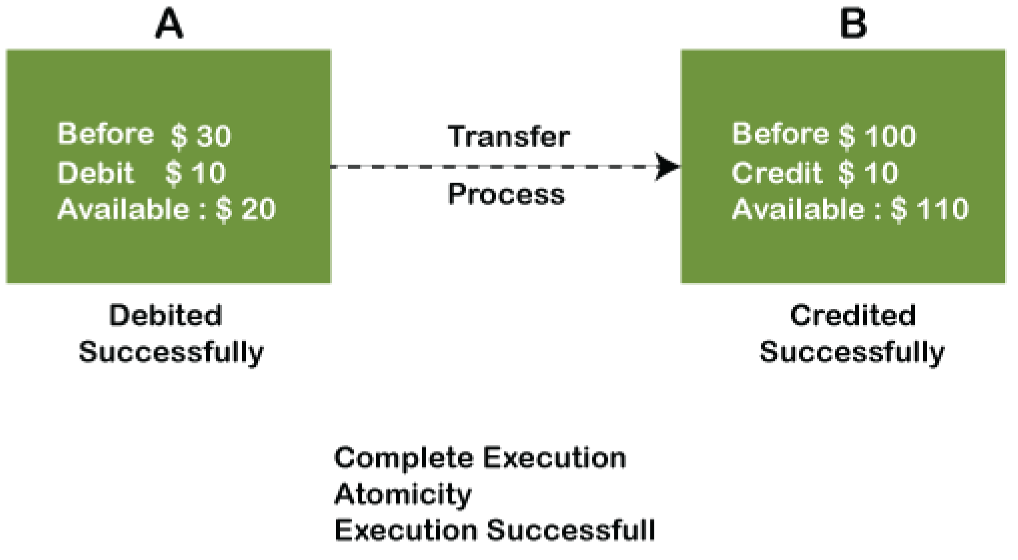
1) Atomicity: The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

Example: If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

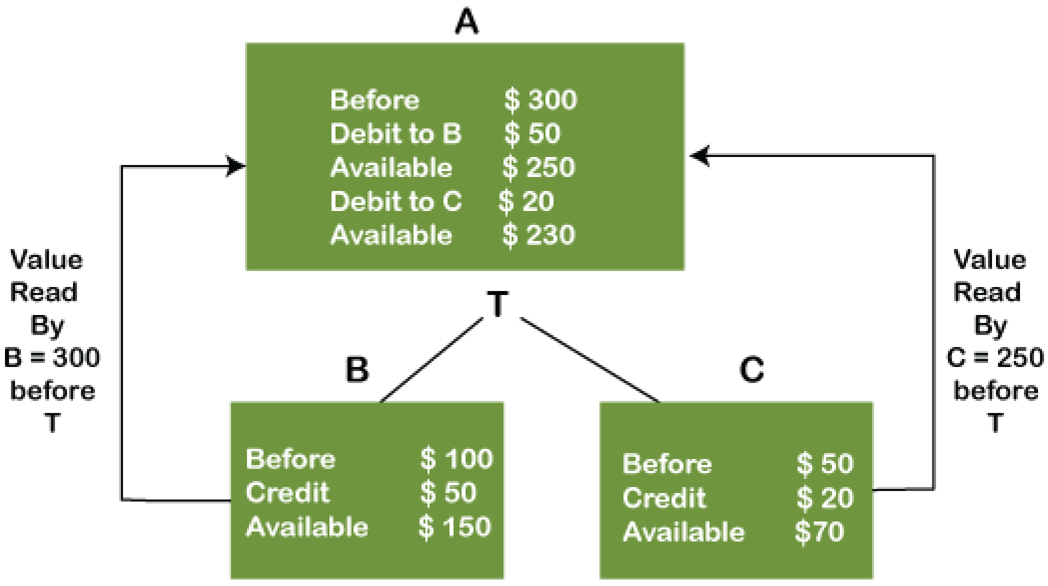
The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.



Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

2) Consistency: The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

Example:

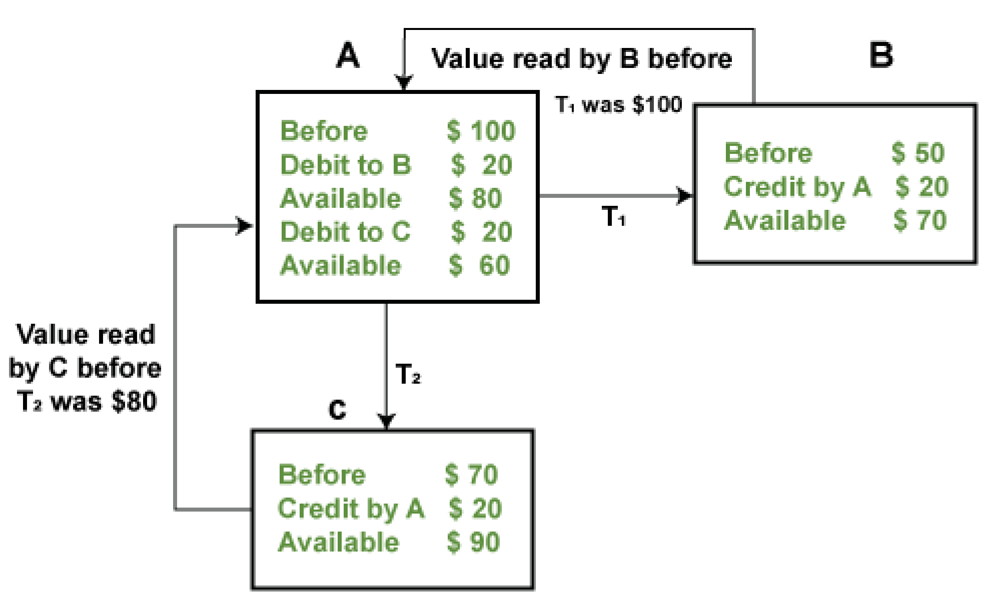


Data Consistent

In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

4) Isolation: The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



Isolation - Independent execution of T_1 & T_2 by A

4) Durability: Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

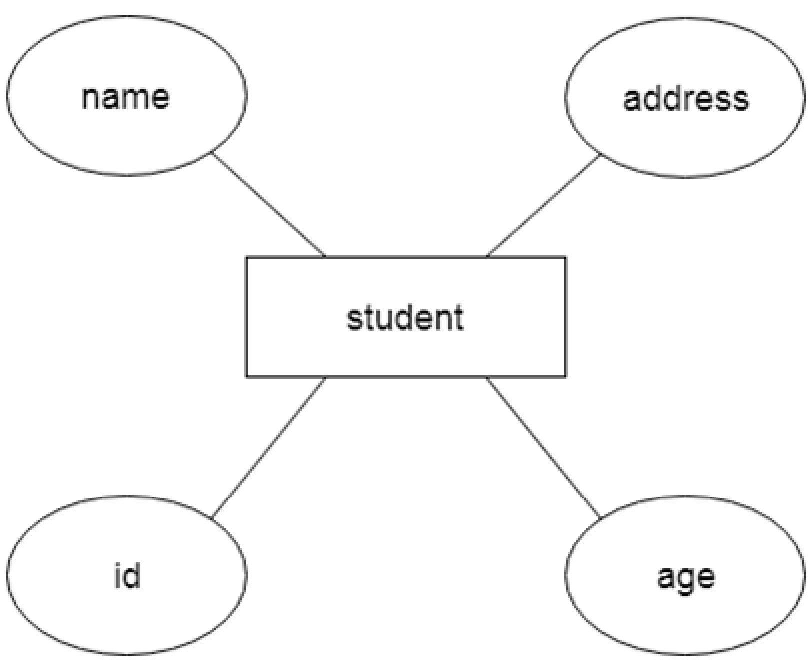
Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

Data modeling

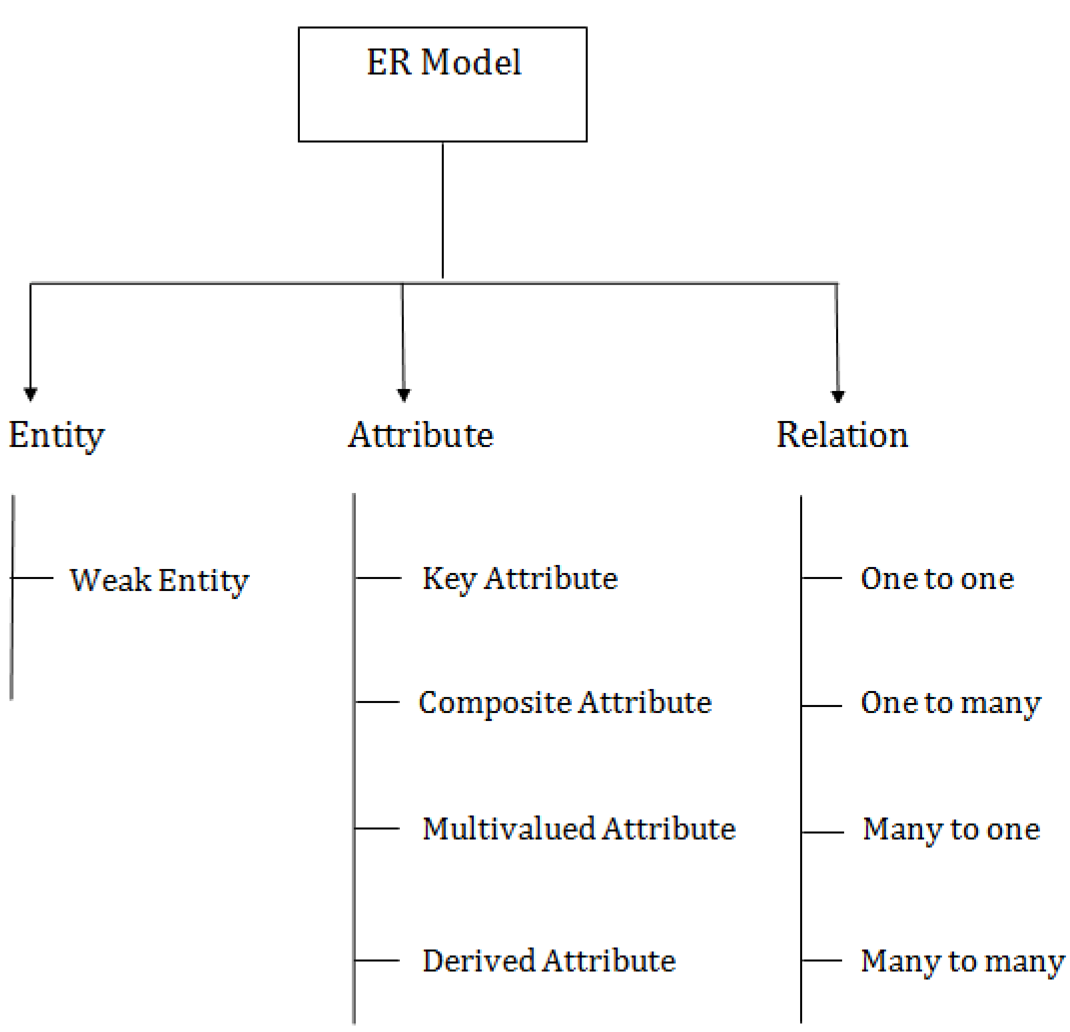
ER model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



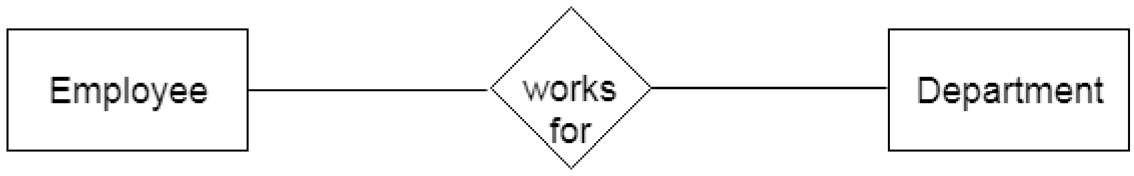
Component of ER Diagram



1. Entity:

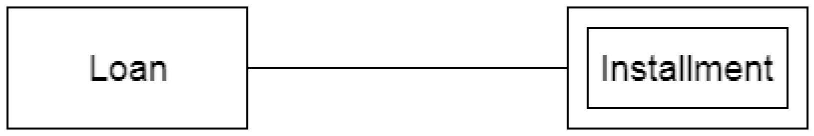
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

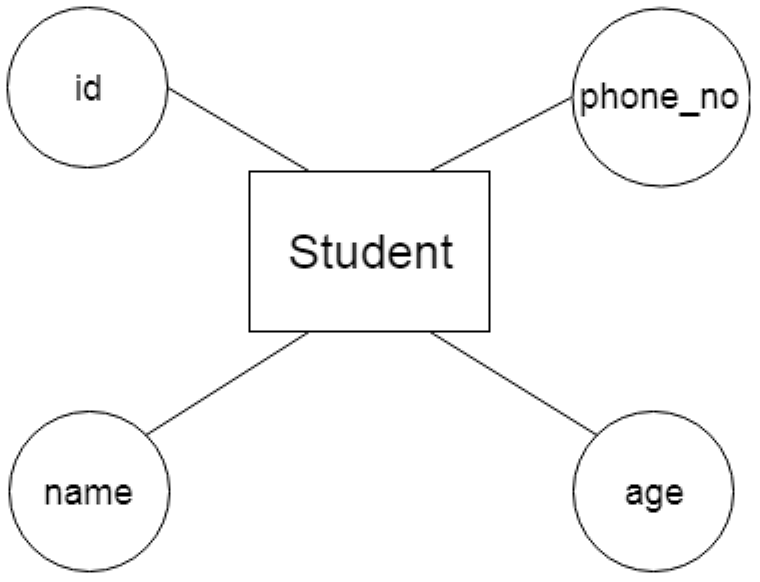
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

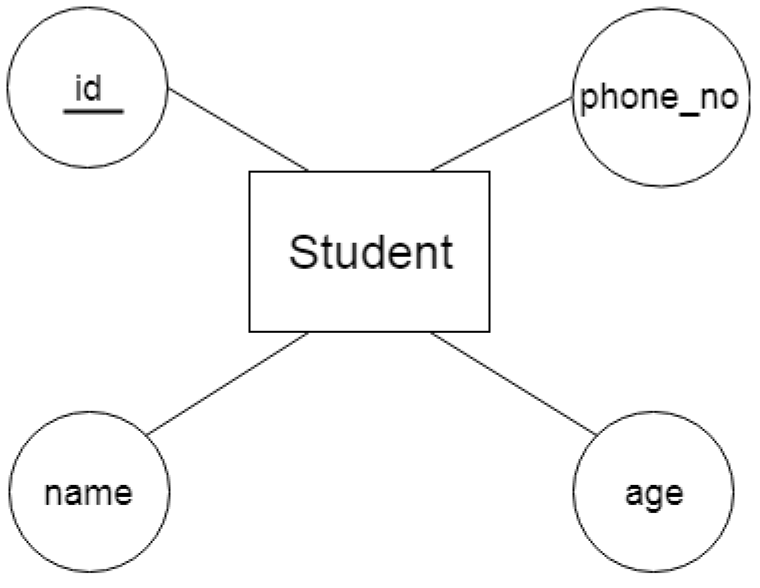
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



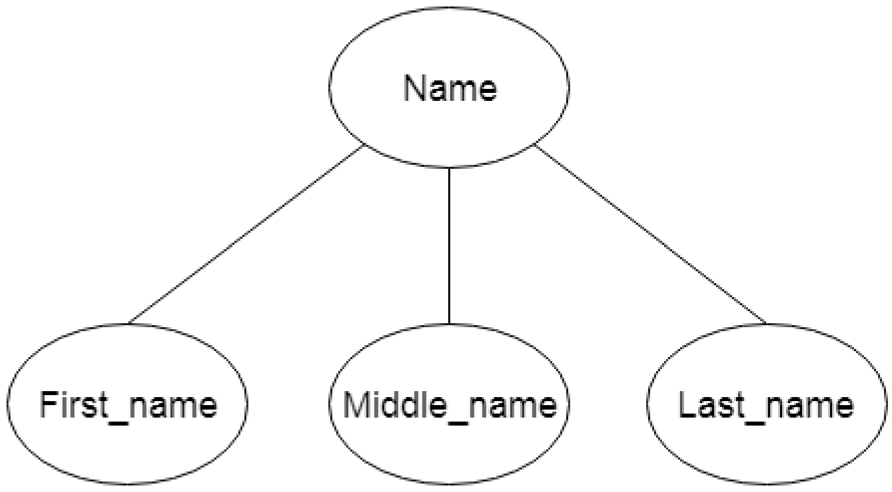
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

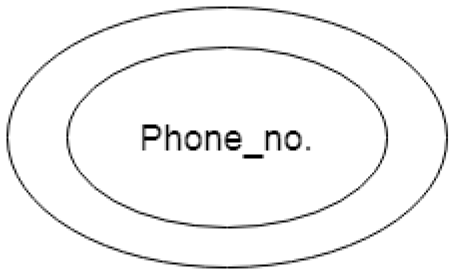
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

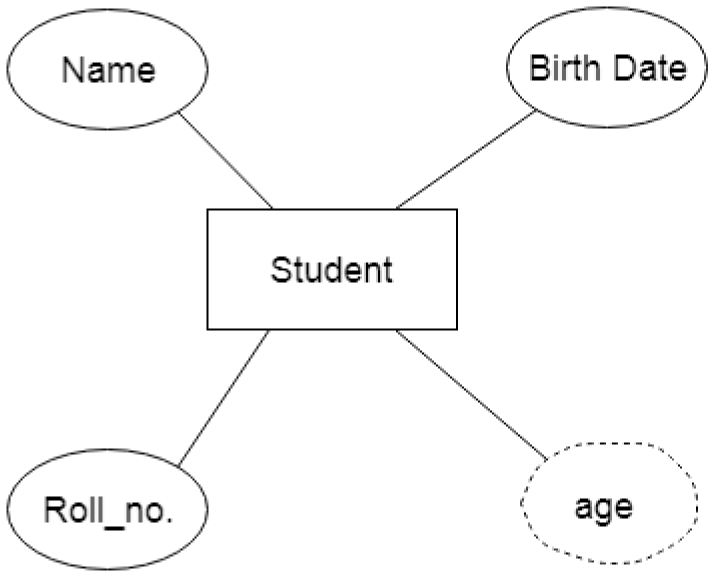
For example, a student can have more than one phone number.



d. Derived Attribute

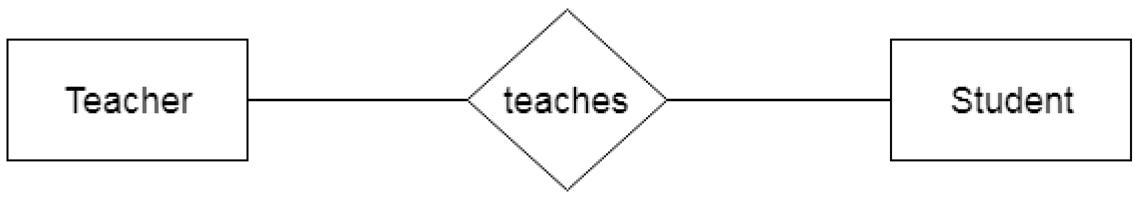
An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



Types of relationship are as follows:

a. One-to-One Relationship

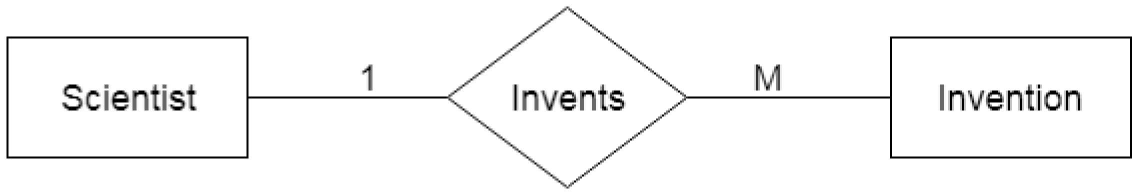
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

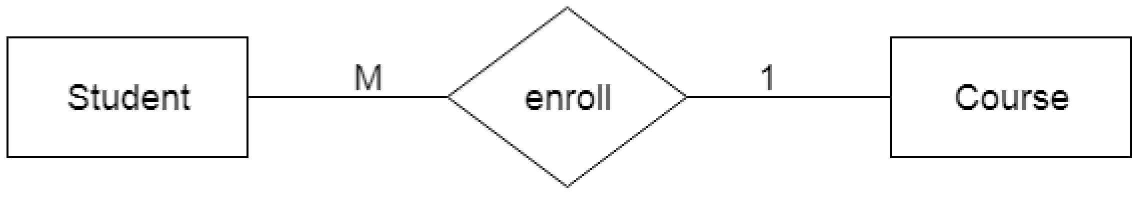
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

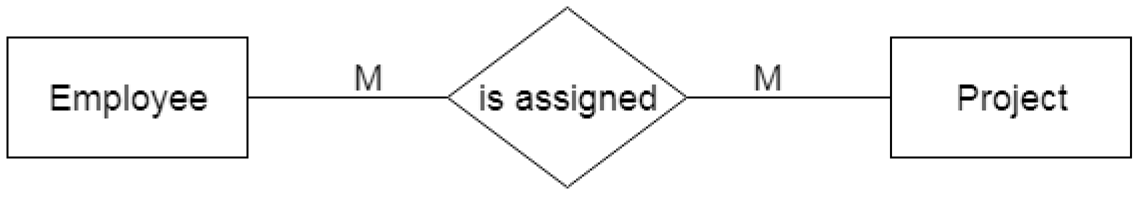
For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.



Notation of ER diagram

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:

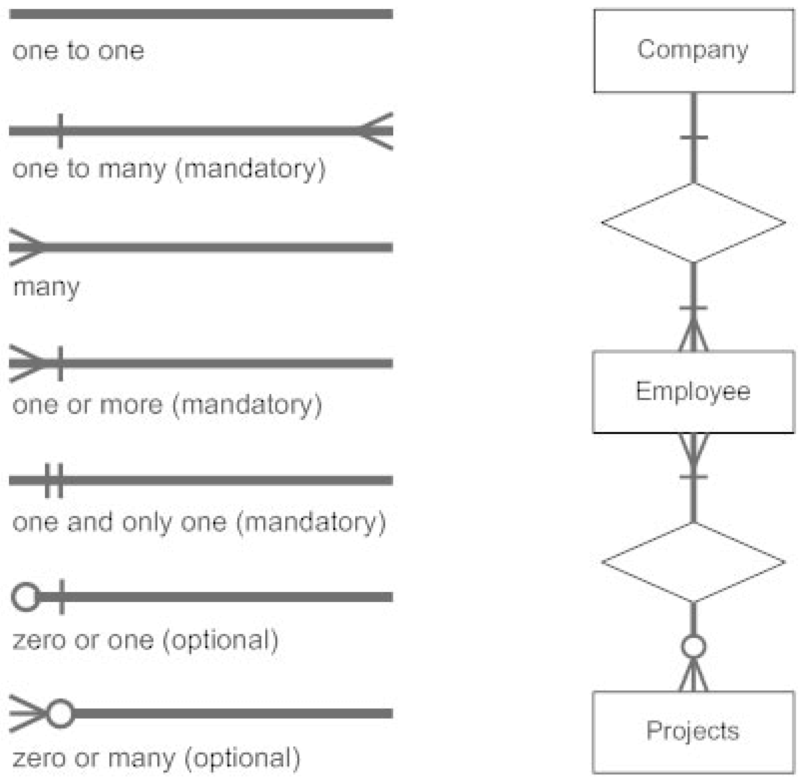


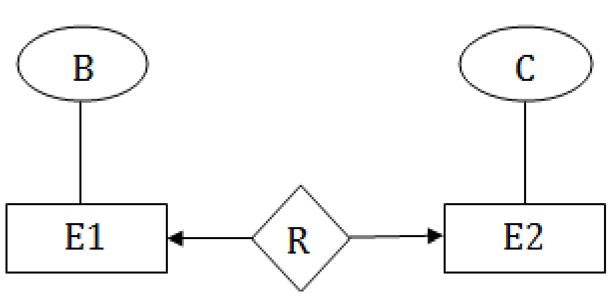
Fig: Notations of ER diagram

Mapping Constraints

- A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
  1. One to one (1:1)
  2. One to many (1:M)
  3. Many to one (M:1)
  4. Many to many (M:M)

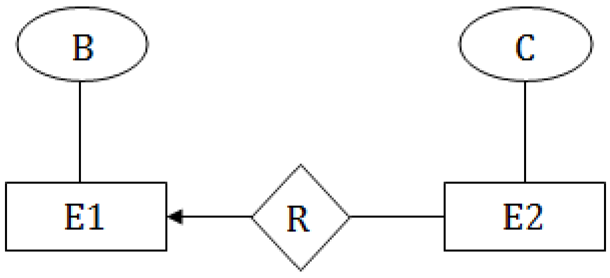
One-to-one

In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



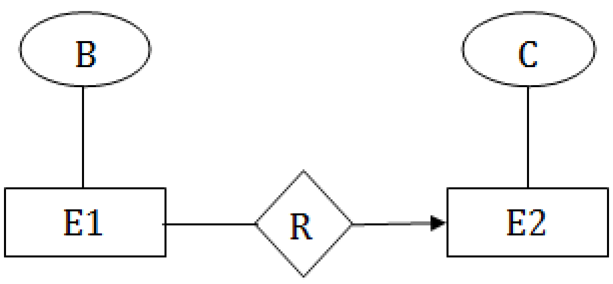
One-to-many

In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.



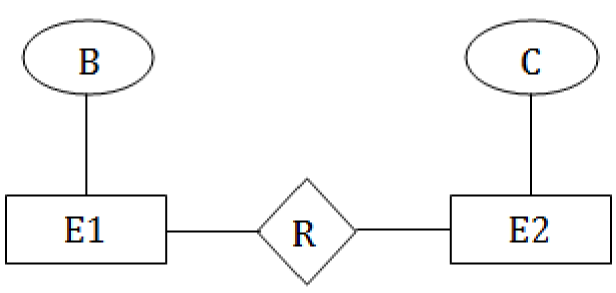
Many-to-one

In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.



