# DBMS

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

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

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

## 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 Database Management System

A database management system has following characteristics:

1. **Data stored into Tables:** Data is never directly stored into the database. Data is stored into tables, created inside the database. DBMS also allows to have relationships between tables which makes the data more meaningful and connected. You can easily understand what type of data is stored where by looking at all the tables created in a database.
2. **Reduced Redundancy:** In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows **Normalisation** which divides the data in such a way that repetition is minimum.
3. **Data Consistency:** On Live data, i.e. data that is being continuosly updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.
4. **Support Multiple user and Concurrent Access:** DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.
5. **Query Language:** DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.
6. **Security:** The DBMS also takes care of the security of data, protecting the data from un-authorised access. In a typical DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access.
7. DBMS supports **transactions**, which allows us to better handle and manage data integrity in real world applications where multi-threading is extensively used.

## 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.

# 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.

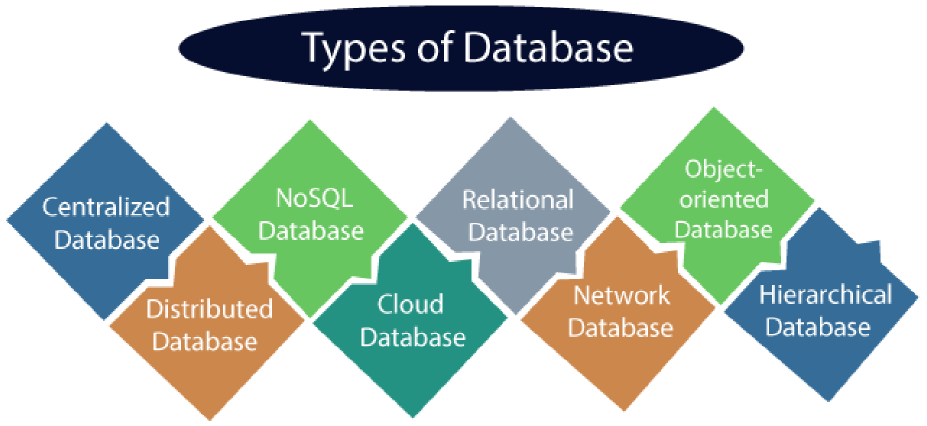
**Domain integrity**: It enforces valid entries for a given column by restricting the type, the format, or the range of values.

**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.

## Types of Database

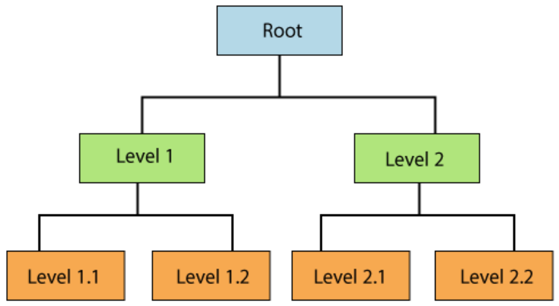
There are various types of databases used for storing different varieties of data:



**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

**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.

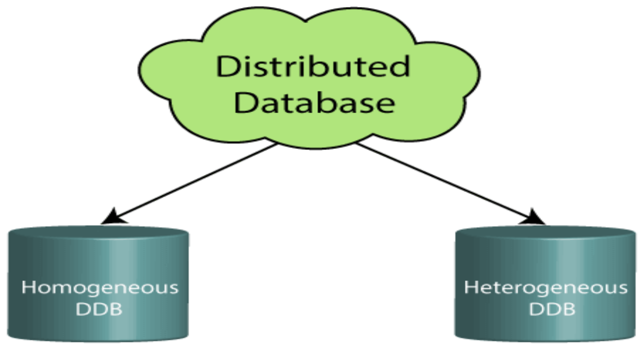
**Hierarchical Databases:** It is the type of database that stores data in the form of parent-children relationship nodes. Here, it organizes data in a tree-like structure.Data get stored in the form of records that are connected via links. Each child record in the tree will contain only one parent. On the other hand, each parent record can have multiple child records.



Hierarchical Database

**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.

**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.

**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.

## 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. |

## DBMS vs. File System

There are following differences between DBMS and File system:

| DBMS | File System |
|---|---|
| DBMS is a collection of data. In DBMS, the user is not | File system is a collection of data. In this system, the user has to write the procedures for |

| | |
|---|---|
| required to write the procedures. | managing the database. |
| DBMS gives an abstract view of data that hides the details. | File system provides the detail of the data representation and storage of data. |
| DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure. | 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 lost. |
| DBMS provides a good protection mechanism. | It is very difficult to protect a file under the file system. |
| DBMS contains a wide variety of sophisticated techniques to store and retrieve the data. | File system can't efficiently store and retrieve the data. |
| 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 other deleting some information or updating some information. |

## **View of Data**

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

## **Data Abstraction**

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

• **Physical level**. The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

**Logical level**. The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures.

**View level**. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.

## **Instances and Schemas :**

Databases change over time as information is inserted and deleted.