Many-to-many

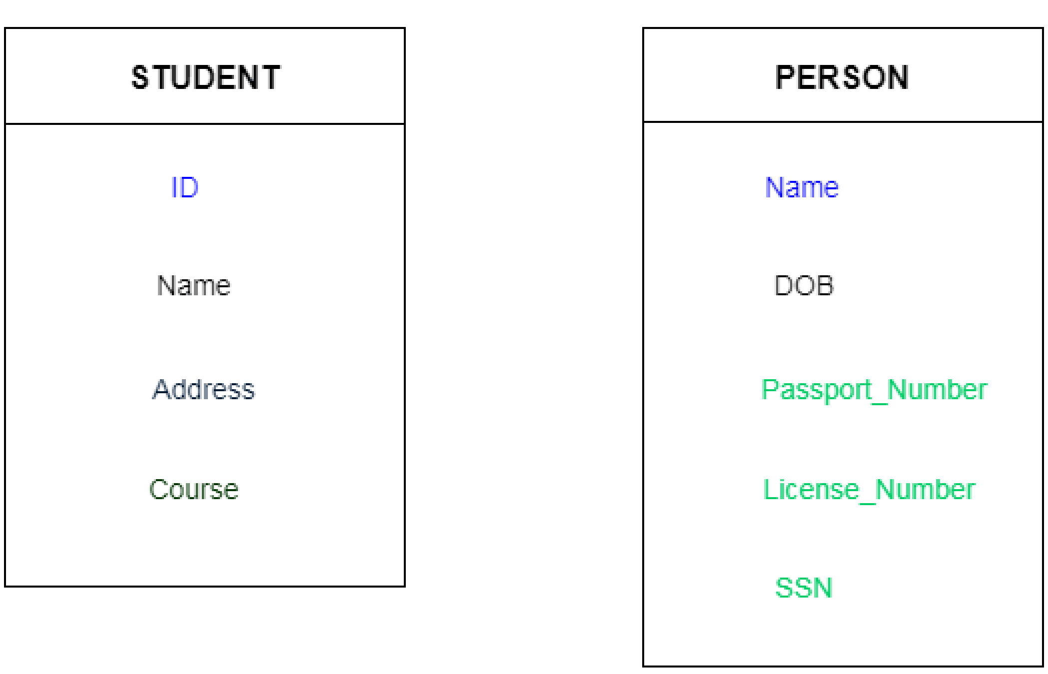
In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.



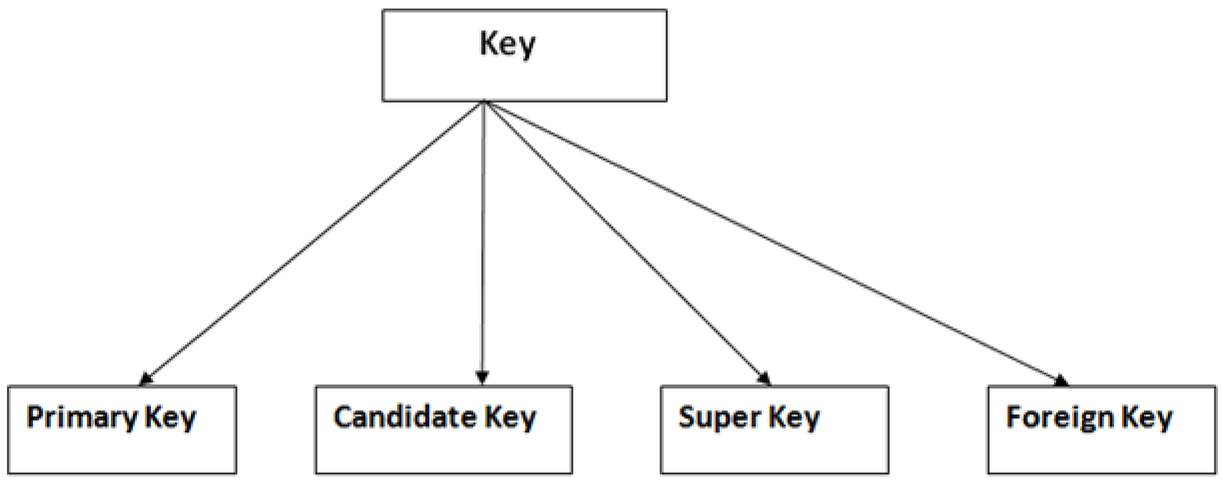
Keys

- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

For example: In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

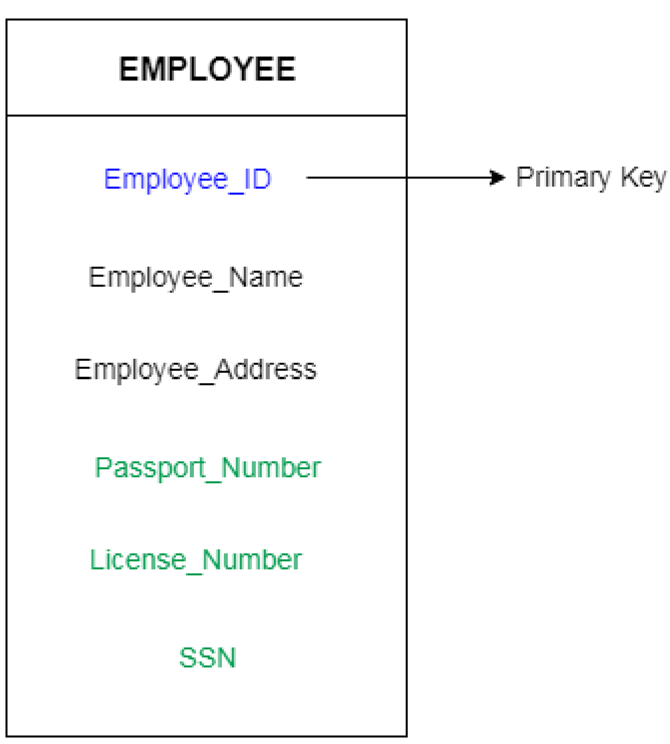


Types of key:



1. Primary key

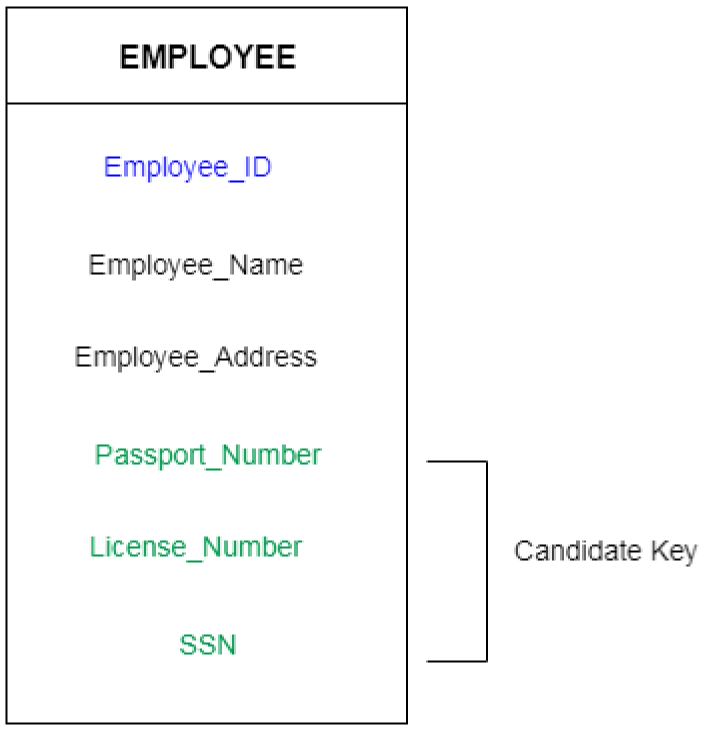
- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
- In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.



2. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



3. Super Key

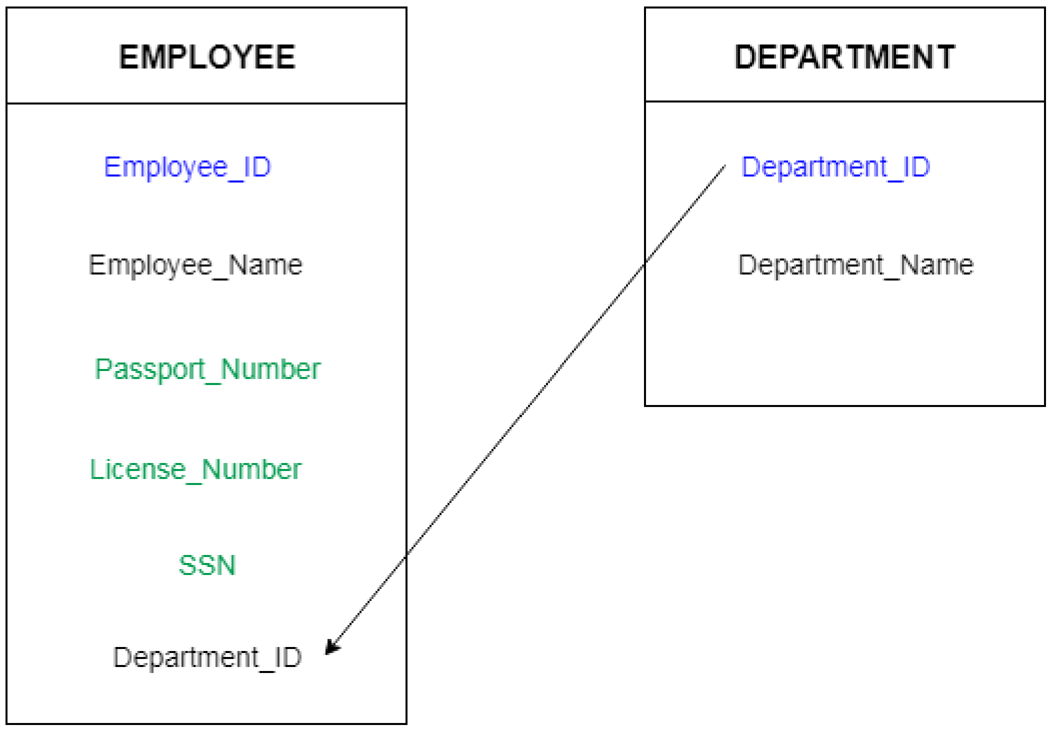
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

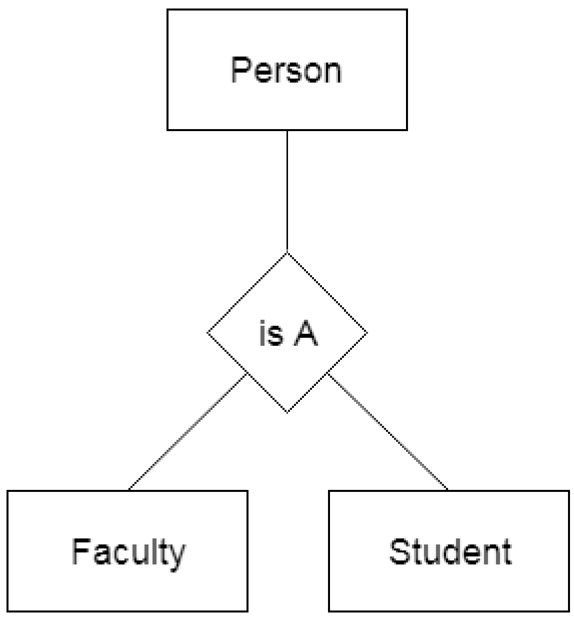


Generalization

- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.

- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

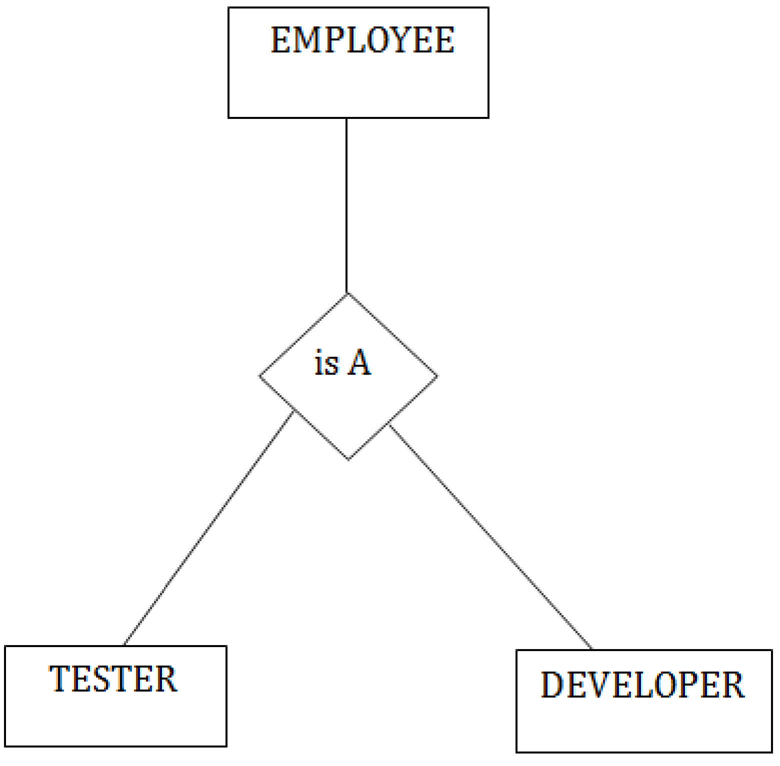
For example, Faculty and Student entities can be generalized and create a higher level entity Person.



Specialization

- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

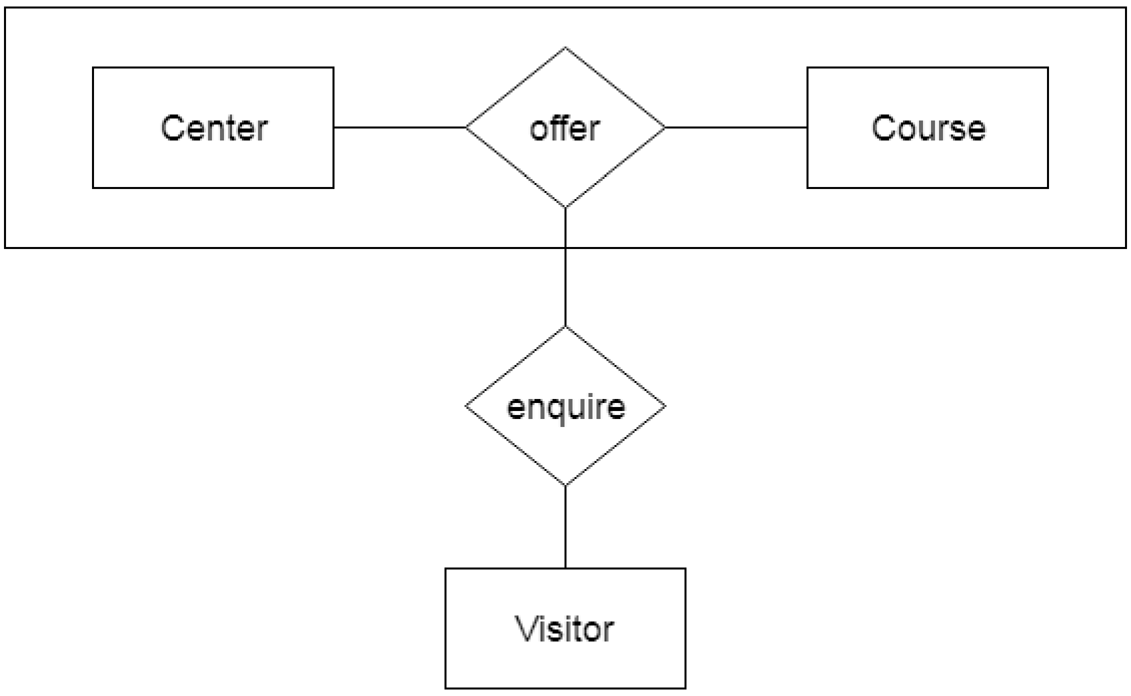
For example: In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

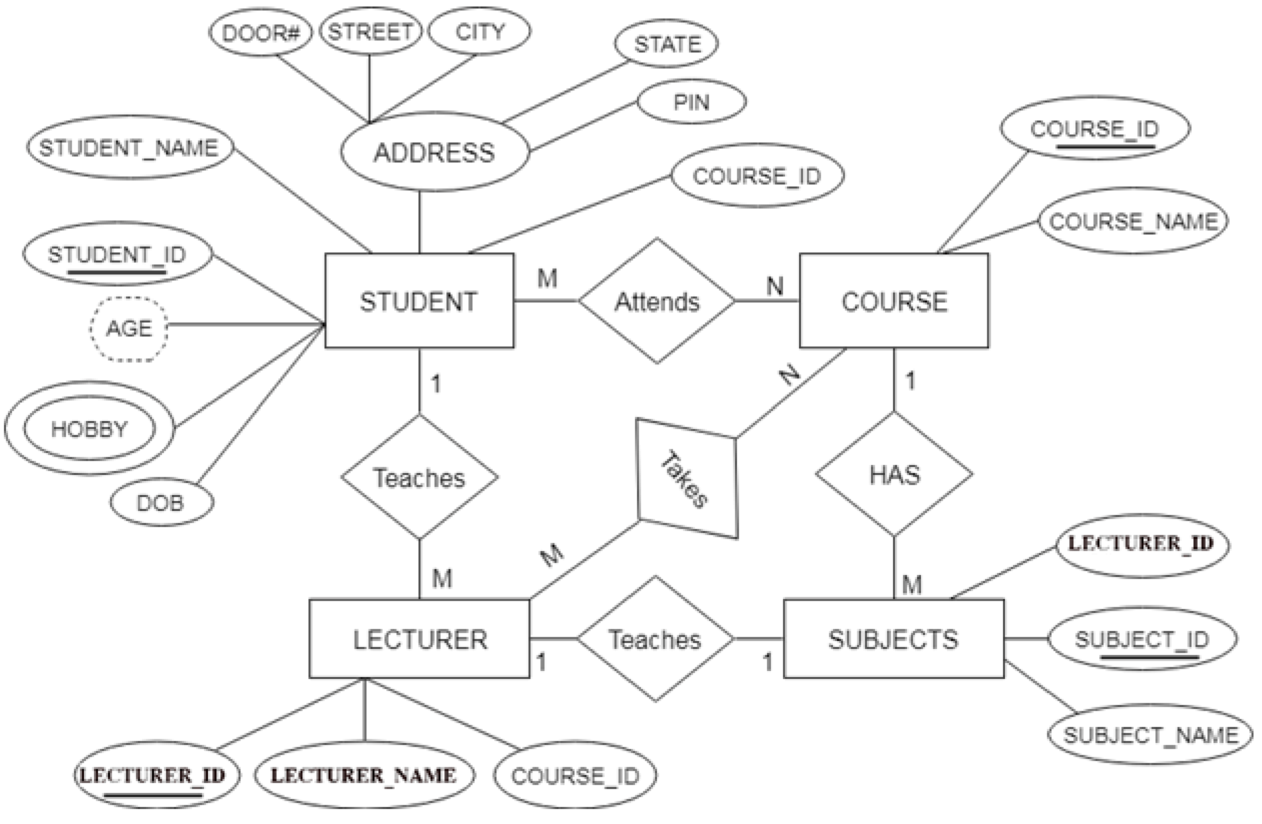


Reduction of ER diagram to Table

The database can be represented using the notations, and these notations can be reduced to a collection of tables.

In the database, every entity set or relationship set can be represented in tabular form.

The ER diagram is given below:



There are some points for converting the ER diagram to the table:

- **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

- **All single-valued attribute becomes a column for the table.**

In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.

- **A key attribute of the entity type represented by the primary key.**

In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.

- **The multivalued attribute is represented by a separate table.**

In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.

- **Composite attribute represented by components.**

In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

- **Derived attributes are not considered in the table.**

In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:

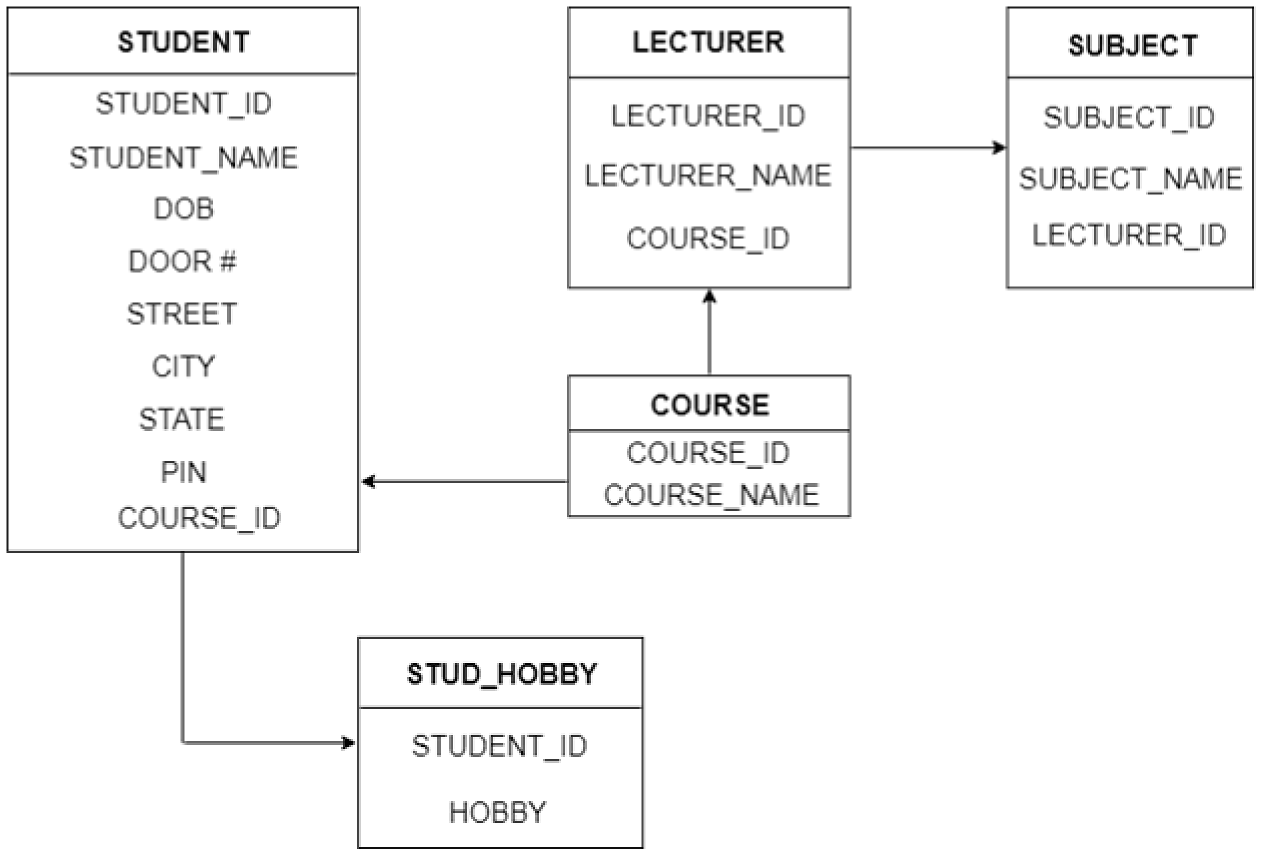


Figure: Table structure

Relationship of higher degree

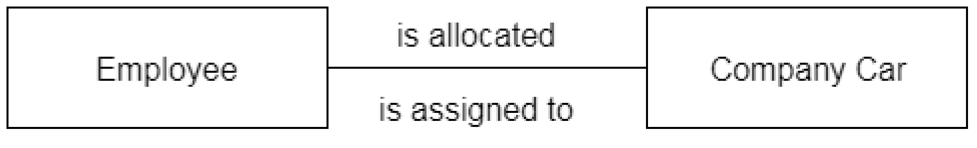
The degree of relationship can be defined as the number of occurrences in one entity that is associated with the number of occurrences in another entity.

There is the three degree of relationship:

1. One-to-one (1:1)
2. One-to-many (1:M)
3. Many-to-many (M:N)

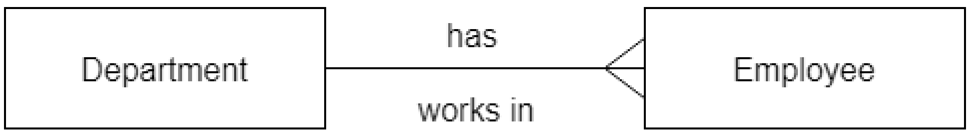
1. One-to-one

- In a one-to-one relationship, one occurrence of an entity relates to only one occurrence in another entity.
- A one-to-one relationship rarely exists in practice.
- **For example:** if an employee is allocated a company car then that car can only be driven by that employee.
- Therefore, employee and company car have a one-to-one relationship.



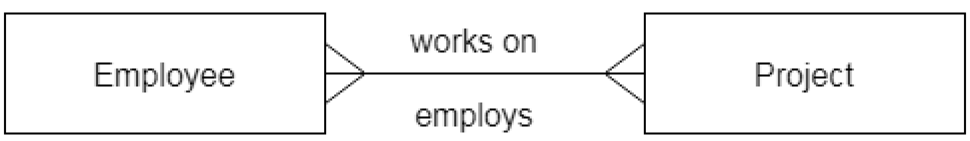
2. One-to-many

- In a one-to-many relationship, one occurrence in an entity relates to many occurrences in another entity.
- **For example:** An employee works in one department, but a department has many employees.
- Therefore, department and employee have a one-to-many relationship.



3. Many-to-many

- In a many-to-many relationship, many occurrences in an entity relate to many occurrences in another entity.
- Same as a one-to-one relationship, the many-to-many relationship rarely exists in practice.
- **For example:** At the same time, an employee can work on several projects, and a project has a team of many employees.
- Therefore, employee and project have a many-to-many relationship.



Relational data Model

Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Domain: It contains a set of atomic values that an attribute can take.

Attribute: It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 9i3988	Delhi	40

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

Properties of Relations

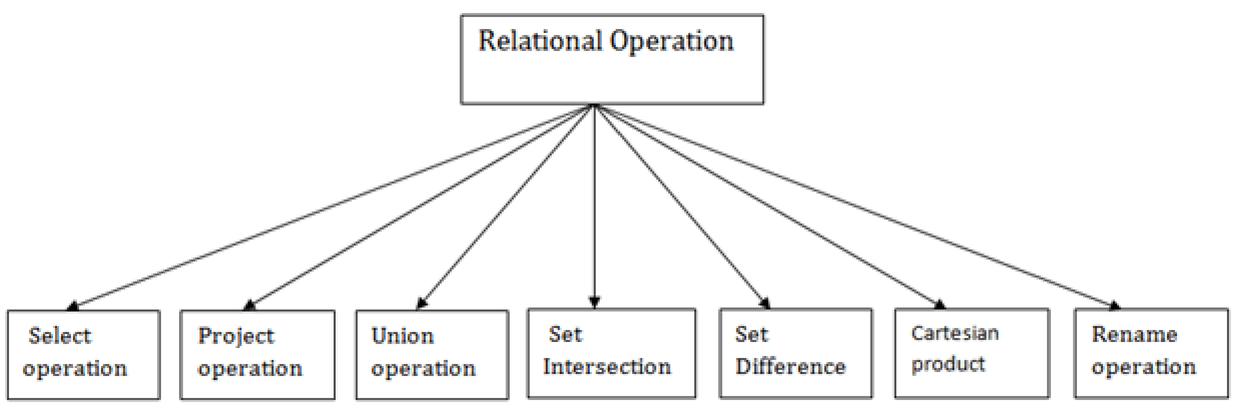
- Name of the relation is distinct from all other relations.

- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

1. Notation: σ p(r)

Where:

σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

1. Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.

- It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson

Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

1. Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians

Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

1. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

Note: Apart from these common operations Relational algebra can be used in Join operations.

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

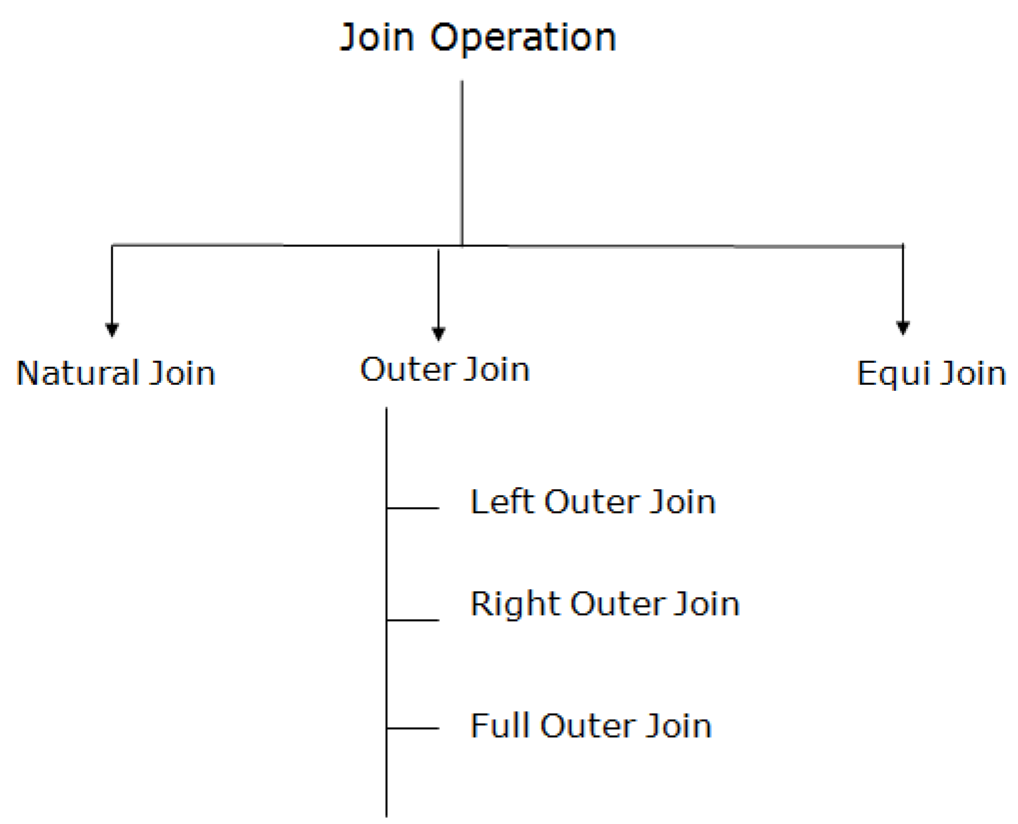
EMP_CODE	SALARY
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE ⋈ SALARY)

Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations:



1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai

Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

a. Left outer join
b. Right outer join
c. Full outer join

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟗ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟗ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER ⋈ PRODUCT

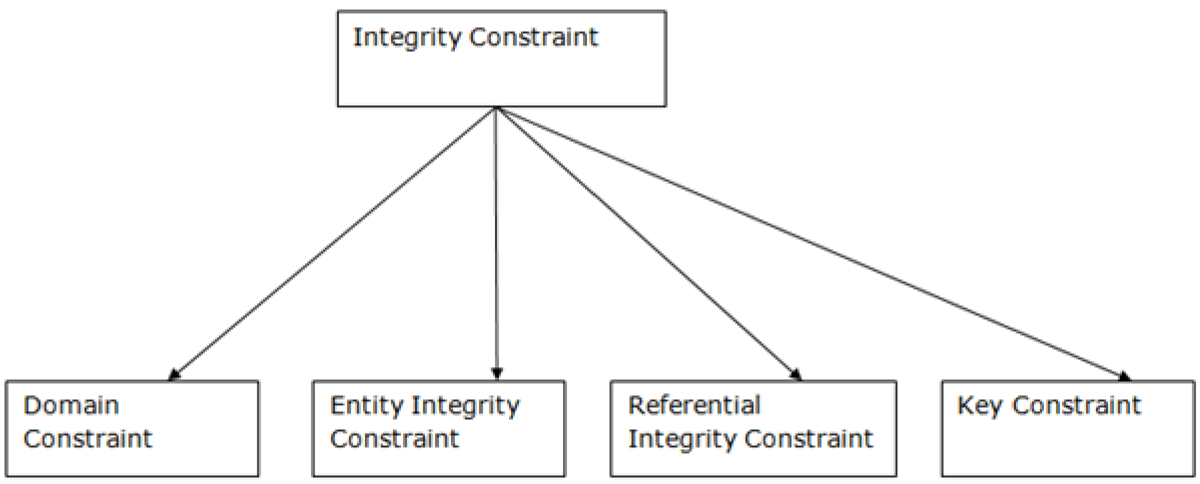
Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:

(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

Primary Key

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

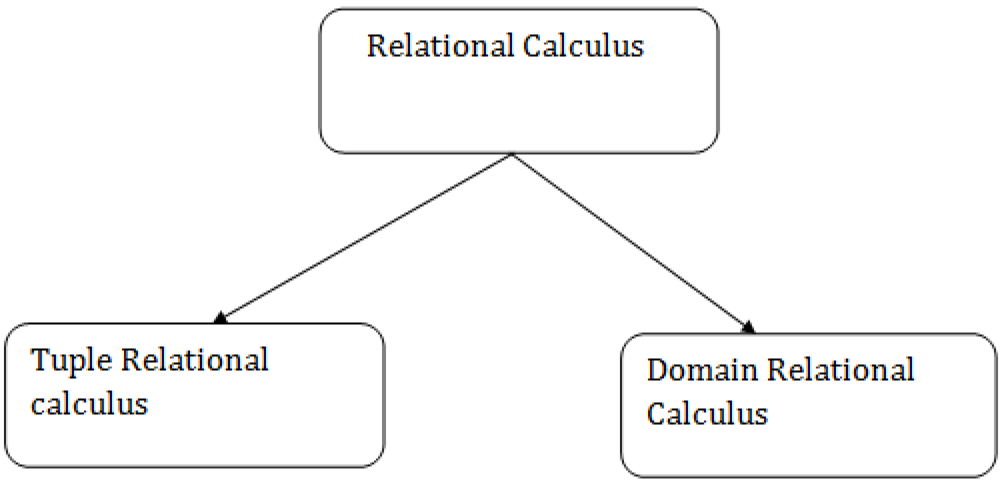
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Relational Calculus

- Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results.
- The relational calculus tells what to do but never explains how to do.

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

- The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.
- The result of the relation can have one or more tuples.

Notation:

1. {T | P (T)}   or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

1. { T.name | Author(T) AND T.article = 'database' }

OUTPUT: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

1. { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

- The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.
- Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ┓ (not).
- It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

Notation:

1. { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1, **a2** are attributes
P stands for formula built by inner attributes

For example:

1. {< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database

Normalization

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

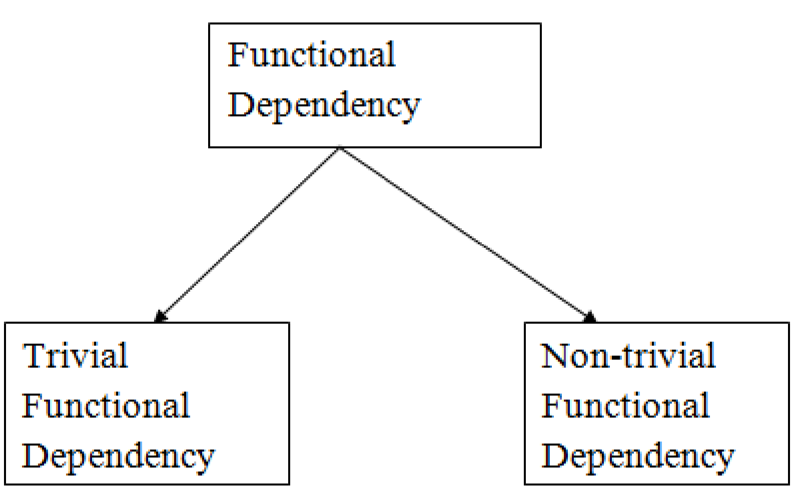
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,

2. Name → DOB

Inference Rule (IR):

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR_1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

1. If X ⊇ Y then X → Y

Example:

1. X = {a, b, c, d, e}
2. Y = {a, b, c}

2. Augmentation Rule (IR_2)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

1. If X → Y then XZ → YZ

Example:

1. For R(ABCD), if A → B then AC → BC

3. Transitive Rule (IR_3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

1. If X → Y and Y → Z then X → Z

4. Union Rule (IR_4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

1. If X → Y and X → Z then X → YZ

Proof:

1. X → Y (given)
2. X → Z (given)
3. X → XY (using IR_2 on 1 by augmentation with X. Where XX = X)
4. XY → YZ (using IR_2 on 2 by augmentation with Y)
5. X → YZ (using IR_3 on 3 and 4)

5. Decomposition Rule (IR_5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

1. If X → YZ then X → Y and X → Z

Proof:

1. $X \rightarrow YZ$ (given)
2. $YZ \rightarrow Y$ (using IR_1 Rule)
3. $X \rightarrow Y$ (using IR_3 on 1 and 2)

6. Pseudo transitive Rule (IR_6)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

1. If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$

Proof:

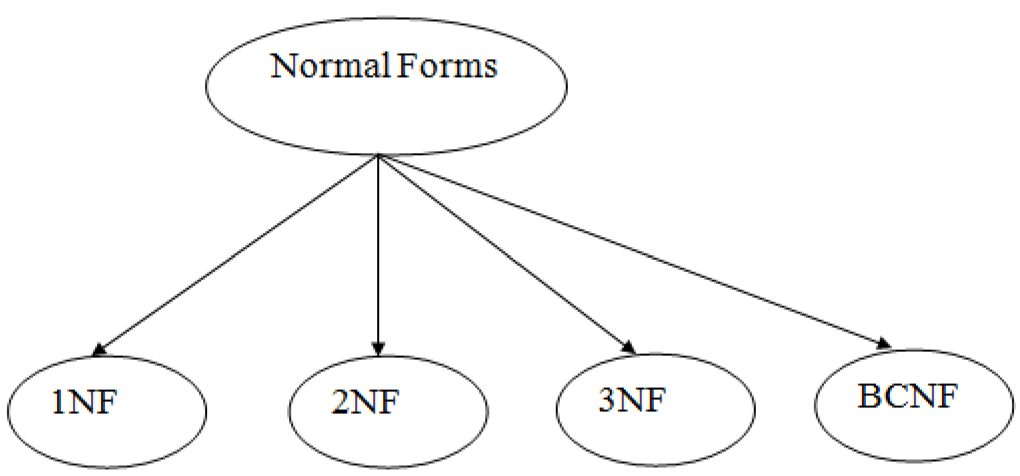
1. $X \rightarrow Y$ (given)
2. $WY \rightarrow Z$ (given)
3. $WX \rightarrow WY$ (using IR_2 on 1 by augmenting with W)
4. $WX \rightarrow Z$ (using IR_3 on 3 and 2)

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 value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

- 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	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

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

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35

83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

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.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010

333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

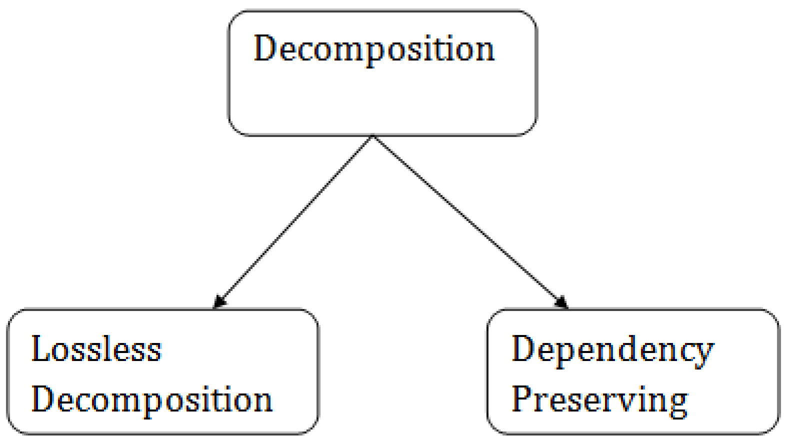
P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales

438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Inclusion Dependency

- Multivalued dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.
- Inclusion dependencies are quite common. They typically show little influence on designing of the database.
- The inclusion dependency is a statement in which some columns of a relation are contained in other columns.
- The example of inclusion dependency is a foreign key. In one relation, the referring relation is contained in the primary key column(s) of the referenced relation.
- Suppose we have two relations R and S which was obtained by translating two entity sets such that every R entity is also an S entity.
- Inclusion dependency would be happen if projecting R on its key attributes yields a relation that is contained in the relation obtained by projecting S on its key attributes.
- In inclusion dependency, we should not split groups of attributes that participate in an inclusion dependency.
- In practice, most inclusion dependencies are key-based that is involved only keys.

Canonical Cover

In the case of updating the database, the responsibility of the system is to check whether the existing functional dependencies are getting violated during the process of updating. In case of a violation of functional dependencies in the new database state, the rollback of the system must take place.

A canonical cover or irreducible a set of functional dependencies FD is a simplified set of FD that has a similar closure as the original set FD.

Extraneous attributes

An attribute of an FD is said to be extraneous if we can remove it without changing the closure of the set of FD.

Example: Given a relational Schema R(A, B, C, D) and set of Function Dependency FD = { B → A, AD → BC, C → ABD }. Find the canonical cover?

Solution: Given FD = { B → A, AD → BC, C → ABD }, now decompose the FD using decomposition rule(Armstrong Axiom).

1. B → A
2. AD → B (using decomposition inference rule on AD → BC)
3. AD → C (using decomposition inference rule on AD → BC)

4. C → A (using decomposition inference rule on C → ABD)
5. C → B (using decomposition inference rule on C → ABD)
6. C → D (using decomposition inference rule on C → ABD)

Now set of FD = { B → A, AD → B, AD → C, C → A, C → B, C → D }

The next step is to find closure of the left side of each of the given FD by including that FD and excluding that FD, if closure in both cases are same then that FD is redundant and we remove that FD from the given set, otherwise if both the closures are different then we do not exclude that FD.

Calculating closure of all FD { B → A, AD → B, AD → C, C → A, C → B, C → D }

1a. Closure B+ = BA using FD = { **B** → **A**, AD → B, AD → C, C → A, C → B, C → D }

1b. Closure B+ = B using FD = { AD → B, AD → C, C → A, C → B, C → D }

From 1 a and 1 b, we found that both the Closure(by including **B** → **A** and excluding **B** → **A**) are not equivalent, hence FD B → A is important and cannot be removed from the set of FD.

2 a. Closure AD+ = ADBC using FD = { B →A, **AD** → **B**, AD → C, C → A, C → B, C → D }

2 b. Closure AD+ = ADCB using FD = { B → A, AD → C, C → A, C → B, C → D }

From 2 a and 2 b, we found that both the Closure (by including **AD** → **B** and excluding **AD** → **B**) are equivalent, hence FD **AD** → **B** is not important and can be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → A, C → B, C → D }

3 a. Closure AD+ = ADCB using FD = { B →A, **AD** → **C**, C → A, C → B, C → D }

3 b. Closure AD+ = AD using FD = { B → A, C → A, C → B, C → D }

From 3 a and 3 b, we found that both the Closure (by including **AD** → **C** and excluding **AD** → **C**) are not equivalent, hence FD AD → C is important and cannot be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → A, C → B, C → D }

4 a. Closure C+ = CABD using FD = { B →A, AD → C, **C** → **A**, C → B, C → D }

4 b. Closure C+ = CBDA using FD = { B → A, AD → C, C → B, C → D }

From 4 a and 4 b, we found that both the Closure (by including **C** → **A** and excluding **C** → **A**) are equivalent, hence FD **C** → **A** is not important and can be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → B, C → D }

5 a. Closure C+ = CBDA using FD = { B →A, AD → C, **C** → **B**, C → D }

5 b. Closure C+ = CD using FD = { B → A, AD → C, C → D }

From 5 a and 5 b, we found that both the Closure (by including **C** → **B** and excluding **C** → **B**) are not equivalent, hence FD **C** → **B** is important and cannot be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → B, C → D }

6 a. Closure C+ = CDBA using FD = { B →A, AD → C, C → B, **C** → **D** }

6 b. Closure C+ = CBA using FD = { B → A, AD → C, C → B }

From 6 a and 6 b, we found that both the Closure(by including **C** → **D** and excluding **C** → **D**) are not equivalent, hence FD **C** → **D** is important and cannot be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → B, C → D }

- Since FD = { B → A, AD → C, C → B, C → D } is resultant FD, now we have checked the redundancy of attribute, since the left side of FD AD → C has two attributes, let's check their importance, i.e. whether they both are important or only one.

Closure AD+ = ADCB using FD = { B →A, **AD** → **C**, C → B, C → D }

Closure A+ = A using FD = { B →A, **AD → C**, C → B, C → D }

Closure D+ = D using FD = { B →A, **AD → C**, C → B, C → D }

Since the closure of AD+, A+, D+ that we found are not all equivalent, hence in FD AD → C, both A and D are important attributes and cannot be removed.

Hence resultant FD = { B → A, AD → C, C → B, C → D } and we can rewrite as

FD = { B → A, AD → C, C → BD } is Canonical Cover of FD = { B → A, AD → BC, C → ABD }.

Example 2: Given a relational Schema R(W, X, Y, Z) and set of Function Dependency FD = { W → X, Y → X, Z → WXY, WY → Z }. Find the canonical cover?

Solution: Given FD = { W → X, Y → X, Z → WXY, WY → Z }, now decompose the FD using decomposition rule(Armstrong Axiom).

1. W → X
2. Y → X
3. Z → W (using decomposition inference rule on Z → WXY)
4. Z → X (using decomposition inference rule on Z → WXY)
5. Z → Y (using decomposition inference rule on Z → WXY)
6. WY → Z

Now set of FD = { W → X, Y → X, WY → Z, Z → W, Z → X, Z → Y }

The next step is to find closure of the left side of each of the given FD by including that FD and excluding that FD, if closure in both cases are same then that FD is redundant and we remove that FD from the given set, otherwise if both the closures are different then we do not exclude that FD.

Calculating closure of all FD { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }

1 a. Closure W+ = WX using FD = { **W → X**, Y → X, Z → W, Z → X, Z → Y, WY → Z }

1 b. Closure W+ = W using FD = { Y → X, Z → W, Z → X, Z → Y, WY → Z }

From 1 a and 1 b, we found that both the Closure (by including **W → X** and excluding **W → X**) are not equivalent, hence FD W → X is important and cannot be removed from the set of FD.

Hence resultant FD = { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }

2 a. Closure Y+ = YX using FD = { W → X, **Y → X**, Z → W, Z → X, Z → Y, WY → Z }

2 b. Closure Y+ = Y using FD = { W → X, Z → W, Z → X, Z → Y, WY → Z }

From 2 a and 2 b we found that both the Closure (by including **Y → X** and excluding **Y → X**) are not equivalent, hence FD Y → X is important and cannot be removed from the set of FD.

Hence resultant FD = { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }

3 a. Closure Z+ = ZWXY using FD = { W → X, Y → X, **Z → W**, Z → X, Z → Y, WY → Z }

3 b. Closure Z+ = ZXY using FD = { W → X, Y → X, Z → X, Z → Y, WY → Z }

From 3 a and 3 b, we found that both the Closure (by including **Z → W** and excluding **Z → W**) are not equivalent, hence FD Z → W is important and cannot be removed from the set of FD.

Hence resultant FD = { W → X, Y → X, Z → W, Z → X, Z → Y, WY → Z }

4 a. Closure Z+ = ZXWY using FD = { W → X, Y → X, Z → W, **Z → X**, Z → Y, WY → Z }

4 b. Closure Z+ = ZWYX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

From 4 a and 4 b, we found that both the Closure (by including **Z → X** and excluding **Z → X**) are equivalent, hence FD Z → X is **not** important and can be removed from the set of FD.

Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

5 a. Closure Z+ = ZYWX using FD = { W → X, Y → X, Z → W, **Z** → **Y**, WY → Z }

5 b. Closure Z+ = ZWX using FD = { W → X, Y → X, Z → W, WY → Z }

From 5 a and 5 b, we found that both the Closure (by including **Z** → **Y** and excluding **Z** → **Y**) are not equivalent, hence FD Z → X is important and cannot be removed from the set of FD.

Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

6 a. Closure WY+ = WYZX using FD = { W → X, Y → X, Z → W, Z → Y, **WY** → **Z** }

6 b. Closure WY+ = WYX using FD = { W → X, Y → X, Z → W, Z → Y }

From 6 a and 6 b, we found that both the Closure (by including **WY** → **Z** and excluding **WY** → **Z**) are not equivalent, hence FD WY → Z is important and cannot be removed from the set of FD.

Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Since FD = { W → X, Y → X, Z → W, Z → Y, WY → Z } is resultant FD now, we have checked the redundancy of attribute, since the left side of FD WY → Z has two attributes at its left, let's check their importance, i.e. whether they both are important or only one.

Closure WY+ = WYZX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Closure W+ = WX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Closure Y+ = YX using FD = { W → X, Y → X, Z → W, Z → Y, WY → Z }

Since the closure of WY+, W+, Y+ that we found are not all equivalent, hence in FD WY → Z, both W and Y are important attributes and cannot be removed.

Hence resultant FD = { W → X, Y → X, Z → W, Z → Y, WY → Z } and we can rewrite as:

FD = { W → X, Y → X, Z → WY, WY → Z } is Canonical Cover of FD = { W → X, Y → X, Z → WXY, WY → Z }.

Example 3: Given a relational Schema R(V, W, X, Y, Z) and set of Function Dependency FD = { V → W, VW → X, Y → VXZ }. Find the canonical cover?

Solution: Given FD = { V → W, VW → X, Y → VXZ }. now decompose the FD using decomposition rule (Armstrong Axiom).

1. V → W
2. VW → X
3. Y → V (using decomposition inference rule on Y → VXZ)
4. Y → X (using decomposition inference rule on Y → VXZ)
5. Y → Z (using decomposition inference rule on Y → VXZ)

Now set of FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.

The next step is to find closure of the left side of each of the given FD by including that FD and excluding that FD, if closure in both cases are same then that FD is redundant and we remove that FD from the given set, otherwise if both the closures are different then we do not exclude that FD.

Calculating closure of all FD { V → W, VW → X, Y → V, Y → X, Y → Z }.

1 a. Closure V+ = VWX using FD = {**V** → **W**, VW → X, Y → V, Y → X, Y → Z}

1 b. Closure V+ = V using FD = {VW → X, Y → V, Y → X, Y → Z }

From 1 a and 1 b, we found that both the Closure(by including **V** → **W** and excluding **V** → **W**) are not equivalent, hence FD V → W is important and cannot be removed from the set of FD.

Hence resultant FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.

2 a. Closure VW+ = VWX using FD = { V → W, **VW** → **X**, Y → V, Y → X, Y → Z }

2 b. Closure VW+ = VW using FD = { V → W, Y → V, Y → X, Y → Z }

From 2 a and 2 b, we found that both the Closure(by including **VW** → **X** and excluding **VW** → **X**) are not equivalent, hence FD VW → X is important and cannot be removed from the set of FD.

Hence resultant FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.

3 a. Closure Y+ = YVXZW using FD = { V → W, VW → X, **Y → V**, Y → X, Y → Z }

3 b. Closure Y+ = YXZ using FD = { V → W, VW → X, Y → X, Y → Z }

From 3 a and 3 b, we found that both the Closure(by including **Y → V** and excluding **Y → V**) are not equivalent, hence FD Y → V is important and cannot be removed from the set of FD.

Hence resultant FD = { V → W, VW → X, Y → V, Y → X, Y → Z }.

4 a. Closure Y+ = YXVZW using FD = { V → W, VW → X, Y → V, **Y → X**, Y → Z }

4 b. Closure Y+ = YVZWX using FD = { V → W, VW → X, Y → V, Y → Z }

From 4 a and 4 b, we found that both the Closure(by including **Y → X** and excluding **Y → X**) are equivalent, hence FD Y → X is **not** important and can be removed from the set of FD.

Hence resultant FD = { V → W, VW → X, Y → V, Y → Z }.

5 a. Closure Y+ = YZVWX using FD = { V → W, VW → X, Y → V, **Y → Z** }

5 b. Closure Y+ = YVWX using FD = { V → W, VW → X, Y → V }

From 5 a and 5 b, we found that both the Closure(by including **Y → Z** and excluding **Y → Z**) are not equivalent, hence FD Y → Z is important and cannot be removed from the set of FD.