The collection of information stored in the database at a particular moment is called an instance of the database.

The overall design of the database is called the database schema

**Data Independence :**

It is the ability to modif y the a schema definition in one level without affecting a schema definition in the next higher level is called data independence.
There are two type :

**Logical Data Independence**

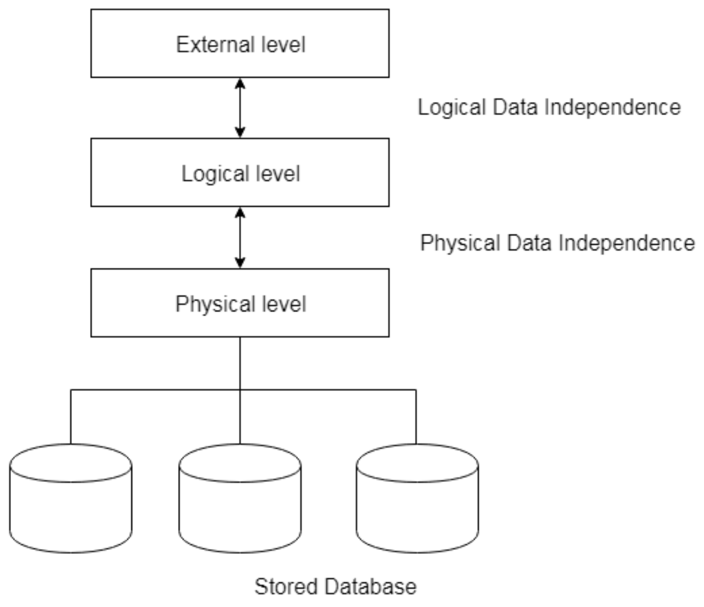
It is the ability to modify the logical schema without causing application programs to be rewritten. Modification at the logical level are necessary whenever the logical structure of the database is altered.

- 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.

**Physical Data Independence**

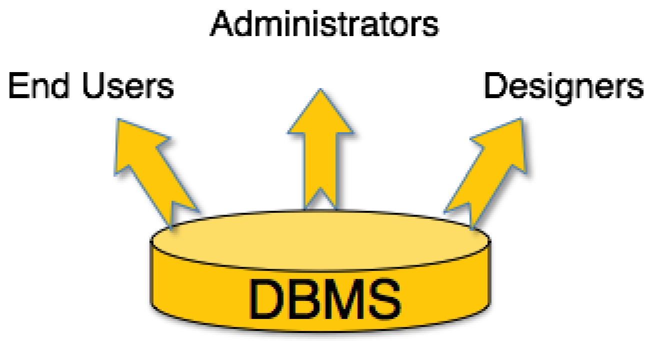
It is the ability to modify the physical schema without causing application programs to be rewritten. Modification at the physical level are necessary to improve the performane.

- 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.



**Users :**

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows −



- **Administrators** − Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.
- **Designers** − Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.
- **End Users** − End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

## Architecture :

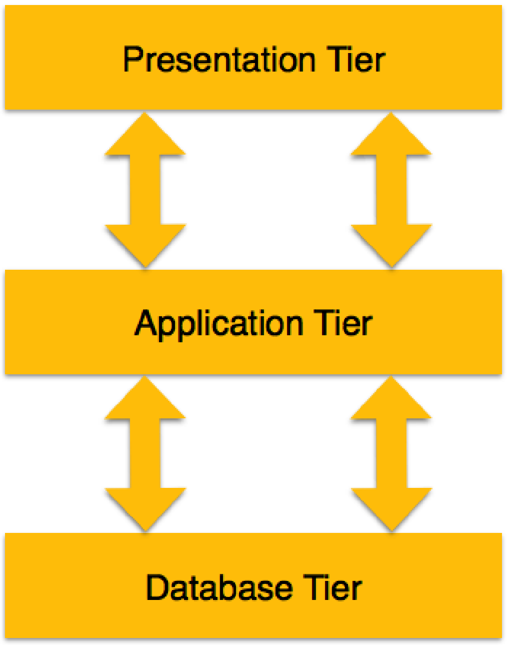
The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently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