Hence resultant FD = { V → W, VW → X, Y → V, Y → Z }.

Since FD = { V → W, VW → X, Y → V, Y → Z } is resultant FD now, we have checked the redundancy of attribute, since the left side of FD VW → X has two attributes at its left, let's check their importance, i.e. whether they both are important or only one.

Closure VW+ = VWX using FD = { V → W, VW → X, Y → V, Y → Z }

Closure V+ = VWX using FD = { V → W, VW → X, Y → V, Y → Z }

Closure W+ = W using FD = { V → W, VW → X, Y → V, Y → Z }

Since the closure of VW+, V+, W+ we found that all the Closures of VW and V are equivalent, hence in FD VW → X, W is not at all an important attribute and can be removed.

Hence resultant FD = { V → W, V → X, Y → V, Y → Z } and we can rewrite as

FD = { V → WX, Y → VZ } is Canonical Cover of FD = { V → W, VW → X, Y → VXZ }.

CONCLUSION: From the above three examples we conclude that canonical cover / irreducible set of functional dependency follows the following steps, which we need to follow while calculating Canonical Cover.

STEP 1: For a given set of FD, decompose each FD using decomposition rule (Armstrong Axiom) if the right side of any FD has more than one attribute.

STEP 2: Now make a new set of FD having all decomposed FD.

STEP 3: Find closure of the left side of each of the given FD by including that FD and excluding that FD, if closure in both cases are same then that FD is redundant and we remove that FD from the given set, otherwise if both the closures are different then we do not exclude that FD.

STEP 4: Repeat step 4 till all the FDs in FD set are complete.

STEP 5: After STEP 4, find resultant FD = { B → A, AD → C, C → B, C → D } which are not redundant.

STEP 6: Check redundancy of attribute, by selecting those FD's from FD sets which are having more than one attribute on its left, let's an FD AD → C has two attributes at its left, let's check their importance, i.e. whether they both are important or only one.

STEP 6 a: Find Closure AD+

STEP 6 b: Find Closure A+

STEP 6 c: Find Closure D+

Compare Closure of STEP (6a, 6b, 6c) if the closure of AD+, A+, D+ are not equivalent, hence in FD AD → C, both A and D are important attributes and cannot be removed, otherwise, we remove the redundant attribute.

Transaction Processing

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)
2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. 1. R(X);
2. 2. X = X - 500;
3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A-100 Write(A)	Read(B) Y:= Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

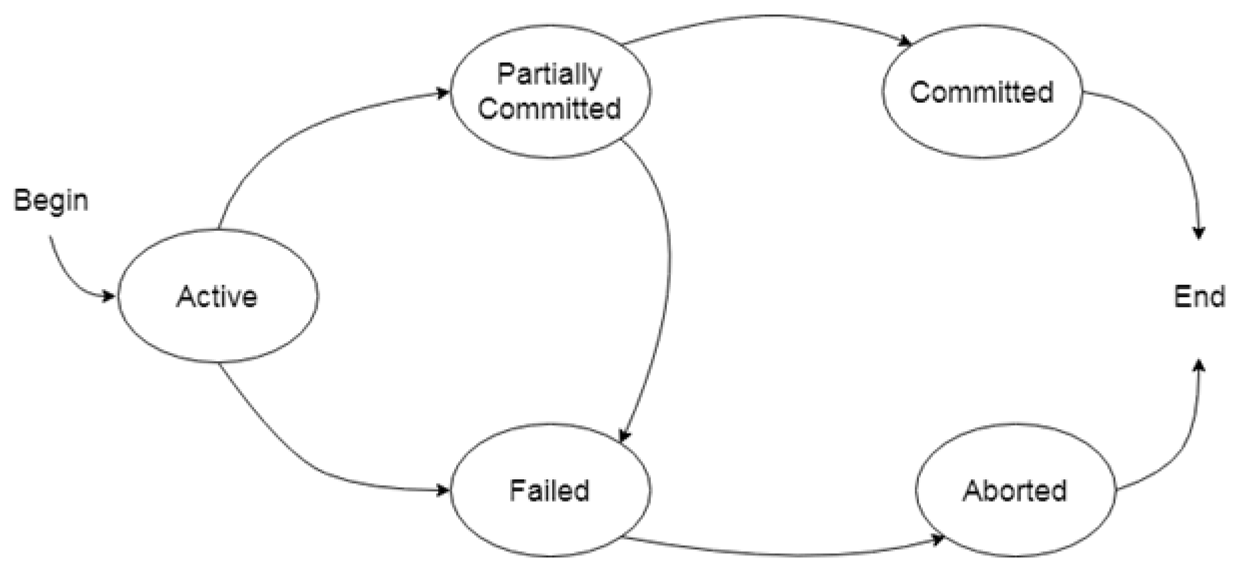
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

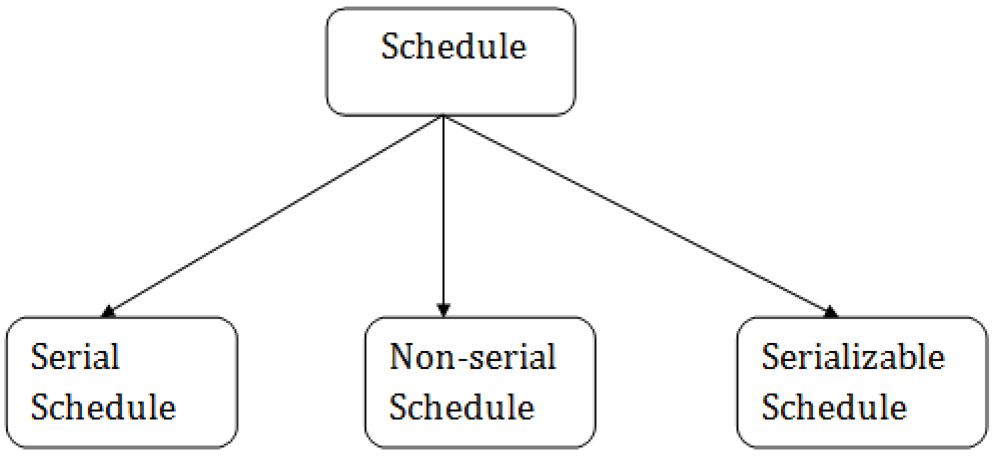
- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T1 which was followed by all the operations of T2.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-serial Schedule

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

(a)

Time

T1	T2

read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);	

	read(A);
	A := A + M;
	write(A);

Schedule A

(b)

Time ↓

T1	T2

	read(A);
	A := A + M;
	write(A);

read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);	

Schedule B

(c)

Time ↓

T1	T2
read(A); A := A – N;	
	read(A); A := A + M;
write(A); read(B);	
	write(A);
B := B + N; write(B);	

Schedule C

(d)

Time ↓

T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Here,

Schedule A and Schedule B are serial schedule.

Schedule C and Schedule D are Non-serial schedule.

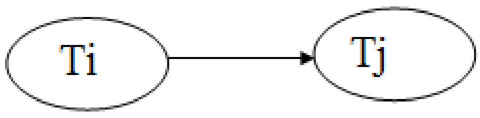
Testing of Serializability

Serialization Graph is used to test the Serializability of a schedule.

Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Precedence graph for Schedule S



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

For example:

Time

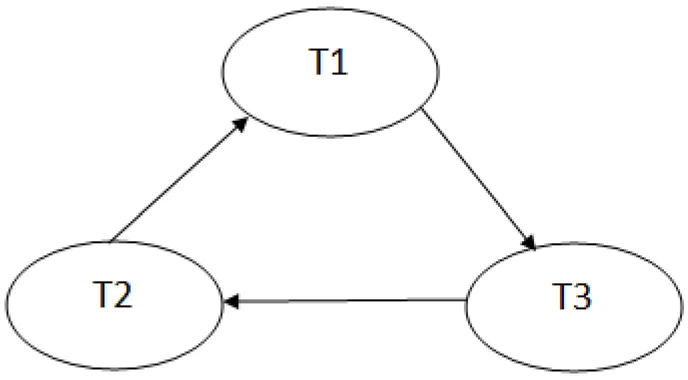
T1	T2	T3
Read(A)		
	Read(B)	
A:= f_1(A)		
		Read(C)
	B:= f_2(B)	
	Write(B)	
		C:= f_3(C)
		Write(C)
Write(A)		
		Read(B)
	Read(A)	
	A:= f_4(A)	
Read(C)		
	Write(A)	
C:= f_5(C)		
Write(C)		
		B:= f_6(B)
		Write(B)

Schedule S1

Explanation:

Read(A): In T1, no subsequent writes to A, so no new edges
Read(B): In T2, no subsequent writes to B, so no new edges
Read(C): In T3, no subsequent writes to C, so no new edges
Write(B): B is subsequently read by T3, so add edge T2 → T3
Write(C): C is subsequently read by T1, so add edge T3 → T1
Write(A): A is subsequently read by T2, so add edge T1 → T2
Write(A): In T2, no subsequent reads to A, so no new edges
Write(C): In T1, no subsequent reads to C, so no new edges
Write(B): In T3, no subsequent reads to B, so no new edges

Precedence graph for schedule S1:



The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.

Time

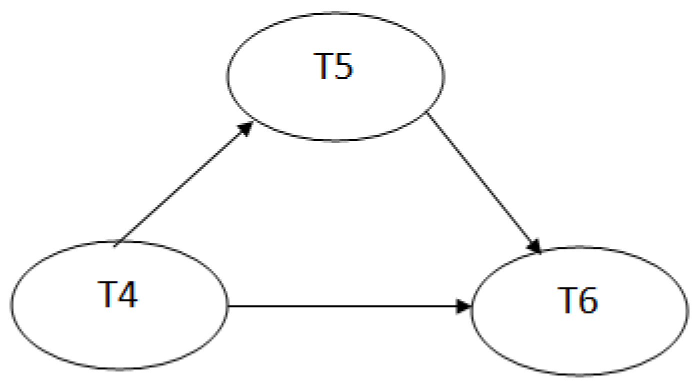
T4	T5	T6
Read(A)		
A:= f1(A)		
Read(C)		
Write(A)		
A:= f2(C)		
	Read(B)	
Write(C)		
	Read(A)	
		Read(C)
	B:= f3(B)	
	Write(B)	
		C:= f4(C)
		Read(B)
		Write(C)
	A:=f5(A)	
	Write(A)	
		B:= f6(B)
		Write(B)

Schedule S2

Explanation:

Read(A): In T4,no subsequent writes to A, so no new edges
Read(C): In T4, no subsequent writes to C, so no new edges
Write(A): A is subsequently read by T5, so add edge T4 → T5
Read(B): In T5,no subsequent writes to B, so no new edges
Write(C): C is subsequently read by T6, so add edge T4 → T6
Write(B): A is subsequently read by T6, so add edge T5 → T6
Write(C): In T6, no subsequent reads to C, so no new edges
Write(A): In T5, no subsequent reads to A, so no new edges
Write(B): In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

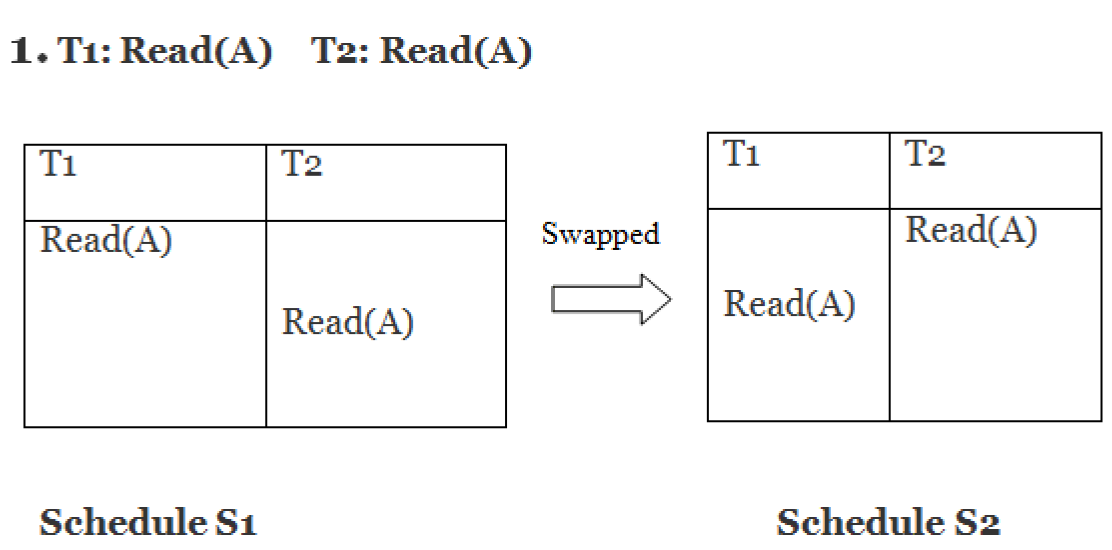
Conflicting Operations

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.



Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Swapped ⇨

T1	T2
	Write(A)
Read(A)	

Schedule S1 **Schedule S2**

Here, S1 ≠ S2. That means it is conflict.

Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:

Non-serial schedule

T1	T2
Read(A) Write(A)	
	Read(A) Write(A)
Read(B) Write(B)	
	Read(B) Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Schedule S2

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A) Write(A) Read(B) Write(B)	
	Read(A) Write(A) Read(B) Write(B)

Since, S1 is conflict serializable.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3
Read(A)		
Write(A)		
	Write(A)	
		Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table 1 shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table 2 shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table 3 shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Failure Classification

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure
2. System crash
3. Disk failure

1. Transaction failure

The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.
2. **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The system aborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

3. System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.

Fail-stop assumption: In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure

4. It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
5. Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.
  1. <Tn, Start>
- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.
  1. <Tn, City, 'Noida', 'Bangalore' >
- When the transaction is finished, then it writes another log to indicate the end of the transaction.
  1. <Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

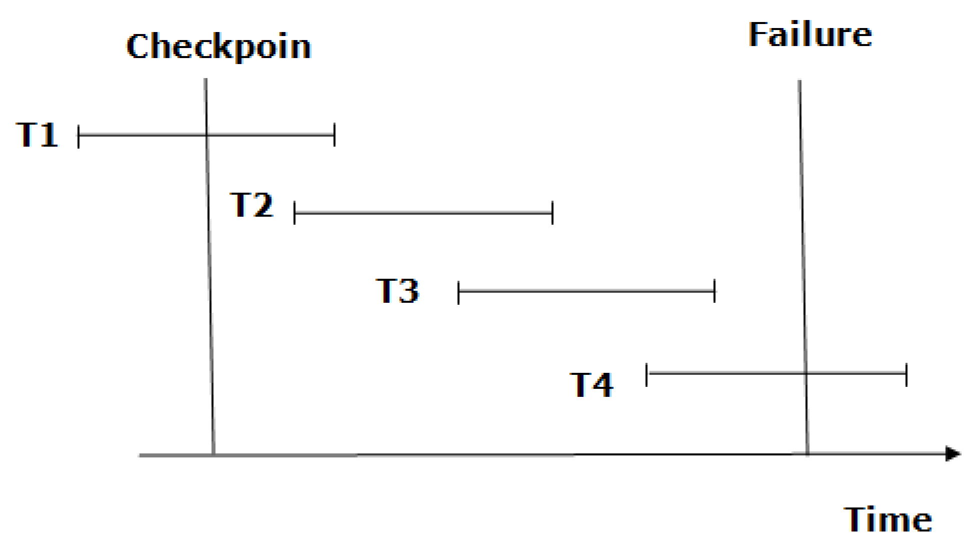
1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

Checkpoint

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

Deadlock in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

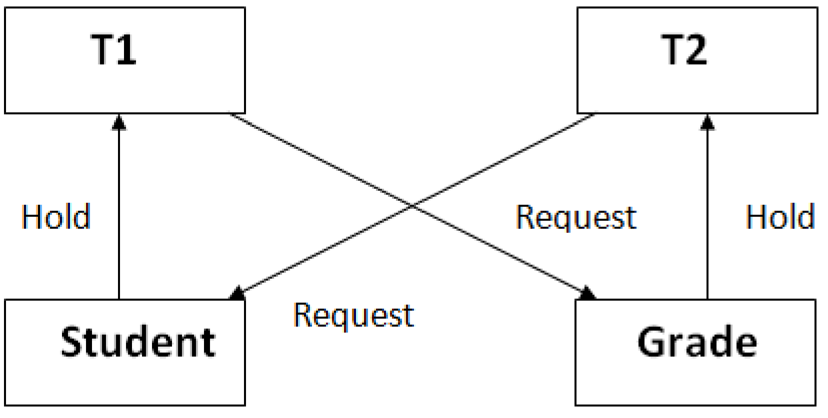


Figure: Deadlock in DBMS

Deadlock Avoidance

- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

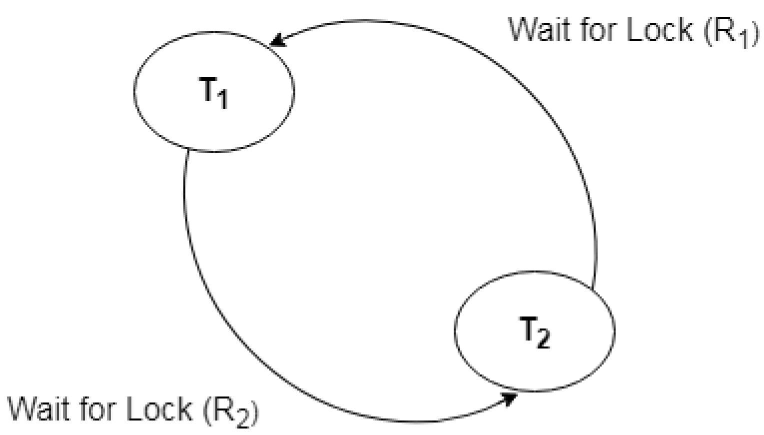
Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

The wait for a graph for the above scenario is shown below:



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

Concurrency Control

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.

- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

We will understand and discuss each protocol one by one in our next sections.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

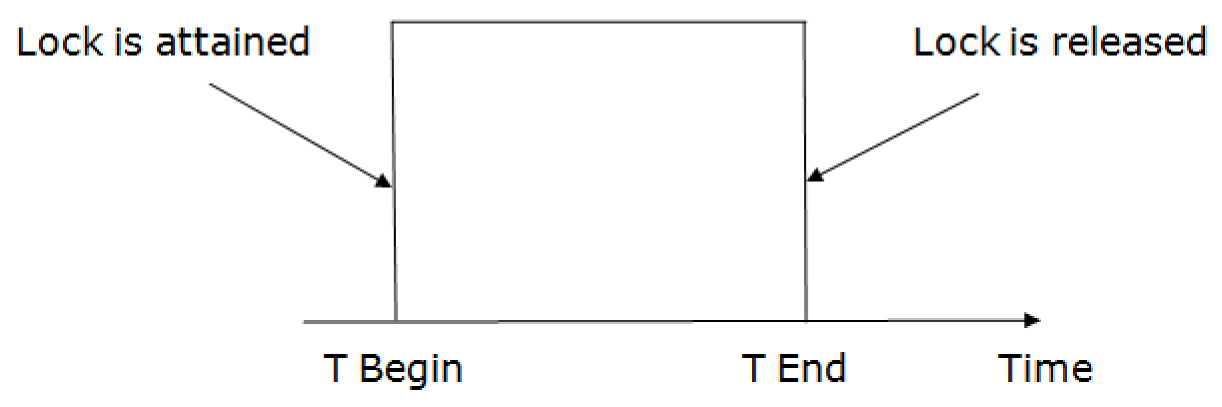
There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

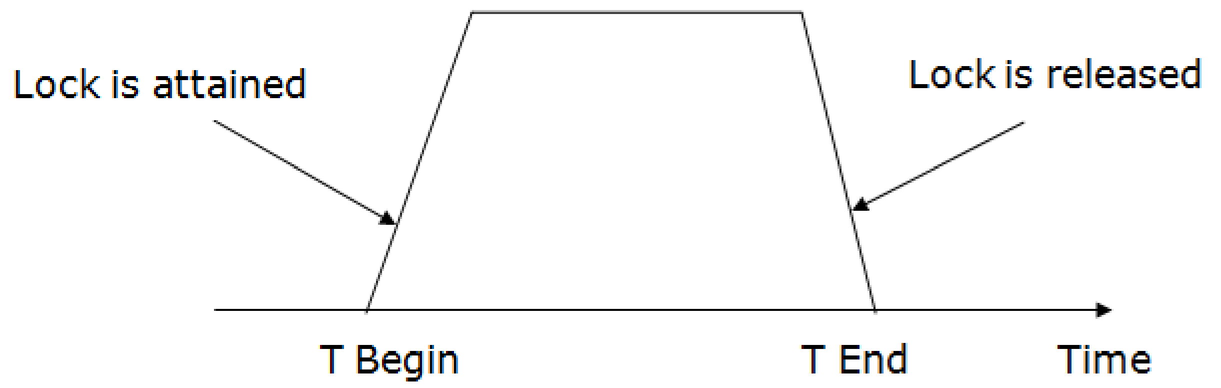
2. Pre-claiming Lock Protocol

- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

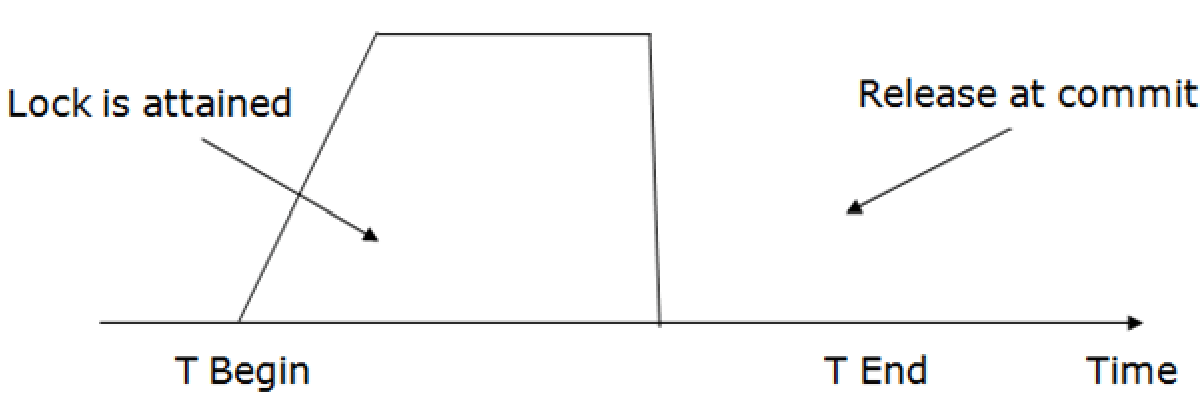
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

Transaction T2:

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.

3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

Thomas write Rule

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

T1	T2
R(A)	
	W(A) Commit
W(A) Commit	

Figure: A Serializable Schedule that is not Conflict Serializable

In the above figure, T1's read and precedes T1's write of the same data item. This schedule does not conflict serializable.

Thomas write rule checks that T2's write is never seen by any transaction. If we delete the write operation in transaction T2, then conflict serializable schedule can be obtained which is shown in below figure.

T1	T2
R(A)	Commit
W(A) Commit	

Figure: A Conflict Serializable Schedule

Multiple Granularity

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
    1. Database
    2. Area
    3. File
    4. Record

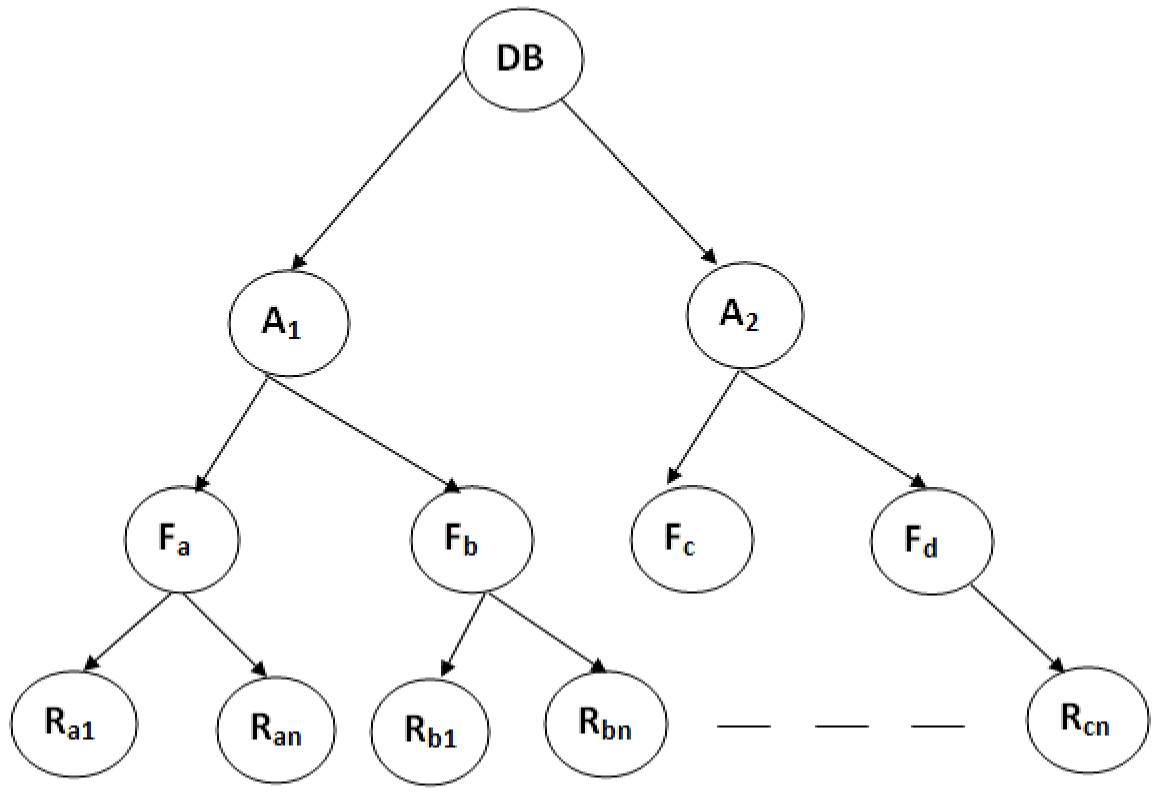


Figure: Multi Granularity tree Hierarchy

In this example, the highest level shows the entire database. The levels below are file, record, and fields.

There are three additional lock modes with multiple granularity:

Intention Mode Lock

Intention-shared (IS): It contains explicit locking at a lower level of the tree but only with shared locks.

Intention-Exclusive (IX): It contains explicit locking at a lower level with exclusive or shared locks.

Shared & Intention-Exclusive (SIX): In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.

Compatibility Matrix with Intention Lock Modes: The below table describes the compatibility matrix for these lock modes:

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

It uses the intention lock modes to ensure serializability. It requires that if a transaction attempts to lock a node, then that node must follow these protocols:

- Transaction T1 should follow the lock-compatibility matrix.
- Transaction T1 firstly locks the root of the tree. It can lock it in any mode.
- If T1 currently has the parent of the node locked in either IX or IS mode, then the transaction T1 will lock a node in S or IS mode only.
- If T1 currently has the parent of the node locked in either IX or SIX modes, then the transaction T1 will lock a node in X, SIX, or IX mode only.
- If T1 has not previously unlocked any node only, then the Transaction T1 can lock a node.
- If T1 currently has none of the children of the node-locked only, then Transaction T1 will unlock a node.

Observe that in multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

- If transaction T1 reads record R_{a9} in file F_a, then transaction T1 needs to lock the database, area A_1 and file F_a in IX mode. Finally, it needs to lock R_{a2} in S mode.
- If transaction T2 modifies record R_{a9} in file F_a, then it can do so after locking the database, area A_1 and file F_a in IX mode. Finally, it needs to lock the R_{a9} in X mode.
- If transaction T3 reads all the records in file F_a, then transaction T3 needs to lock the database, and area A in IS mode. At last, it needs to lock F_a in S mode.
- If transaction T4 reads the entire database, then T4 needs to lock the database in S mode.

Recovery with Concurrent Transaction

- Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering.
- To ease this situation, 'checkpoint' concept is used by most DBMS.

As we have discussed checkpoint in Transaction Processing Concept of this tutorial, so you can go through the concepts again to make things more clear.

File organization

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.

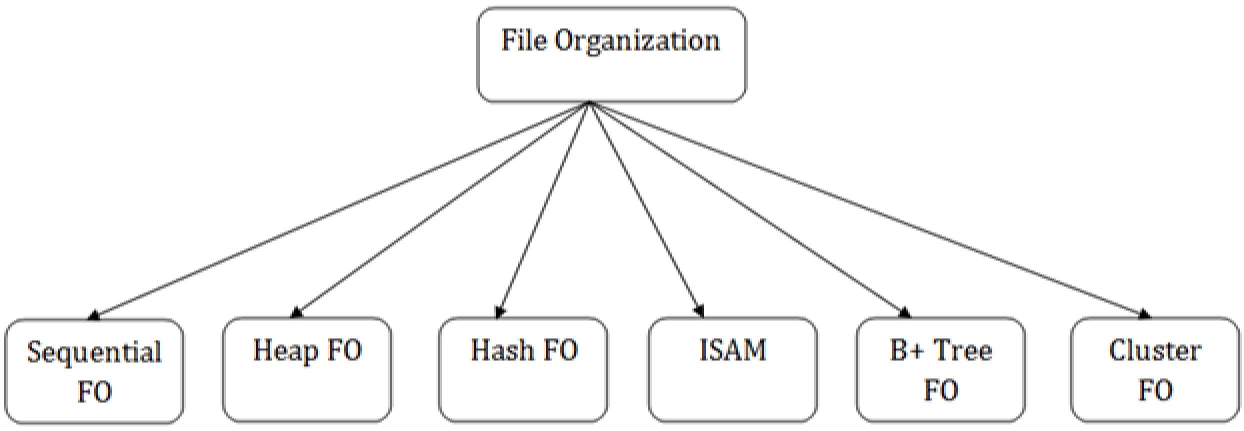
Objective of file organization

- 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:



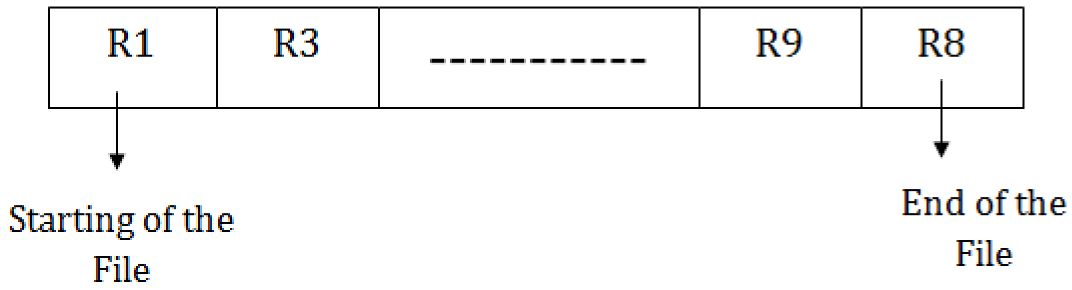
- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

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:

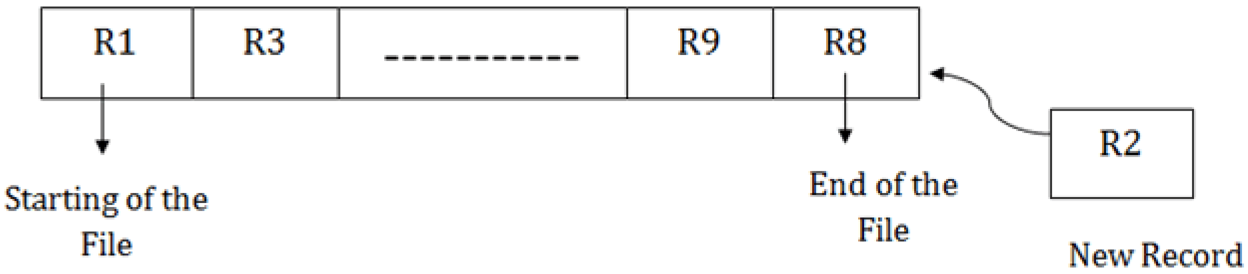
1. Pile File Method:

- 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.



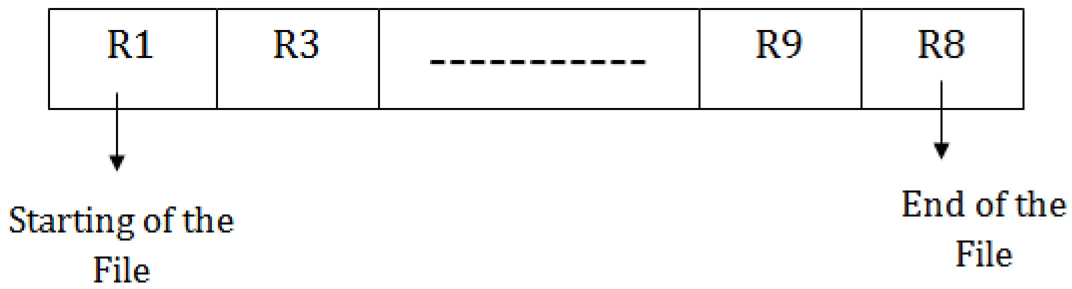
Insertion of the new record:

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.



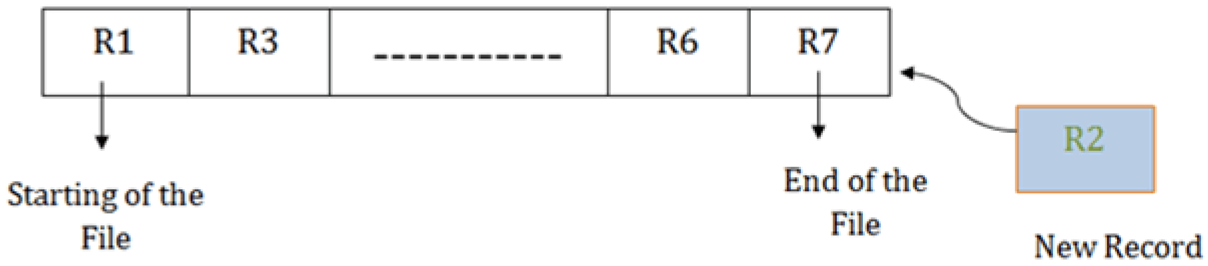
2. Sorted File Method:

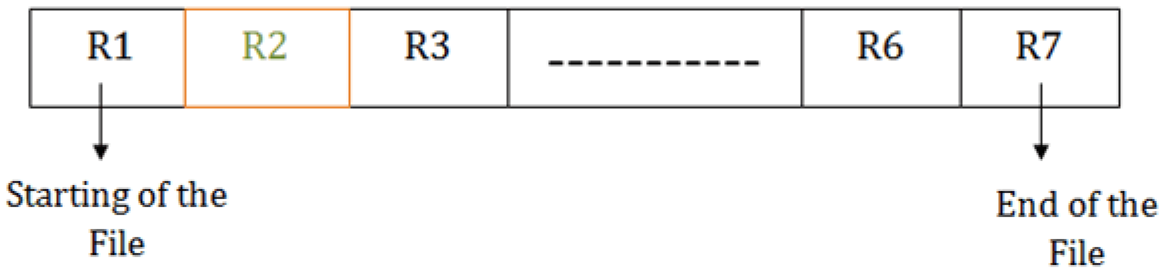
- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and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

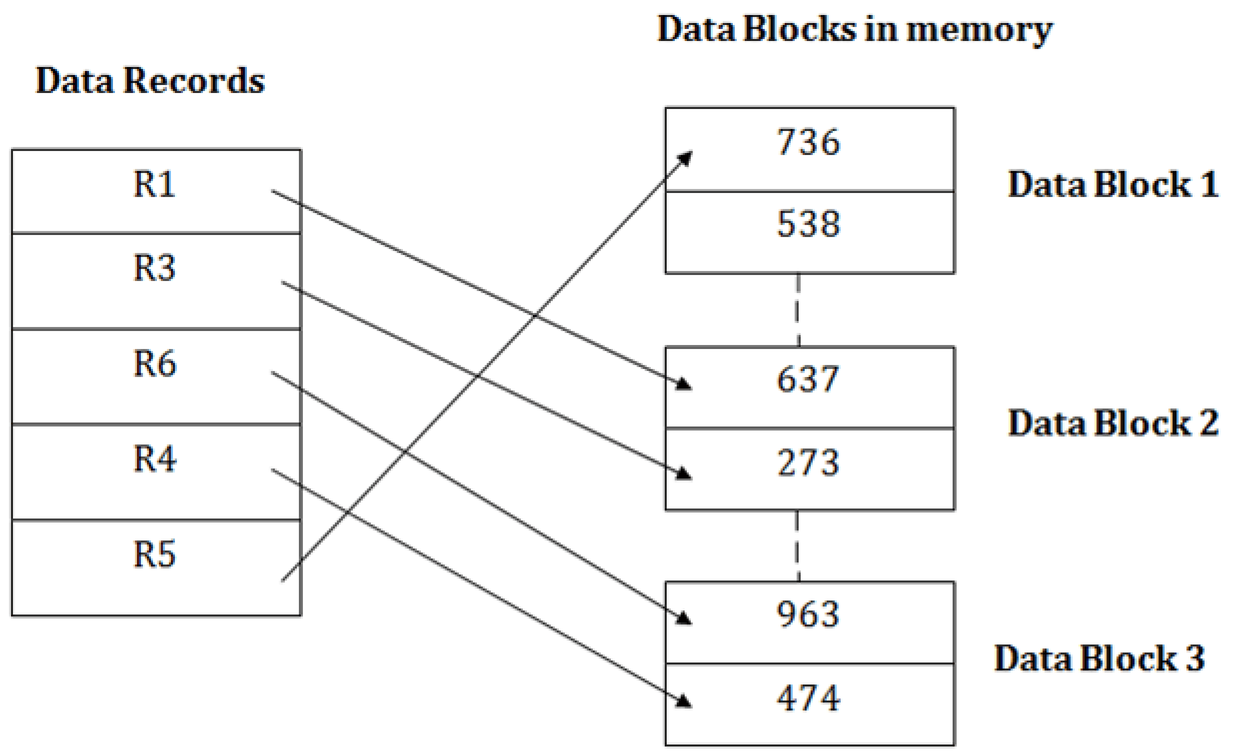
- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed 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.

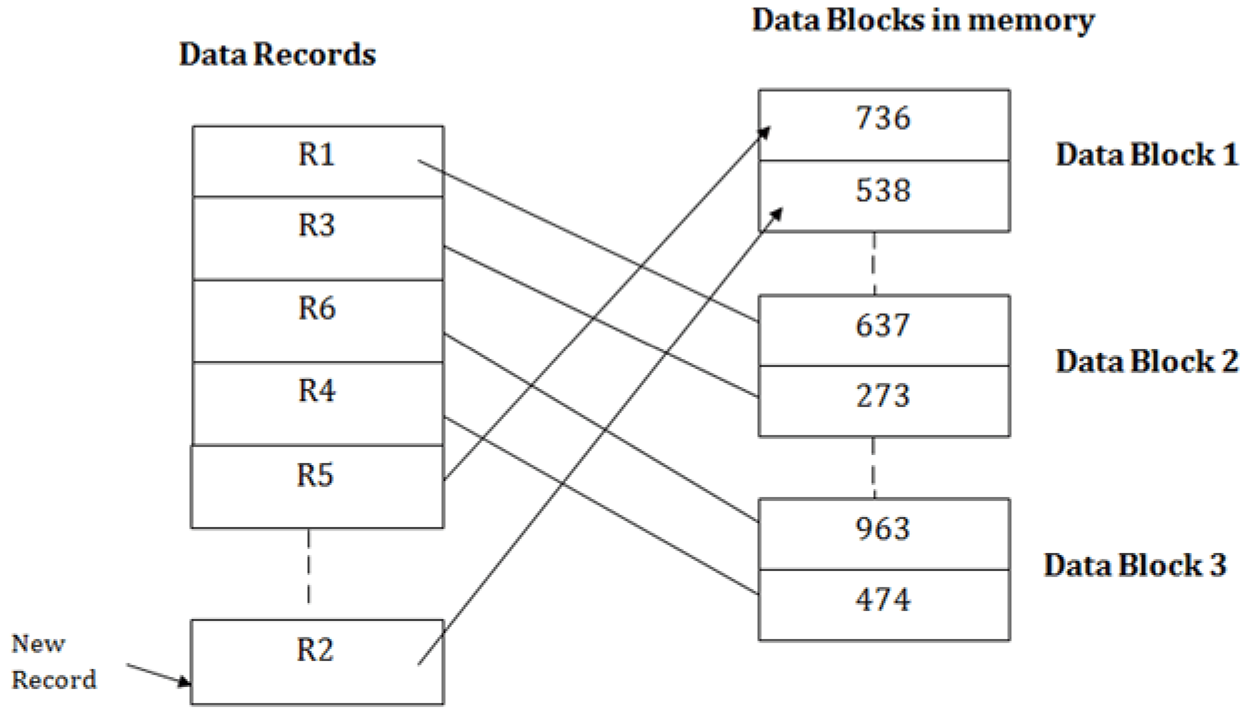
Heap file organization

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.

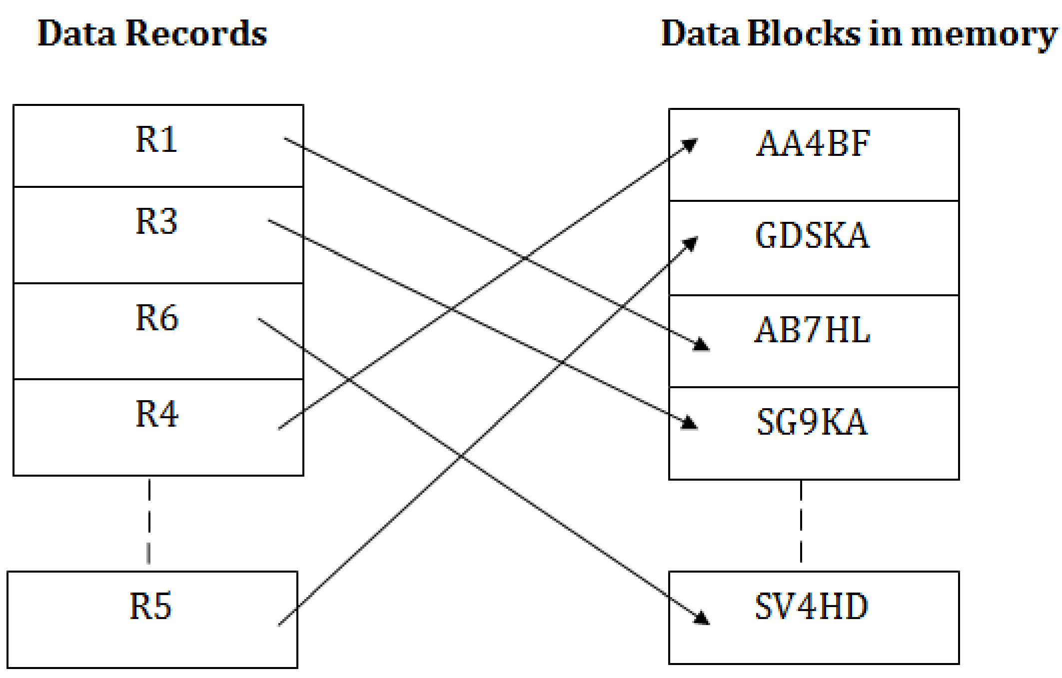
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

- This method is inefficient for the large database because it takes time to search or modify the record.
-
- This method is inefficient for large databases.

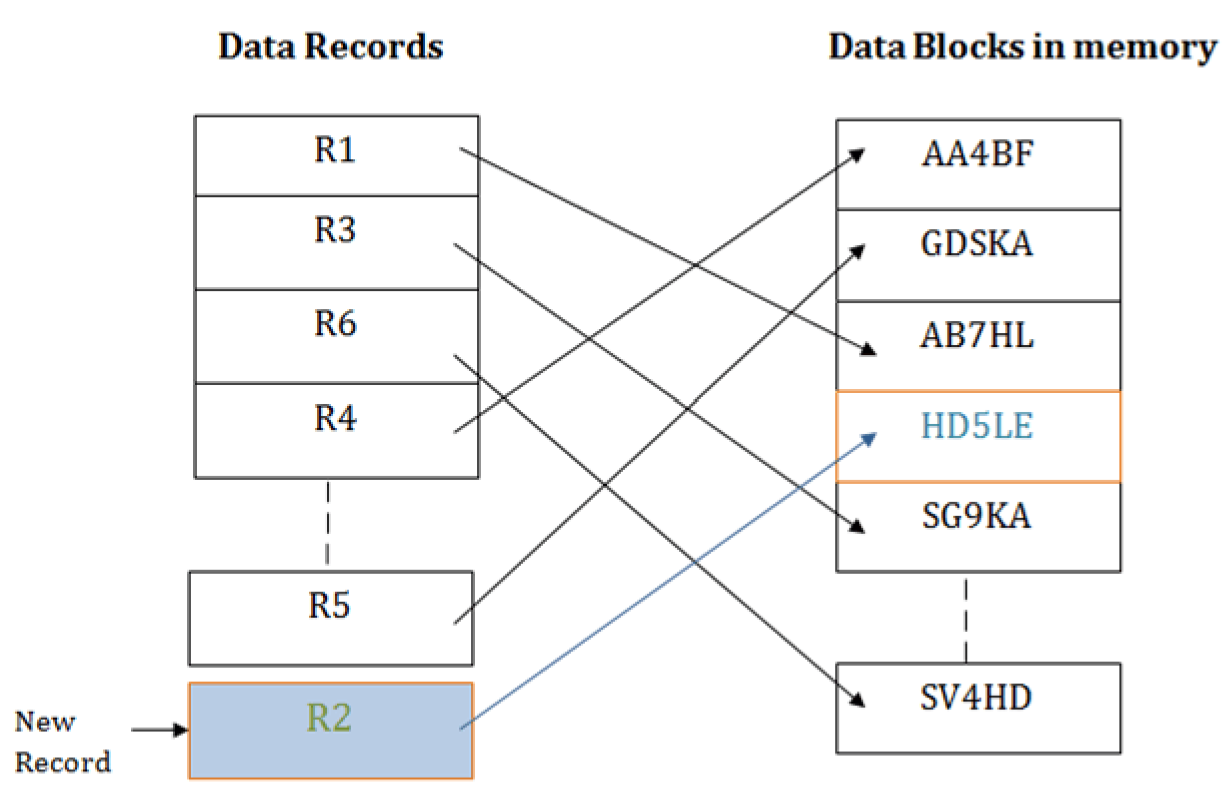
Hash File Organization

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.

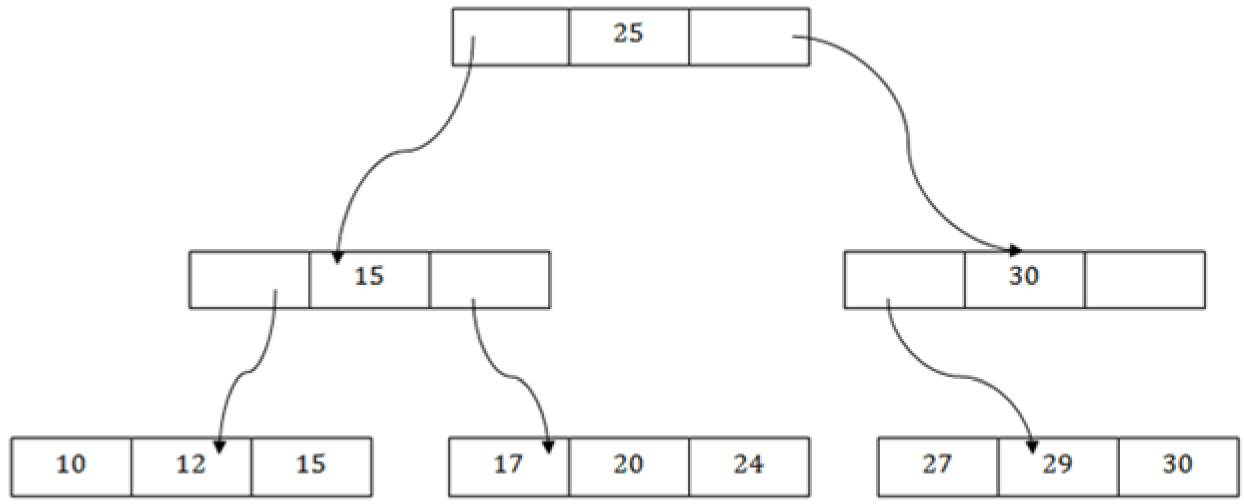
In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



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

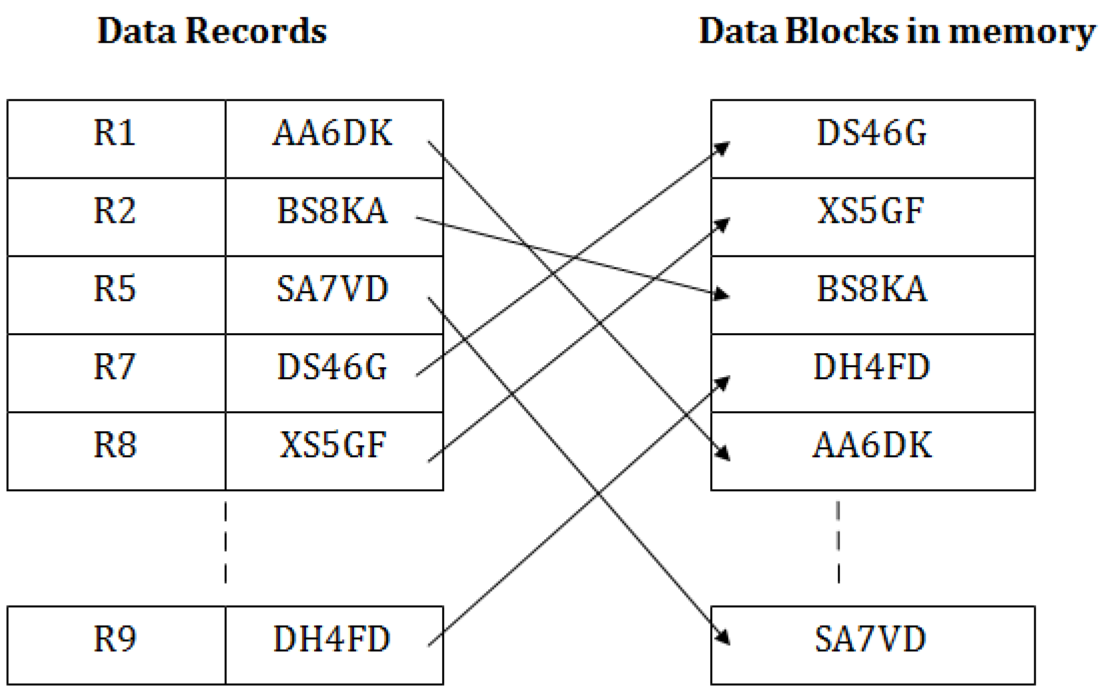
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and 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.
- This method is not suitable for a table with a 1:1 condition.