**In 1-tier architecture**, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

**If the architecture of DBMS is 2-tier**, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

## **3-tier Architecture**

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

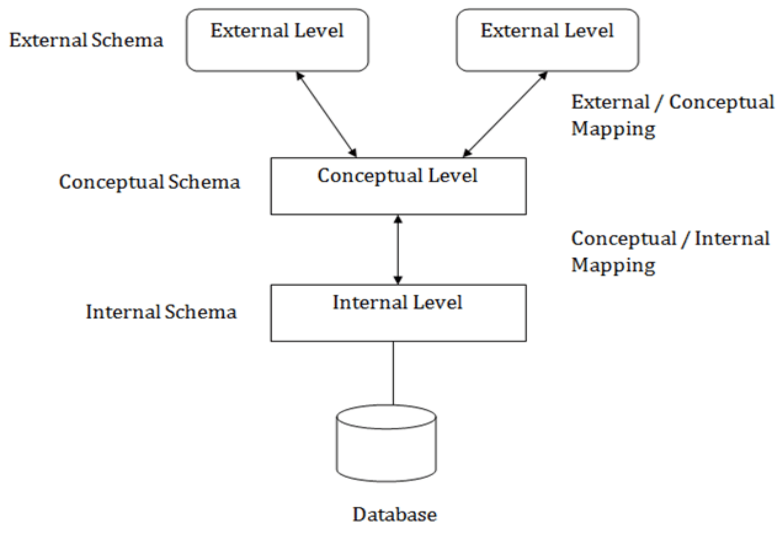


- **Database (Data) Tier** − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier** − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

## Three schema Architecture OR ANSI/SPARC Architecture OR Three –Level 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.

## 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.

## 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

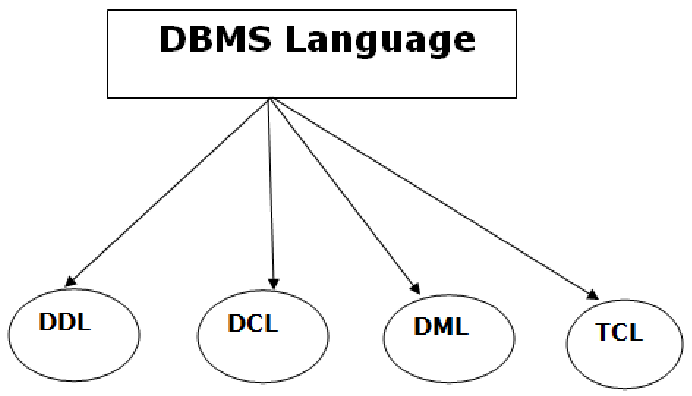
- At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
- An external schema is also known as view schema.
- Each view schema describes the database part that a particular user group is interested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

- The view schema describes the end user interaction with database systems

## 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.

### Data dictionary:

The data dictionary is considered to be a special type of table that can only be accessed and updated by the database system itself (not a regular user). The database system consults the data dictionary before reading or modifying actual data.

## **Database Administrator : (DBA)**

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA).

The functions of a DBA include:

• **Schema definition** : The DBA creates the original database schema by executing a set of data definition statements in the DDL.

• **Storage structure and access-method definition**.

• **Schema and physical-organization modification** :The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

• **Granting of authorization for data access** : By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

## **DBMS Database Models**

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

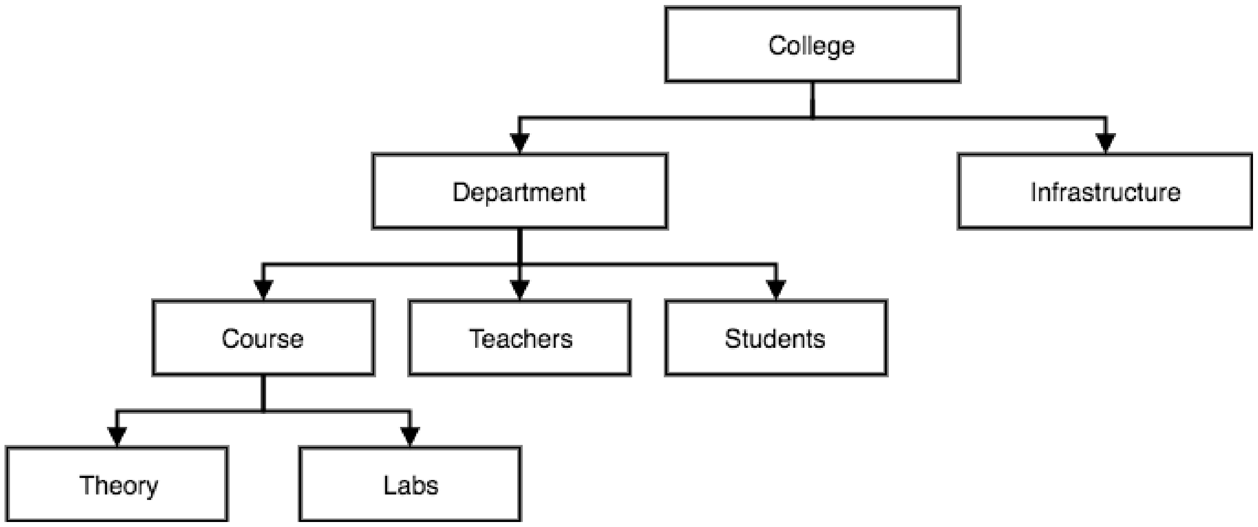
## Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.