Indexing and B+ Tree

Indexing in DBMS

- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

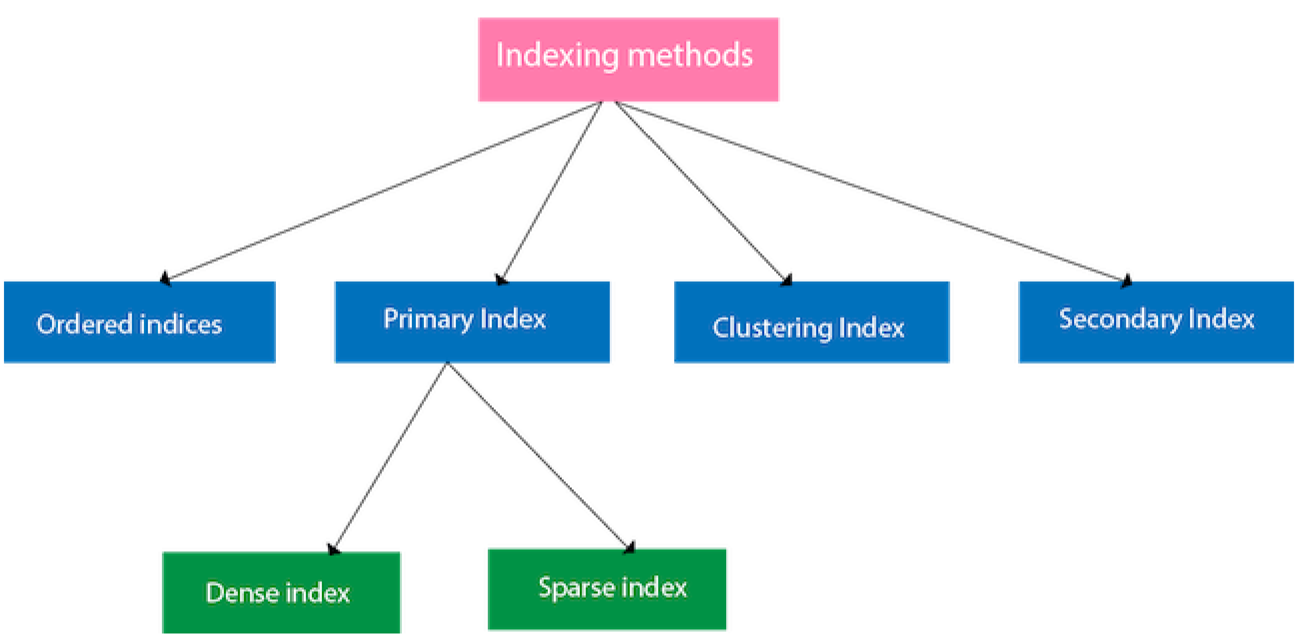
Indexes can be created using some database columns.

Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

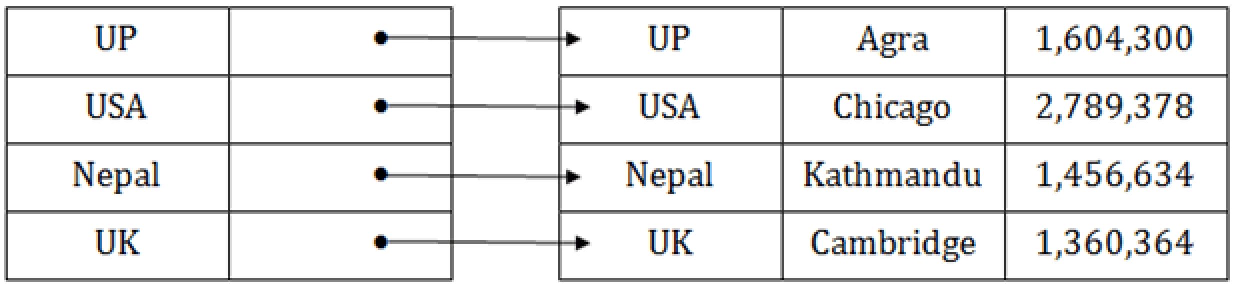
- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.

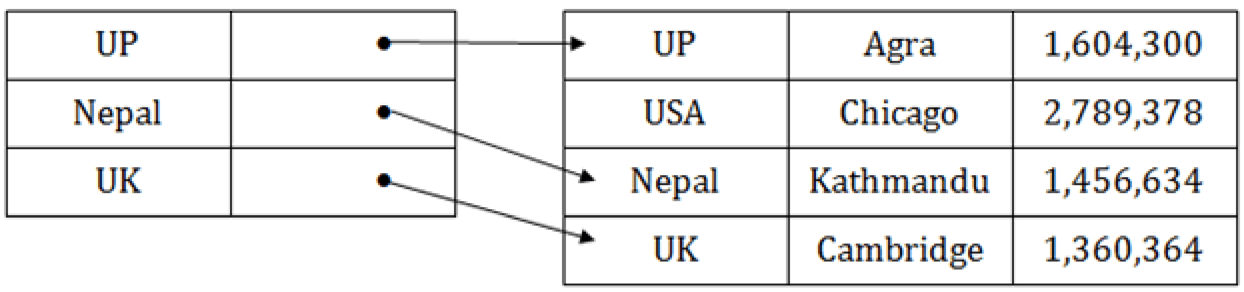
Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



Sparse index

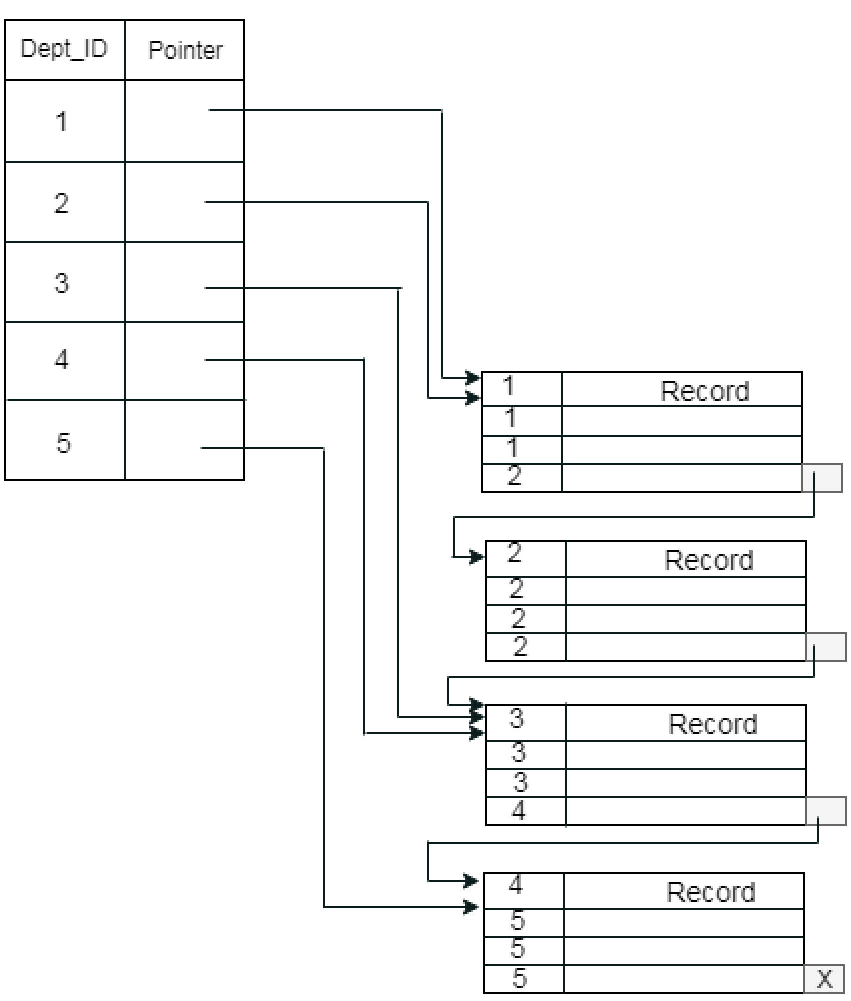
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



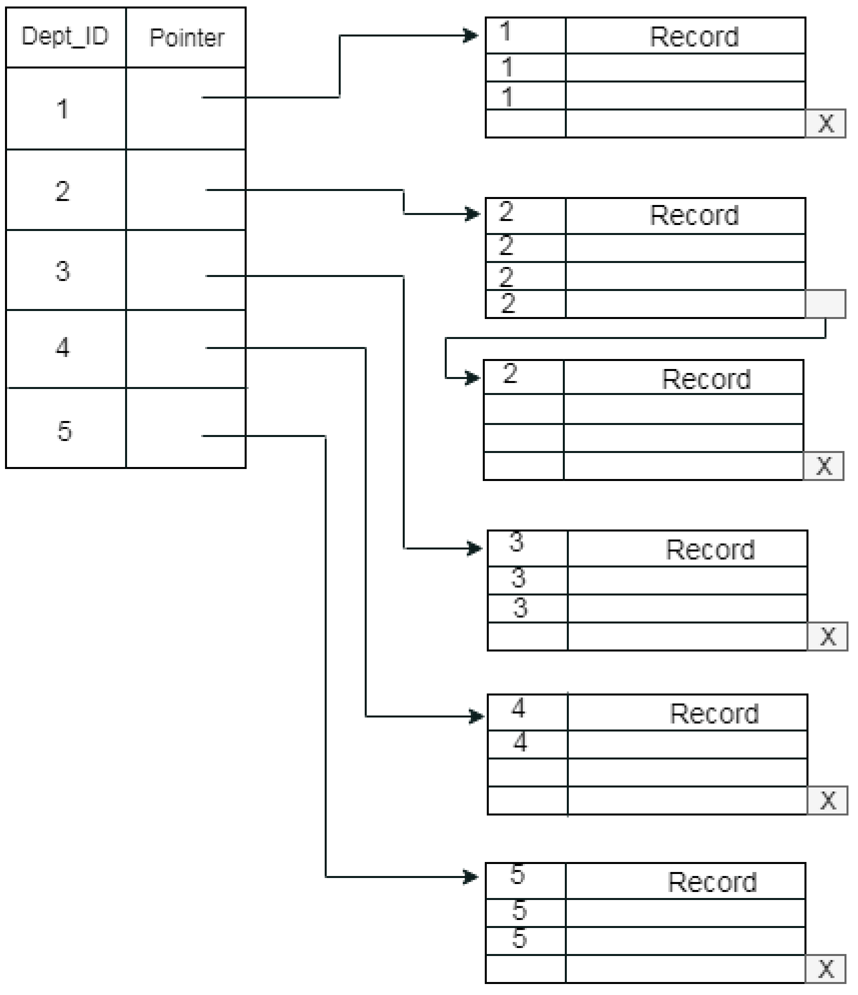
Clustering Index

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.



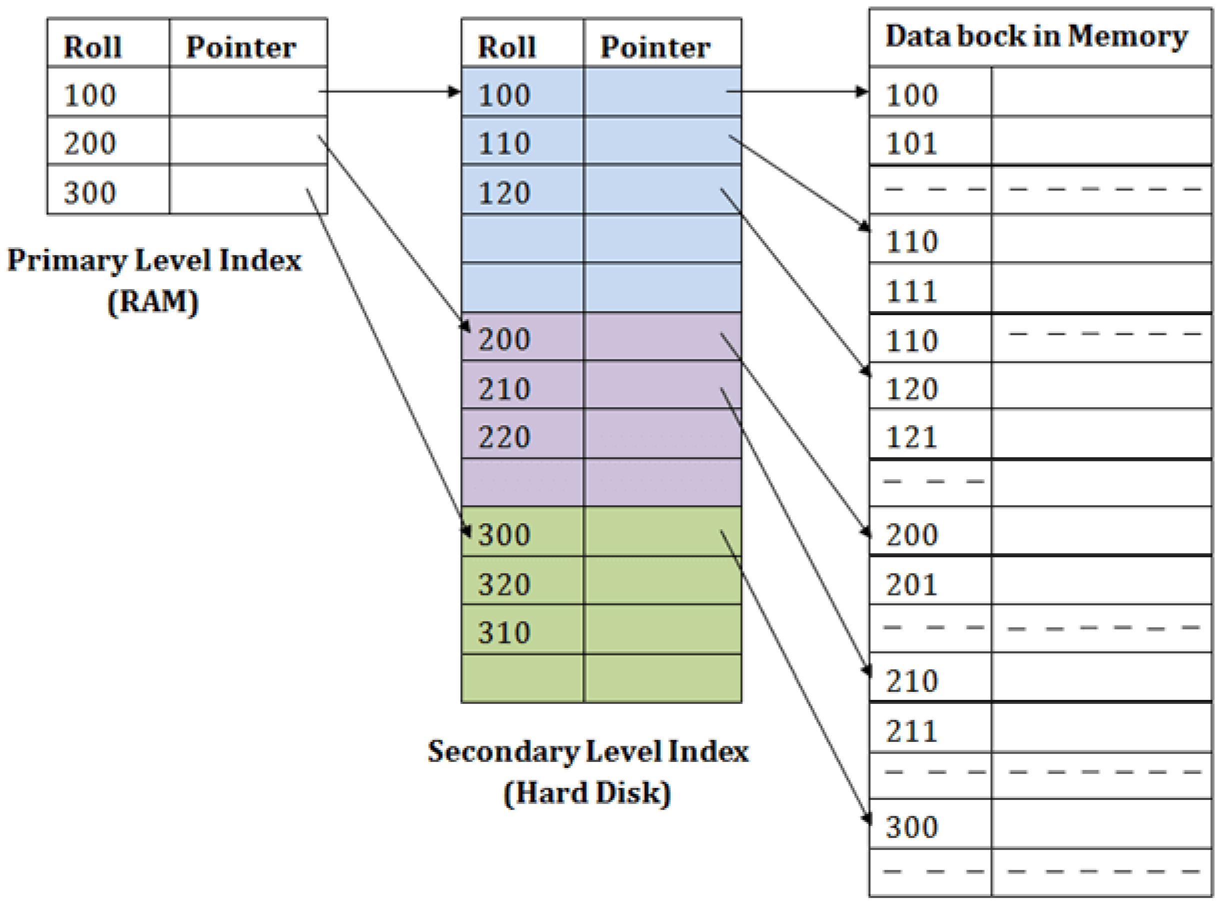
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index

In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



For example:

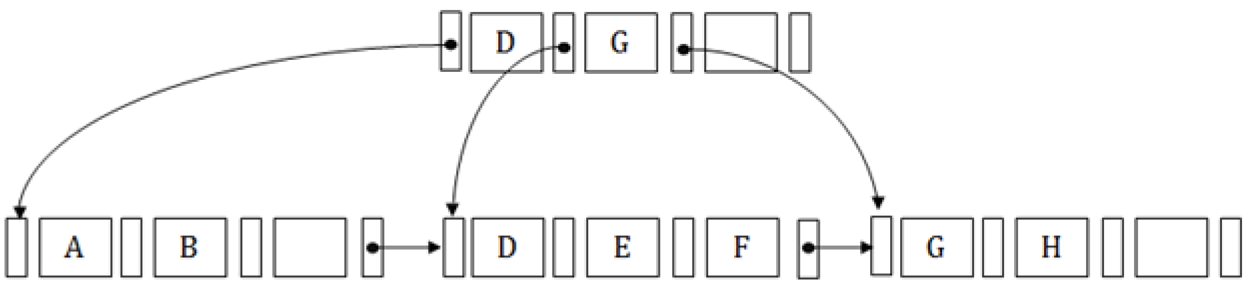
- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

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.

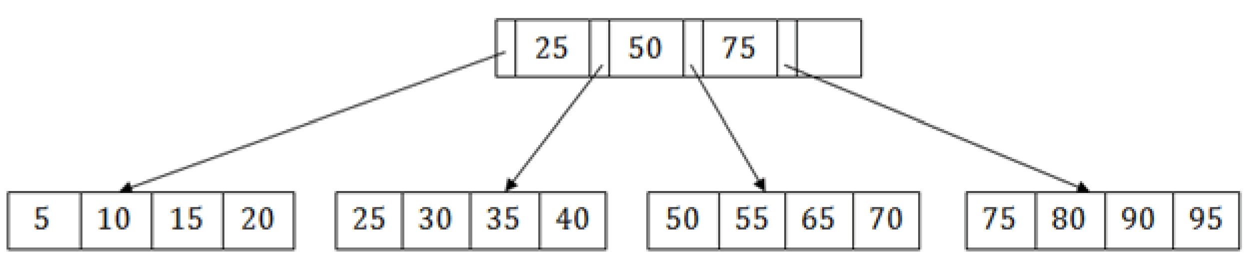
Leaf node

- 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.

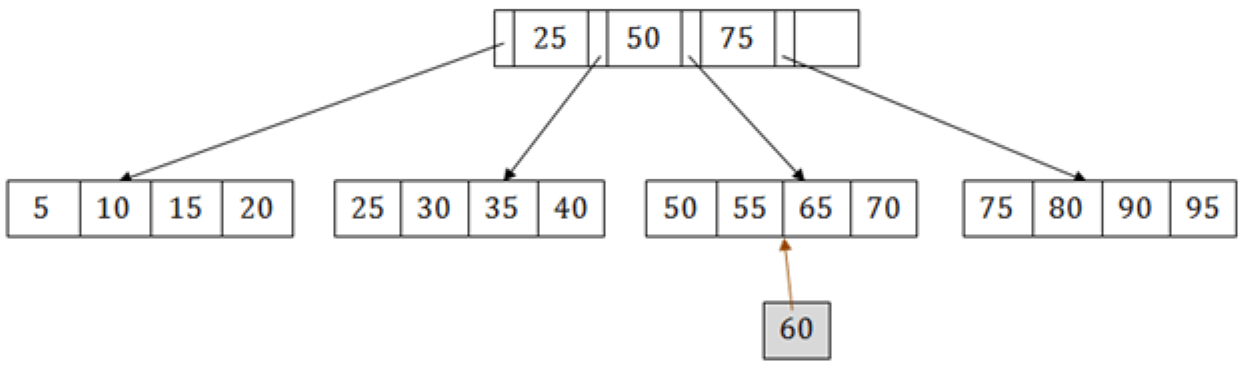
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



B+ Tree Insertion

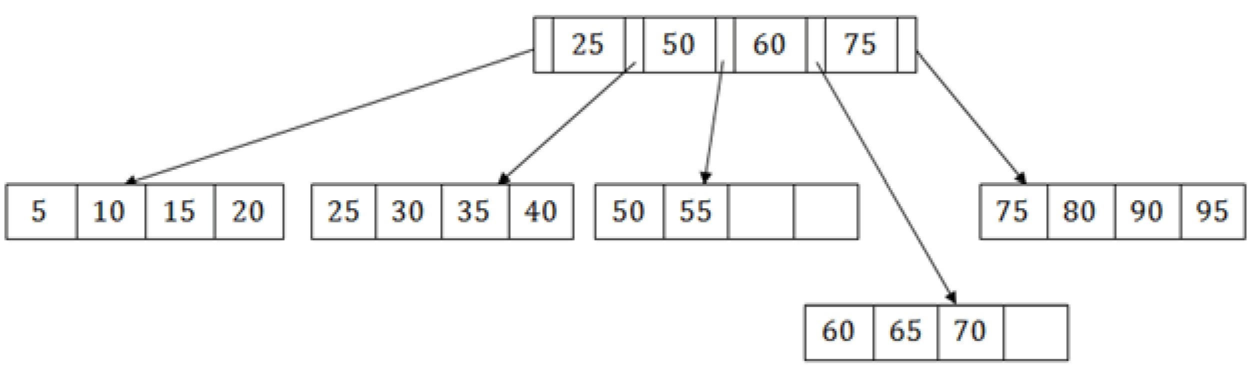
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

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.

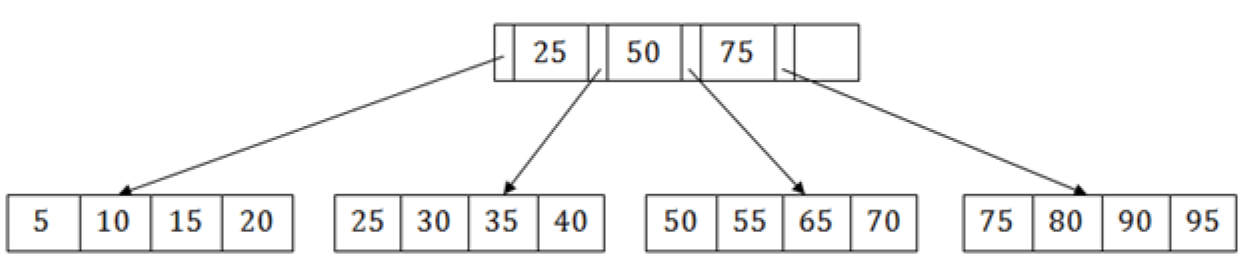


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

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:



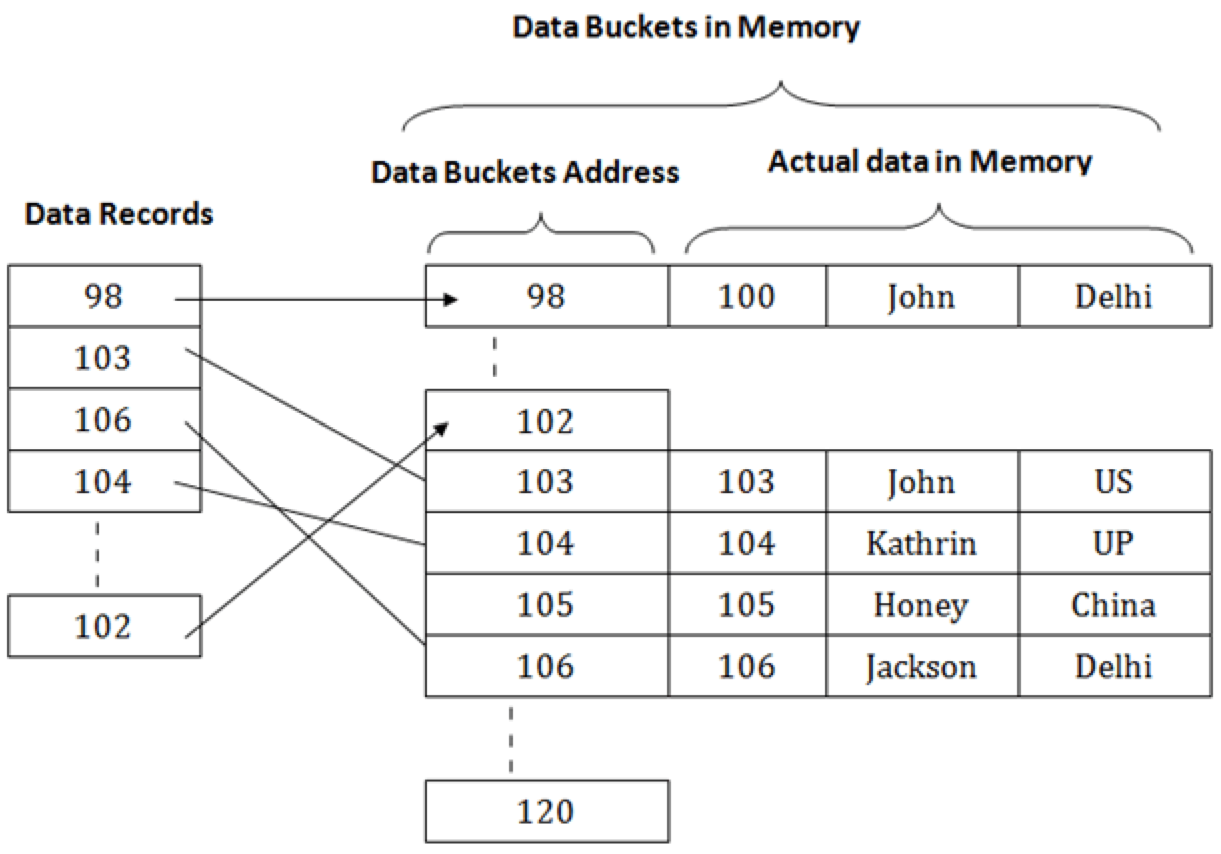
Hashing

Hashing

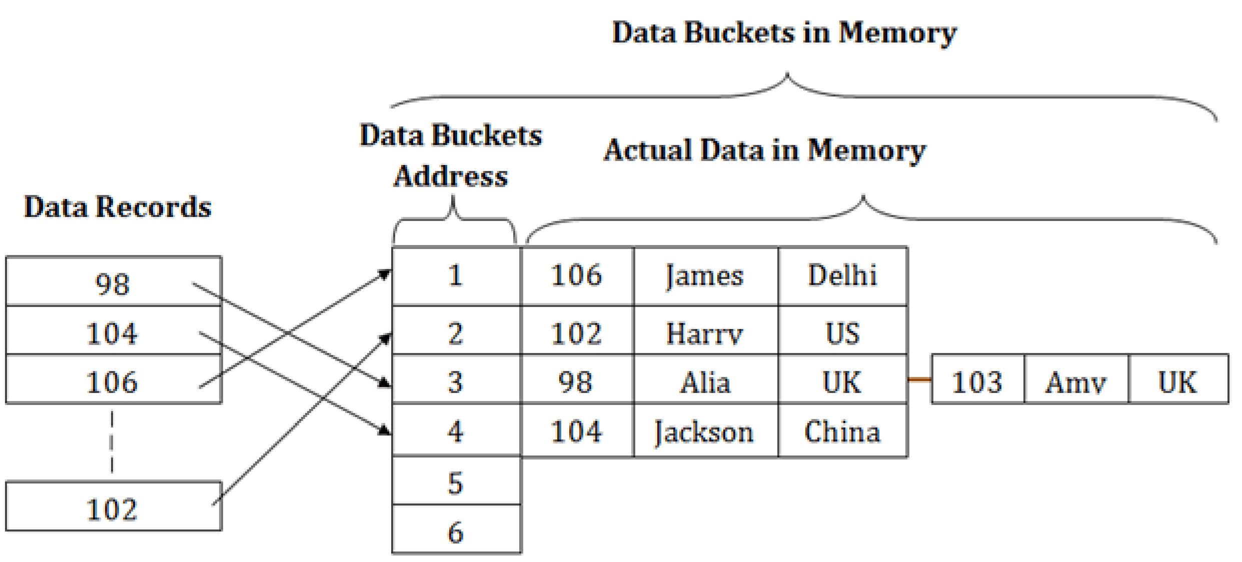
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

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.

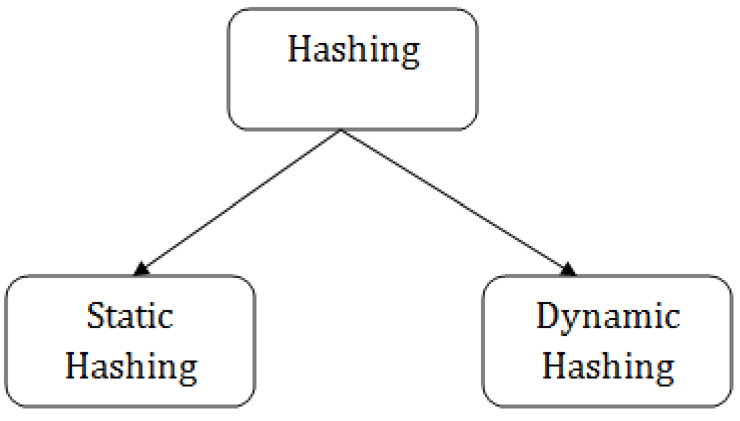
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary 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:

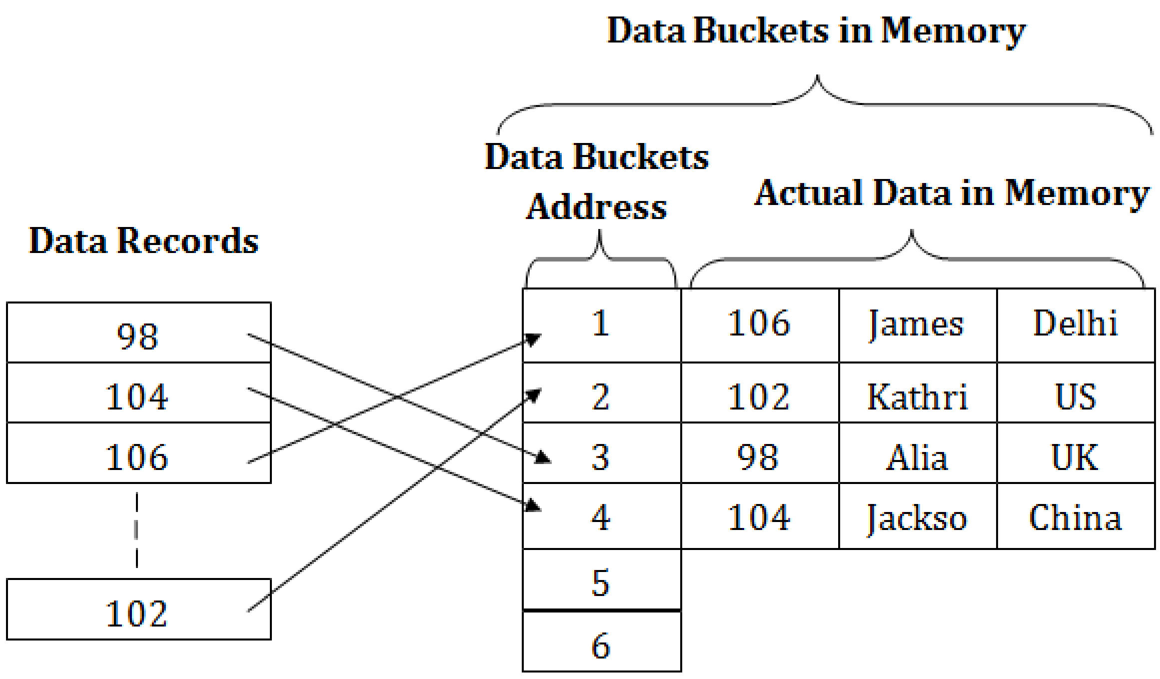


- Static Hashing
- Dynamic Hashing

Static Hashing

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.



Operations of Static Hashing

- **Searching a record**

When a record needs to be searched, then the same hash function retrieves the address of the bucket where the data is stored.

- **Insert a Record**

When a new record is inserted into the table, then we will generate an address for a new record based on the hash key and record is stored in that location.

- **Delete a Record**

To delete a record, we will first fetch the record which is supposed to be deleted. Then we will delete the records for that address in memory.

- **Update a Record**

To update a record, we will first search it using a hash function, and then the data record is updated.

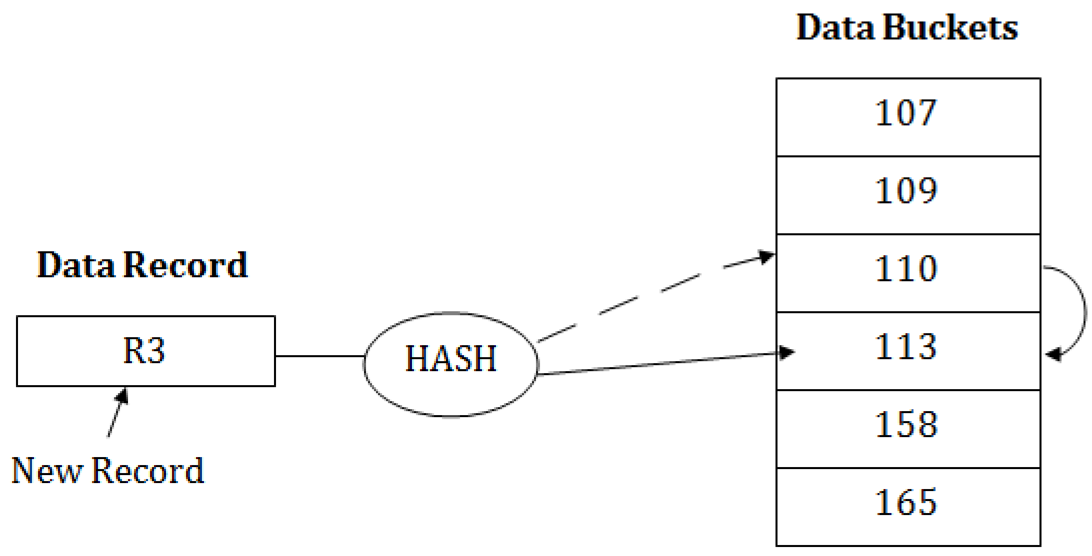
If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**. This is a critical situation in this method.

To overcome this situation, there are various methods. Some commonly used methods are as follows:

1. Open Hashing

When a hash function generates an address at which data is already stored, then the next bucket will be allocated to it. This mechanism is called as **Linear Probing**.

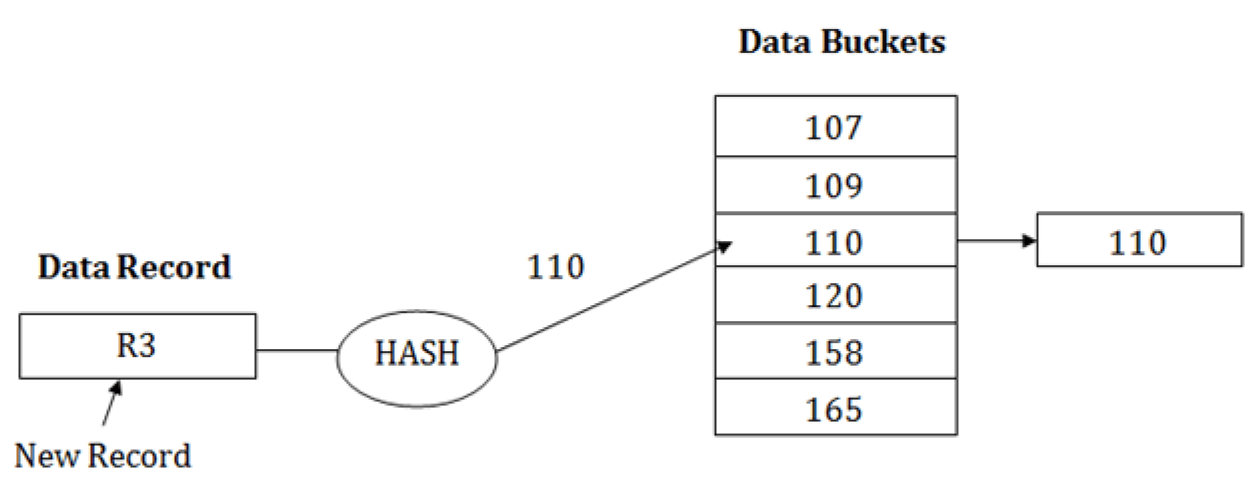
For example: suppose R3 is a new address which needs to be inserted, the hash function generates address as 112 for R3. But the generated address is already full. So the system searches next available data bucket, 113 and assigns R3 to it.



2. Close Hashing

When buckets are full, then a new data bucket is allocated for the same hash result and is linked after the previous one. This mechanism is known as **Overflow chaining**.

For example: Suppose R3 is a new address which needs to be inserted into the table, the hash function generates address as 110 for it. But this bucket is full to store the new data. In this case, a new bucket is inserted at the end of 110 buckets and is linked to it.



Dynamic Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

How to search a key

- First, calculate the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

How to insert a new record

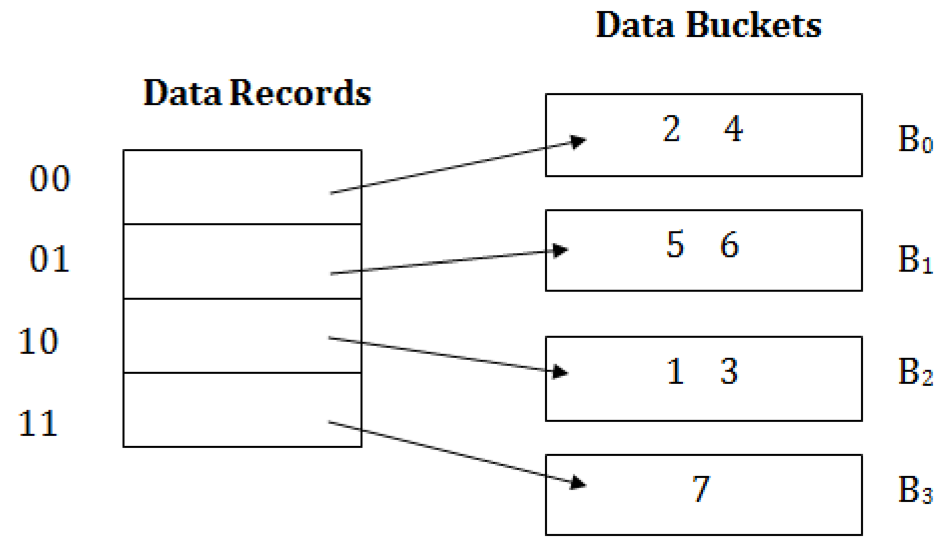
- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

For example:

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

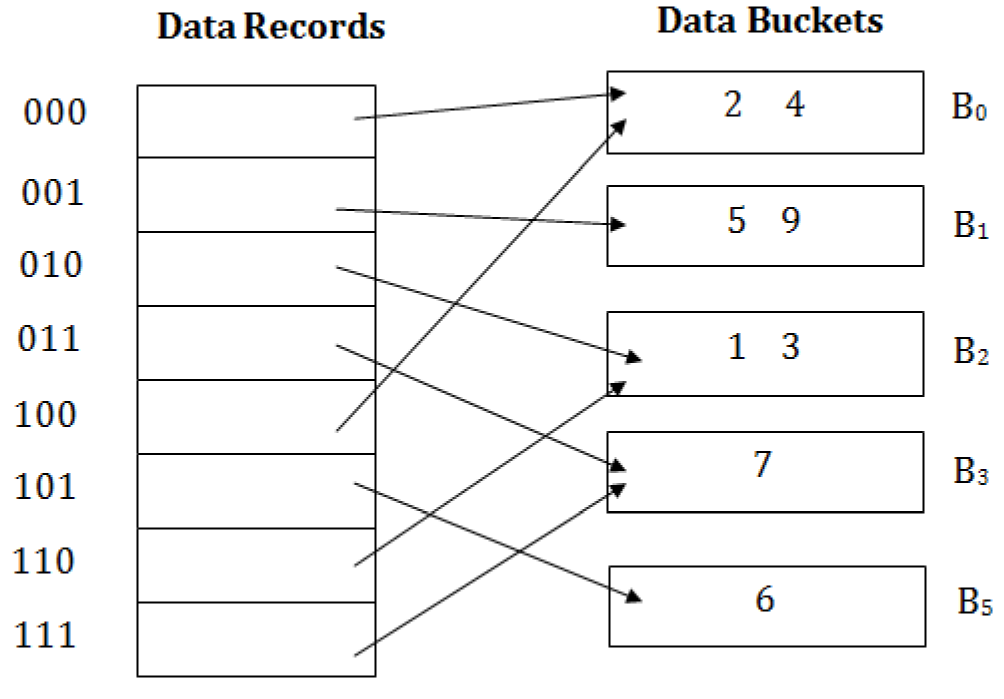
Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.



Insert key 9 with hash address 10001 into the above structure:

- Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
- The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
- Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
- Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
- Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.



Advantages of dynamic hashing

- In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
- In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.

- This method is good for the dynamic database where data grows and shrinks frequently.

Disadvantages of dynamic hashing

- In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
- In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

RAID

RAID

RAID refers to redundancy array of the independent disk. It is a technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both. It gives you the ability to survive one or more drive failure depending upon the RAID level used.

It consists of an array of disks in which multiple disks are connected to achieve different goals.

RAID technology

There are 7 levels of RAID schemes. These schemas are as RAID 0, RAID 1, ..., RAID 6.

These levels contain the following characteristics:

- It contains a set of physical disk drives.
- In this technology, the operating system views these separate disks as a single logical disk.
- In this technology, data is distributed across the physical drives of the array.
- Redundancy disk capacity is used to store parity information.
- In case of disk failure, the parity information can be helped to recover the data.

Standard RAID levels

RAID 0

- RAID level 0 provides data stripping, i.e., a data can place across multiple disks. It is based on stripping that means if one disk fails then all data in the array is lost.
- This level doesn't provide fault tolerance but increases the system performance.

Example:

Disk 0	Disk 1	Disk 2	Disk 3
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35

In this figure, block 0, 1, 2, 3 form a stripe.

In this level, instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one.

Disk 0	Disk 1	Disk 2	Disk 3
20	22	24	26
21	23	25	27
28	30	32	34
29	31	33	35

In this above figure, there is no duplication of data. Hence, a block once lost cannot be recovered.

Pros of RAID 0:

- In this level, throughput is increased because multiple data requests probably not on the same disk.
- This level full utilizes the disk space and provides high performance.
- It requires minimum 2 drives.

Cons of RAID 0:

- It doesn't contain any error detection mechanism.
- The RAID 0 is not a true RAID because it is not fault-tolerance.
- In this level, failure of either disk results in complete data loss in respective array.

RAID 1

This level is called mirroring of data as it copies the data from drive 1 to drive 2. It provides 100% redundancy in case of a failure.

Example:

Disk 0	Disk 1	Disk 2	Disk 3
A	A	B	B
C	C	D	D
E	E	F	F
G	G	H	H

Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.

Pros of RAID 1:

- The main advantage of RAID 1 is fault tolerance. In this level, if one disk fails, then the other automatically takes over.
- In this level, the array will function even if any one of the drives fails.

Cons of RAID 1:

- In this level, one extra drive is required per drive for mirroring, so the expense is higher.

RAID 2

- RAID 2 consists of bit-level striping using hamming code parity. In this level, each data bit in a word is recorded on a separate disk and ECC code of data words is stored on different set disks.
- Due to its high cost and complex structure, this level is not commercially used. This same performance can be achieved by RAID 3 at a lower cost.

Pros of RAID 2:

- This level uses one designated drive to store parity.
- It uses the hamming code for error detection.

Cons of RAID 2:

- It requires an additional drive for error detection.

RAID 3

- RAID 3 consists of byte-level striping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
- In case of drive failure, the parity drive is accessed, and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive.
- In this level, data can be transferred in bulk. Thus high-speed data transmission is possible.

Disk 0	Disk 1	Disk 2	Disk 3
A	B	C	P(A, B, C)
D	E	F	P(D, E, F)
G	H	I	P(G, H, I)
J	K	L	P(J, K, L)

Pros of RAID 3:

- In this level, data is regenerated using parity drive.
- It contains high data transfer rates.
- In this level, data is accessed in parallel.

Cons of RAID 3:

- It required an additional drive for parity.
- It gives a slow performance for operating on small sized files.

RAID 4

- RAID 4 consists of block-level stripping with a parity disk. Instead of duplicating data, the RAID 4 adopts a parity-based approach.
- This level allows recovery of at most 1 disk failure due to the way parity works. In this level, if more than one disk fails, then there is no way to recover the data.
- Level 3 and level 4 both are required at least three disks to implement RAID.

Disk 0	Disk 1	Disk 2	Disk 3
A	B	C	P0
D	E	F	P1
G	H	I	P2
J	K	L	P3

In this figure, we can observe one disk dedicated to parity.

In this level, parity can be calculated using an XOR function. If the data bits are 0,0,0,1 then the parity bits is XOR(0,1,0,0) = 1. If the parity bits are 0,0,1,1 then the parity bit is XOR(0,0,1,1)= 0. That means, even number of one results in parity 0 and an odd number of one results in parity 1.

C1	C2	C3	C4	Parity
0	1	0	0	1
0	0	1	1	0

Suppose that in the above figure, C2 is lost due to some disk failure. Then using the values of all the other columns and the parity bit, we can recompute the data bit stored in C2. This level allows us to recover lost data.

RAID 5

- RAID 5 is a slight modification of the RAID 4 system. The only difference is that in RAID 5, the parity rotates among the drives.
- It consists of block-level striping with DISTRIBUTED parity.
- Same as RAID 4, this level allows recovery of at most 1 disk failure. If more than one disk fails, then there is no way for data recovery.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

This figure shows that how parity bit rotates.

This level was introduced to make the random write performance better.

Pros of RAID 5:

- This level is cost effective and provides high performance.
- In this level, parity is distributed across the disks in an array.
- It is used to make the random write performance better.

Cons of RAID 5:

- In this level, disk failure recovery takes longer time as parity has to be calculated from all available drives.
- This level cannot survive in concurrent drive failure.

RAID 6

- This level is an extension of RAID 5. It contains block-level stripping with 2 parity bits.
- In RAID 6, you can survive 2 concurrent disk failures. Suppose you are using RAID 5, and RAID 1. When your disks fail, you need to replace the failed disk because if simultaneously another disk fails then you won't be able to recover any of the data, so in this case RAID 6 plays its part where you can survive two concurrent disk failures before you run out of options.

Disk 1	Disk 2	Disk 3	Disk 4
A0	B0	Q0	P0
A1	Q1	P1	D1
Q2	P2	C2	D2

P3	B3	C3	Q3

Pros of RAID 6:

- This level performs RAID 0 to strip data and RAID 1 to mirror. In this level, stripping is performed before mirroring.
- In this level, drives required should be multiple of 2.

Cons of RAID 6:

- It is not utilized 100% disk capability as half is used for mirroring.
- It contains very limited scalability.

SQL Introduction

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

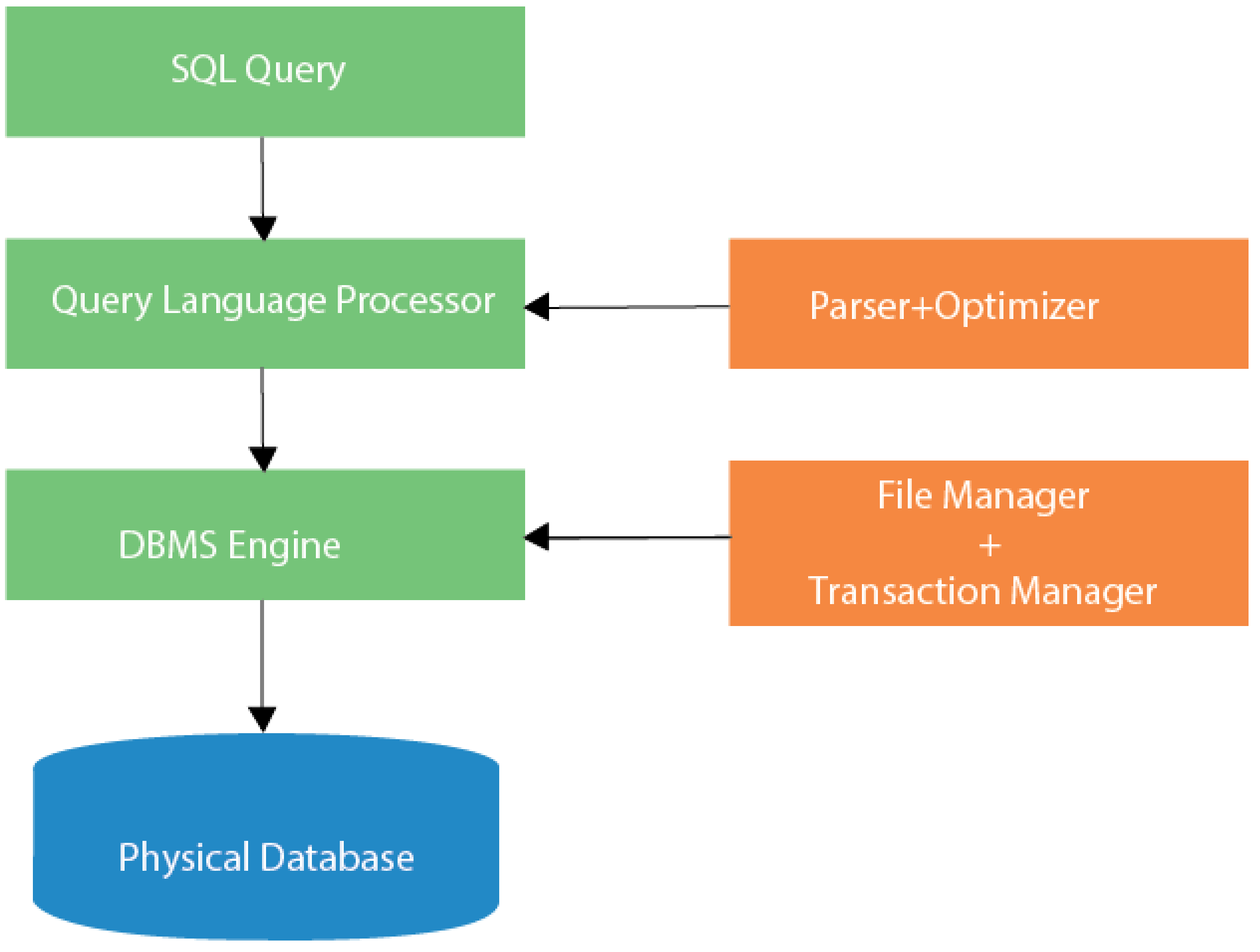
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

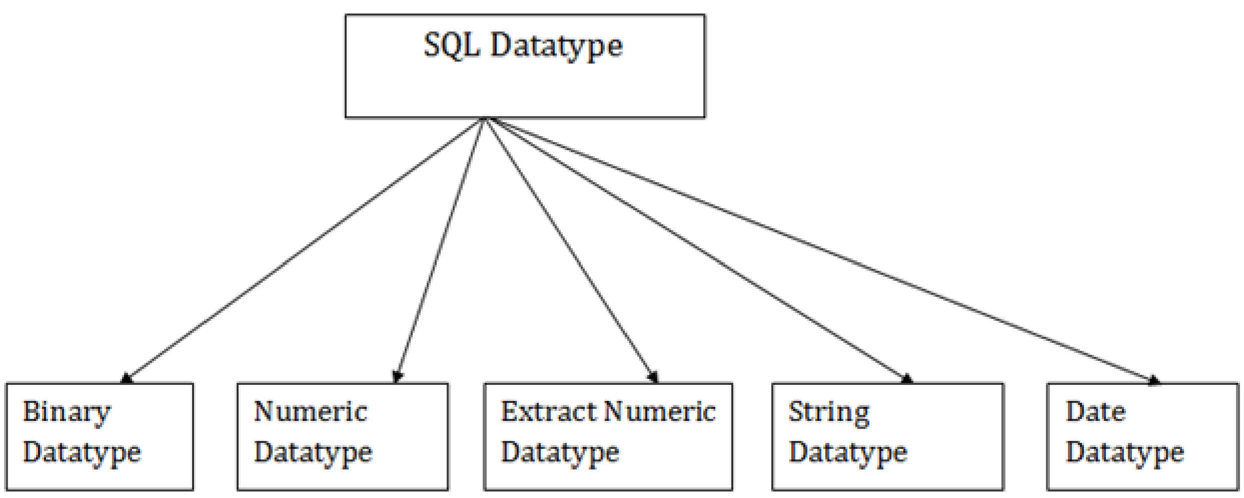
Multiple data view

Using the SQL language, the users can make different views of the database structure.

SQL Datatype

- SQL Datatype is used to define the values that a column can contain.
- Every column is required to have a name and data type in the database table.

Datatype of SQL:



1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

Data Type	Description
binary	It has a maximum length of 8000 bytes. It contains fixed-length binary data.
varbinary	It has a maximum length of 8000 bytes. It contains variable-length binary data.
image	It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data.

2. Approximate Numeric Datatype :

The subtypes are given below:

Data type	From	To	Description
float	-1.79E + 308	1.79E + 308	It is used to specify a floating-point value e.g. 6.2, 2.9 etc.
real	-3.40e + 38	3.40E + 38	It specifies a single precision floating point number

3. Exact Numeric Datatype

The subtypes are given below:

Data type	Description
int	It is used to specify an integer value.
smallint	It is used to specify small integer value.
bit	It has the number of bits to store.
decimal	It specifies a numeric value that can have a decimal number.
numeric	It is used to specify a numeric value.

4. Character String Datatype

The subtypes are given below:

Data type	Description
char	It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters.
varchar	It has a maximum length of 8000 characters. It contains variable-length non-unicode characters.
text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters.

5. Date and time Datatypes

The subtypes are given below:

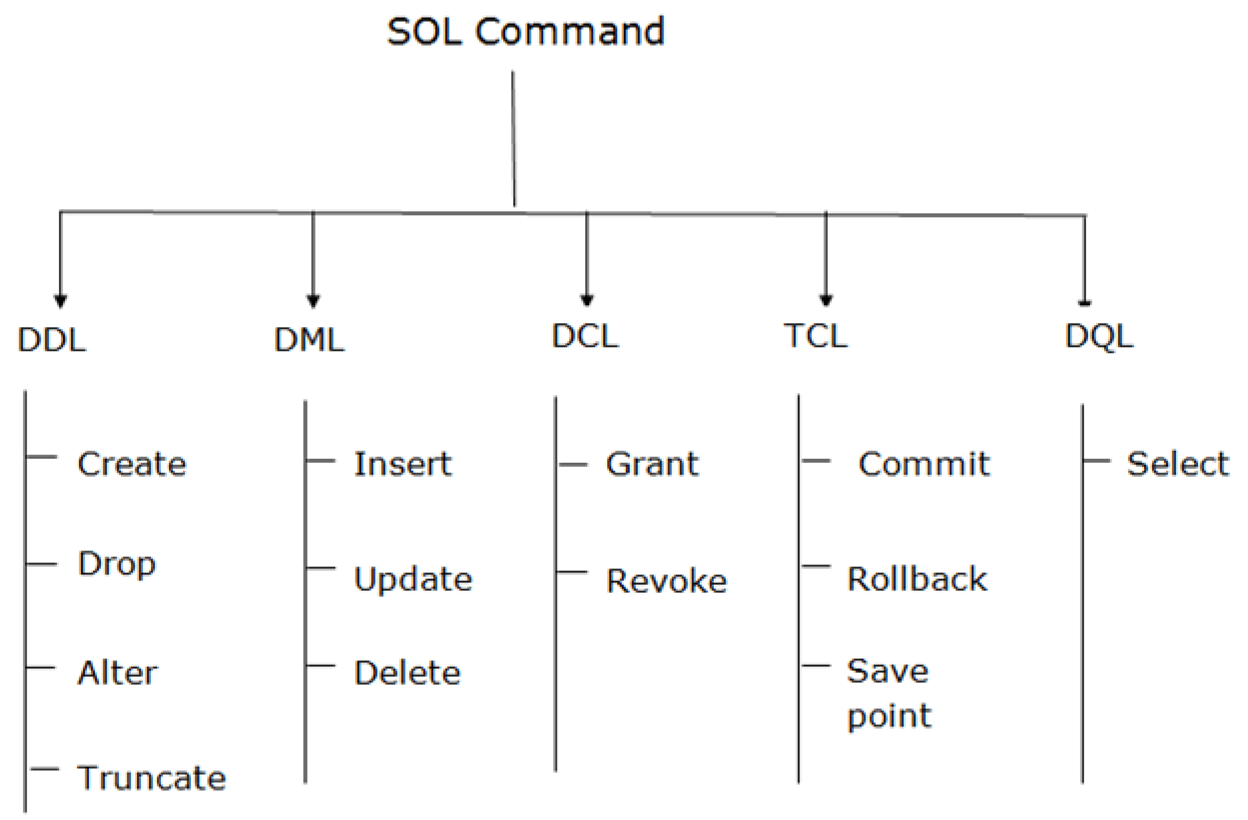
Datatype	Description
date	It is used to store the year, month, and days value.
time	It is used to store the hour, minute, and second values.
timestamp	It stores the year, month, day, hour, minute, and the second value

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

1. DROP TABLE table_name;

Example

1. DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

1. ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE table_name MODIFY(column_definitions....);

EXAMPLE

```
1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

```
1. TRUNCATE TABLE table_name;
```

Example:

```
1. TRUNCATE TABLE EMPLOYEE;
```

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);
```

Or

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

For example:

```
1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
1. UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

For example:

```
1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax:

```
1. DELETE FROM table_name [WHERE condition];
```

For example:

```
1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

1. COMMIT;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

1. ROLLBACK;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

1. SAVEPOINT SAVEPOINT_NAME;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

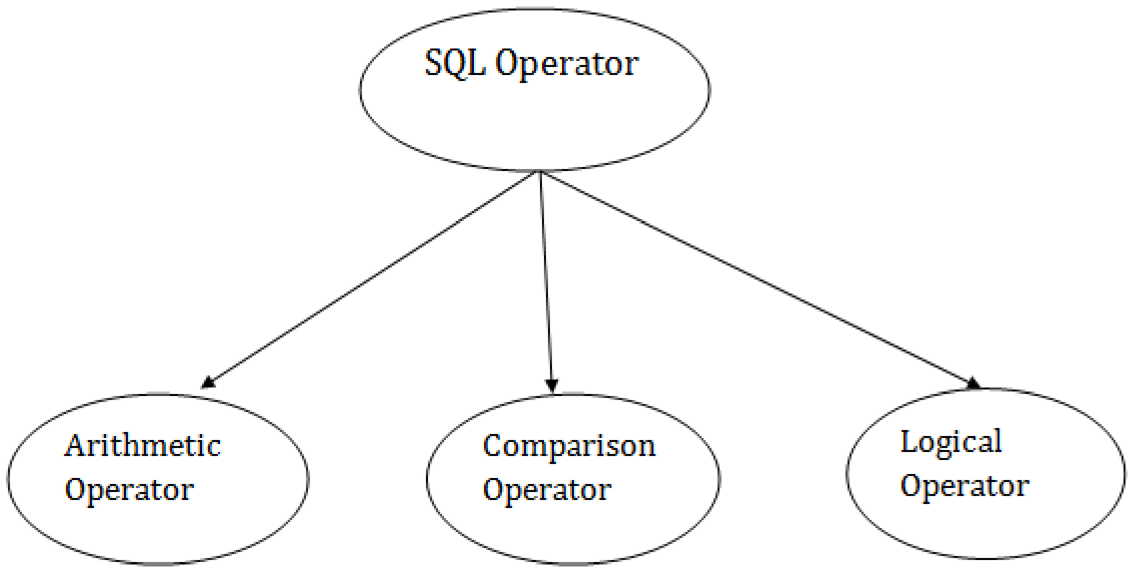
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

SQL Operator

There are various types of SQL operator:



SQL Arithmetic Operators

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
+	It adds the value of both operands.	a+b will give 30
-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200
/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description	Example
=	It checks if two operands values are equal or not, if the values are queal then condition becomes true.	(a=b) is not true
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.	(a!=b) is true
<>	It checks if two operands values are equal or not, if values are not equal then condition becomes true.	(a<>b) is true
>	It checks if the left operand value is greater than right operand value, if yes then condition becomes true.	(a>b) is not true
<	It checks if the left operand value is less than right operand value, if yes then condition becomes true.	(a<b) is true
>=	It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true.	(a>=b) is not true
<=	It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true.	(a<=b) is true
!<	It checks if the left operand value is not less than the right operand value, if yes then condition becomes true.	(a!=b) is not true
!>	It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true.	(a!>b) is true

SQL Logical Operators

There is the list of logical operator used in SQL:

Operator	Description
ALL	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in an SQL statement.
ANY	It compares the values in the list according to the condition.
BETWEEN	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXISTS	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

SQL Table

- SQL Table is a collection of data which is organized in terms of rows and columns. In DBMS, the table is known as relation and row as a tuple.
- Table is a simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of the **EMPLOYEE** table:

EMP_ID	EMP_NAME	CITY	PHONE_NO
1	Kristen	Washington	7289201223
2	Anna	Franklin	9378282882
3	Jackson	Bristol	9264783838
4	Kellan	California	7254728346
5	Ashley	Hawaii	9638482678

In the above table, "EMPLOYEE" is the table name, "EMP_ID", "EMP_NAME", "CITY", "PHONE_NO" are the column names. The combination of data of multiple columns forms a row, e.g., 1, "Kristen", "Washington" and 7289201223 are the data of one row.

Operation on Table

1. Create table
2. Drop table
3. Delete table
4. Rename table

SQL Create Table

SQL create table is used to create a table in the database. To define the table, you should define the name of the table and also define its columns and column's data type.

Syntax

```
create table "table_name"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

Example

```
SQL> CREATE TABLE EMPLOYEE (
EMP_ID INT                    NOT NULL,
EMP_NAME VARCHAR (25) NOT NULL,
PHONE_NO INT                  NOT NULL,
ADDRESS CHAR (30),
PRIMARY KEY (ID)
);
```

If you create the table successfully, you can verify the table by looking at the message by the SQL server. Else you can use DESC command as follows:

SQL> DESC EMPLOYEE;

Field	Type	Null	Key	Default	Extra
EMP_ID	int(11)	NO	PRI	NULL	
EMP_NAME	varchar(25)	NO		NULL	
PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

- 4 rows in set (0.35 sec)

Now you have an EMPLOYEE table in the database, and you can use the stored information related to the employees.

Drop table

A SQL drop table is used to delete a table definition and all the data from a table. When this command is executed, all the information available in the table is lost forever, so you have to very careful while using this command.

Syntax

1. DROP TABLE "table_name";

Firstly, you need to verify the **EMPLOYEE** table using the following command:

1. SQL> DESC EMPLOYEE;

Field	Type	Null	Key	Default	Extra
EMP_ID	int(11)	NO	PRI	NULL	
EMP_NAME	varchar(25)	NO		NULL	
PHONE_NO	NO	int(11)		NULL	
ADDRESS	YES			NULL	char(30)

- 4 rows in set (0.35 sec)

This table shows that EMPLOYEE table is available in the database, so we can drop it as follows:

1. SQL>DROP TABLE EMPLOYEE;

Now, we can check whether the table exists or not using the following command:

1. Query OK, 0 rows affected (0.01 sec)

As this shows that the table is dropped, so it doesn't display it.

SQL DELETE table

In SQL, DELETE statement is used to delete rows from a table. We can use WHERE condition to delete a specific row from a table. If you want to delete all the records from the table, then you don't need to use the WHERE clause.

Syntax

1. DELETE FROM table_name WHERE condition;

Example

Suppose, the EMPLOYEE table having the following records:

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Denzel	Boston	7353662627	100000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

The following query will DELETE an employee whose ID is 2.

1. SQL> DELETE FROM EMPLOYEE
2. WHERE EMP_ID = 3;

Now, the EMPLOYEE table would have the following records.

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
4	Angelina	Denver	9232673822	600000
5	Robert	Washington	9367238263	350000
6	Christian	Los angels	7253847382	260000

If you don't specify the WHERE condition, it will remove all the rows from the table.

1. DELETE FROM EMPLOYEE;

Now, the EMPLOYEE table would not have any records.

SQL SELECT Statement

In SQL, the SELECT statement is used to query or retrieve data from a table in the database. The returns data is stored in a table, and the result table is known as result-set.

Syntax

1. SELECT column1, column2, ...
2. FROM table_name;

Here, the expression is the field name of the table that you want to select data from.

Use the following syntax to select all the fields available in the table:

1. SELECT * FROM table_name;

Example:

EMPLOYEE

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

To fetch the EMP_ID of all the employees, use the following query:

1. SELECT EMP_ID FROM EMPLOYEE;

Output

EMP_ID
1
2
3
4
5

To fetch the EMP_NAME and SALARY, use the following query:

1. SELECT EMP_NAME, SALARY FROM EMPLOYEE;

EMP_NAME	SALARY
Kristen	150000
Russell	200000
Angelina	600000
Robert	350000
Christian	260000

To fetch all the fields from the EMPLOYEE table, use the following query:

1. SELECT * FROM EMPLOYEE

Output

EMP_ID	EMP_NAME	CITY	PHONE_NO	SALARY
1	Kristen	Chicago	9737287378	150000
2	Russell	Austin	9262738271	200000
3	Angelina	Denver	9232673822	600000
4	Robert	Washington	9367238263	350000
5	Christian	Los angels	7253847382	260000

SQL INSERT Statement

The SQL INSERT statement is used to insert a single or multiple data in a table. In SQL, You can insert the data in two ways:

1. Without specifying column name
2. By specifying column name

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36

1. Without specifying column name

If you want to specify all column values, you can specify or ignore the column values.

Syntax

1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value 3, ... Value N);

Query

1. INSERT INTO EMPLOYEE VALUES (6, 'Marry', 'Canada', 600000, 48);

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

2. By specifying column name

To insert partial column values, you must have to specify the column names.

Syntax

1. INSERT INTO TABLE_NAME
2. [(col1, col2, col3,.... col N)]
3. VALUES (value1, value2, value 3, ... Value N);

Query

1. INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, AGE) VALUES (7, 'Jack', 40);

Output: After executing this query, the table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48
7	Jack	null	null	40

Note: In SQL INSERT query, if you add values for all columns then there is no need to specify the column name. But, you must be sure that you are entering the values in the same order as the column exists.

SQL Update Statement

The SQL UPDATE statement is used to modify the data that is already in the database. The condition in the WHERE clause decides that which row is to be updated.

Syntax

1. UPDATE table_name
2. SET column1 = value1, column2 = value2, ...
3. WHERE condition;

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating single record

Update the column EMP_NAME and set the value to 'Emma' in the row where SALARY is 500000.

Syntax

1. UPDATE table_name

2. SET column_name = value
3. WHERE condition;

Query

1. UPDATE EMPLOYEE
2. SET EMP_NAME = 'Emma'
3. WHERE SALARY = 500000;

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Emma	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Updating multiple records

If you want to update multiple columns, you should separate each field assigned with a comma. In the EMPLOYEE table, update the column EMP_NAME to 'Kevin' and CITY to 'Boston' where EMP_ID is 5.

Syntax

1. UPDATE table_name
2. SET column_name = value1, column_name2 = value2
3. WHERE condition;

Query

1. UPDATE EMPLOYEE
2. SET EMP_NAME = 'Kevin', City = 'Boston'
3. WHERE EMP_ID = 5;

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Kevin	Boston	200000	36
6	Marry	Canada	600000	48

Without use of WHERE clause

If you want to update all row from a table, then you don't need to use the WHERE clause. In the EMPLOYEE table, update the column EMP_NAME as 'Harry'.

Syntax

1. UPDATE table_name
2. SET column_name = value1;

Query

1. UPDATE EMPLOYEE
2. SET EMP_NAME = 'Harry';

Output

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Harry	Chicago	200000	30
2	Harry	Austin	300000	26
3	Harry	Denver	100000	42
4	Harry	Washington	500000	29
5	Harry	Los angels	200000	36
6	Harry	Canada	600000	48

SQL DELETE Statement

The SQL DELETE statement is used to delete rows from a table. Generally, DELETE statement removes one or more records form a table.

Syntax

1. DELETE FROM table_name WHERE some_condition;

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Deleting Single Record

Delete the row from the table EMPLOYEE where EMP_NAME = 'Kristen'. This will delete only the fourth row.

Query

1. DELETE FROM EMPLOYEE
2. WHERE EMP_NAME = 'Kristen';

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Deleting Multiple Record

Delete the row from the EMPLOYEE table where AGE is 30. This will delete two rows(first and third row).

Query

1. DELETE FROM EMPLOYEE WHERE AGE= 30;

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

Delete all of the records

Delete all the row from the EMPLOYEE table. After this, no records left to display. The EMPLOYEE table will become empty.

Syntax

1. DELETE * FROM table_name;
2. or
3. DELETE FROM table_name;

Query

1. DELETE FROM EMPLOYEE;

Output: After executing this query, the EMPLOYEE table will look like:

EMP_ID	EMP_NAME	CITY	SALARY	AGE

Note: Using the condition in the WHERE clause, we can delete single as well as multiple records. If you want to delete all the records from the table, then you don't need to use the WHERE clause.

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

1. CREATE VIEW view_name AS
2. SELECT column1, column2.....
3. FROM table_name
4. WHERE condition;

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

1. CREATE VIEW DetailsView AS
2. SELECT NAME, ADDRESS
3. FROM Student_Details
4. WHERE STU_ID < 4;

Just like table query, we can query the view to view the data.

1. SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

1. CREATE VIEW MarksView AS
2. SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
3. FROM Student_Detail, Student_Mark
4. WHERE Student_Detail.NAME = Student_Marks.NAME;

To display data of View MarksView:

1. SELECT * FROM MarksView;

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

1. DROP VIEW view_name;

Example:

If we want to delete the View **MarksView**, we can do this as:

1. DROP VIEW MarksView;

SQL Index

- Indexes are special lookup tables. It is used to retrieve data from the database very fast.
- An Index is used to speed up select queries and where clauses. But it shows down the data input with insert and update statements. Indexes can be created or dropped without affecting the data.
- An index in a database is just like an index in the back of a book.

- **For example:** When you reference all pages in a book that discusses a certain topic, you first have to refer to the index, which alphabetically lists all the topics and then referred to one or more specific page numbers.

1. Create Index statement

It is used to create an index on a table. It allows duplicate value.

Syntax

1. CREATE INDEX index_name
2. ON table_name (column1, column2, ...);

Example

1. CREATE INDEX idx_name
2. ON Persons (LastName, FirstName);

2. Unique Index statement

It is used to create a unique index on a table. It does not allow duplicate value.

Syntax

1. CREATE UNIQUE INDEX index_name
2. ON table_name (column1, column2, ...);

Example

1. CREATE UNIQUE INDEX websites_idx
2. ON websites (site_name);

3. Drop Index Statement

It is used to delete an index in a table.

Syntax

1. DROP INDEX index_name;

Example

1. DROP INDEX websites_idx;

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. Subqueries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

Syntax

1. SELECT column_name
2. FROM table_name
3. WHERE column_name expression operator
4. (SELECT column_name from table_name WHERE ...);

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

1. SELECT *
2. FROM EMPLOYEE
3. WHERE ID IN (SELECT ID
4. FROM EMPLOYEE
5. WHERE SALARY > 4500);

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

2. Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

1. INSERT INTO table_name (column1, column2, column3....)
2. SELECT *
3. FROM table_name
4. WHERE VALUE OPERATOR

Example

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
1. INSERT INTO EMPLOYEE_BKP
2.   SELECT * FROM EMPLOYEE
3.   WHERE ID IN (SELECT ID
4.   FROM EMPLOYEE);
```

3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
1. UPDATE table
2. SET column_name = new_value
3. WHERE VALUE OPERATOR
4.   (SELECT COLUMN_NAME
5.   FROM TABLE_NAME
6.   WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
1. UPDATE EMPLOYEE
2.   SET SALARY = SALARY * 0.25
3.   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
4.     WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

4. Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
1. DELETE FROM TABLE_NAME
2. WHERE VALUE OPERATOR
3.   (SELECT COLUMN_NAME
4.   FROM TABLE_NAME
5.   WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

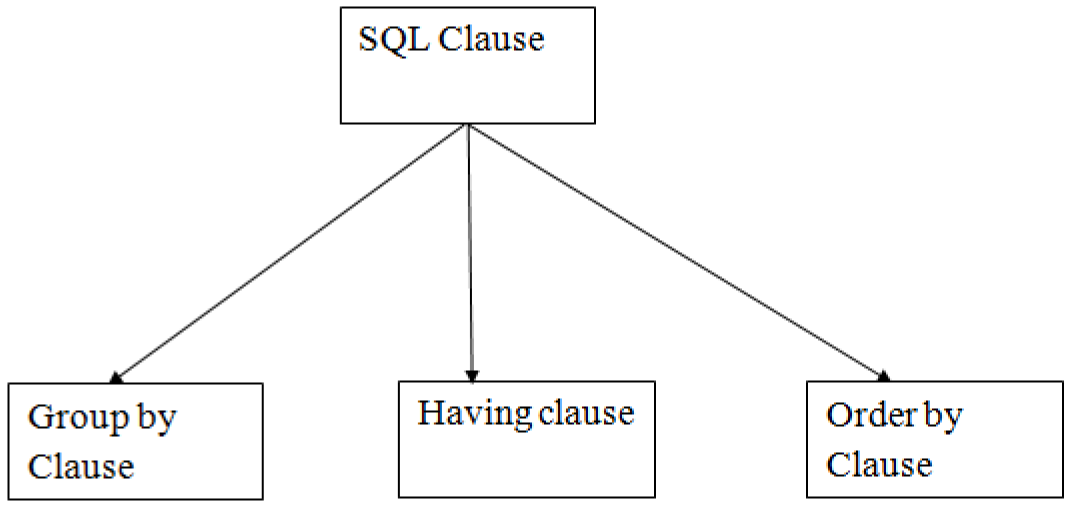
1. DELETE FROM EMPLOYEE
2. WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
3. WHERE AGE >= 29);

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

SQL Clauses

The following are the various SQL clauses:



1. GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax

1. SELECT column
2. FROM table_name
3. WHERE conditions
4. GROUP BY column
5. ORDER BY column

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST

Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example:

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;

Output:

```
Com1    5
Com2    3
Com3    2
```

2. HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

1. SELECT column1, column2
2. FROM table_name
3. WHERE conditions
4. GROUP BY column1, column2
5. HAVING conditions
6. ORDER BY column1, column2;

Example:

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;

Output:

```
Com1    5
Com2    3
```

3. ORDER BY

- The ORDER BY clause sorts the result-set in ascending or descending order.

- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

1. SELECT column1, column2
2. FROM table_name
3. WHERE condition
4. ORDER BY column1, column2... ASC|DESC;

Where

ASC: It is used to sort the result set in ascending order by expression.

DESC: It sorts the result set in descending order by expression.

Example: Sorting Results in Ascending Order

Table:

CUSTOMER

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
23	David	Bangkok
34	Alina	Dubai
45	John	UK
56	Harry	US

Enter the following SQL statement:

1. SELECT *
2. FROM CUSTOMER
3. ORDER BY NAME;

Output:

CUSTOMER_ID	NAME	ADDRESS
34	Alina	Dubai
23	David	Bangkok
56	Harry	US
45	John	UK
12	Kathrin	US

Example: Sorting Results in Descending Order

Using the above CUSTOMER table

1. SELECT *
2. FROM CUSTOMER
3. ORDER BY NAME DESC;

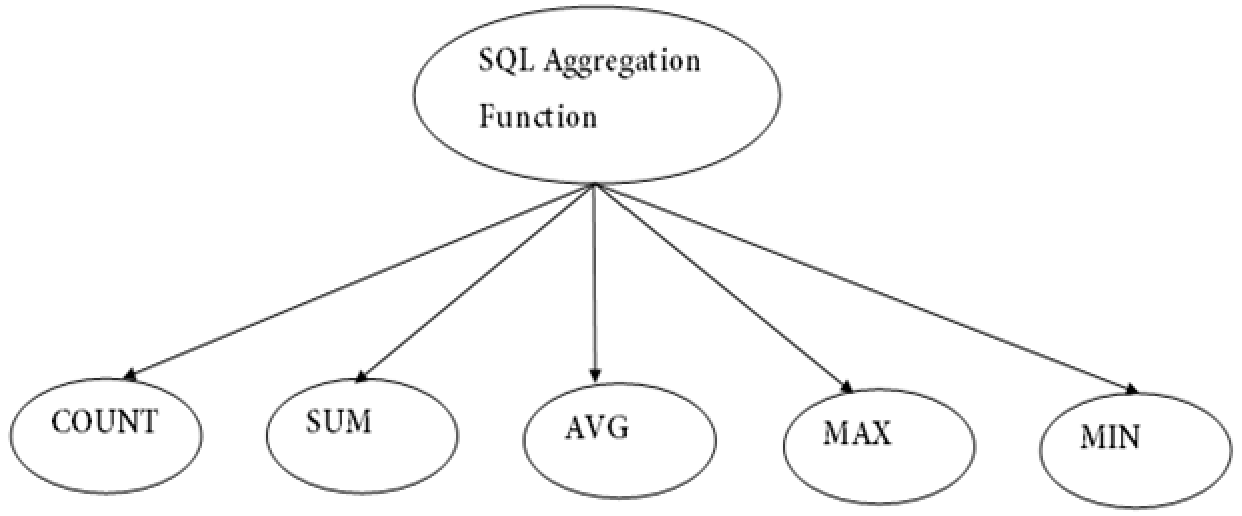
Output:

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
45	John	UK
56	Harry	US
23	David	Bangkok
34	Alina	Dubai

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

1. COUNT(*)
2. or
3. COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20

Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;

Output:

```
10
```

Example: COUNT with WHERE

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE>=20;

Output:

```
7
```

Example: COUNT() with DISTINCT

1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;

Output:

```
3
```

Example: COUNT() with GROUP BY

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;

Output:

```
Com1    5
Com2    3
Com3    2
```

Example: COUNT() with HAVING

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;

Output:

```
Com1     5
Com2     3
```

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

1. SUM()
2. or
3. SUM([ALL|DISTINCT] expression)

Example: SUM()

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;

Output:

```
670
```

Example: SUM() with WHERE

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3;

Output:

```
320
```

Example: SUM() with GROUP BY

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3
4. GROUP BY COMPANY;

Output:

```
Com1     150
Com2     170
```

Example: SUM() with HAVING

1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=170;

Output:

```
Com1     335
Com3     170
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

1. AVG()
2. or
3. AVG([ALL|DISTINCT] expression)

Example:

1. SELECT AVG(COST)
2. FROM PRODUCT_MAST;

Output:

```
67.00
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

1. SELECT MAX(RATE)
2. FROM PRODUCT_MAST;

```
30
```

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

1. MIN()
2. or
3. MIN([ALL|DISTINCT] expression)

Example:

1. SELECT MIN(RATE)
2. FROM PRODUCT_MAST;

Output:

```
10
```

SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30

2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. INNER JOIN table2
4. ON table1.matching_column = table2.matching_column;

Query

1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. INNER JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.

Syntax

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. LEFT JOIN table2
4. ON table1.matching_column = table2.matching_column;

Query

1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. LEFT JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.

Syntax

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. RIGHT JOIN table2
4. ON table1.matching_column = table2.matching_column;

Query

1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. RIGHT JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

4. FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

1. SELECT table1.column1, table1.column2, table2.column1,....
2. FROM table1
3. FULL JOIN table2
4. ON table1.matching_column = table2.matching_column;

Query

1. SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
2. FROM EMPLOYEE
3. FULL JOIN PROJECT
4. ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

SQL Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

1. SELECT column_name FROM table1
2. UNION
3. SELECT column_name FROM table2;

Example:

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan

5	David

Union SQL query will be:

1. SELECT * FROM First
2. UNION
3. SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

1. SELECT column_name FROM table1
2. UNION ALL
3. SELECT column_name FROM table2;

Example: Using the above First and Second table.

Union All query will be like:

1. SELECT * FROM First
2. UNION ALL
3. SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.

- It has no duplicates and it arranges the data in ascending order by default.

Syntax

1. SELECT column_name FROM table1
2. INTERSECT
3. SELECT column_name FROM table2;

Example:

Using the above First and Second table.

Intersect query will be:

1. SELECT * FROM First
2. INTERSECT
3. SELECT * FROM Second;

The resultset table will look like:

ID	NAME
3	Jackson

4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

1. SELECT column_name FROM table1
2. MINUS
3. SELECT column_name FROM table2;

Example

Using the above First and Second table.

Minus query will be:

1. SELECT * FROM First
2. MINUS
3. SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry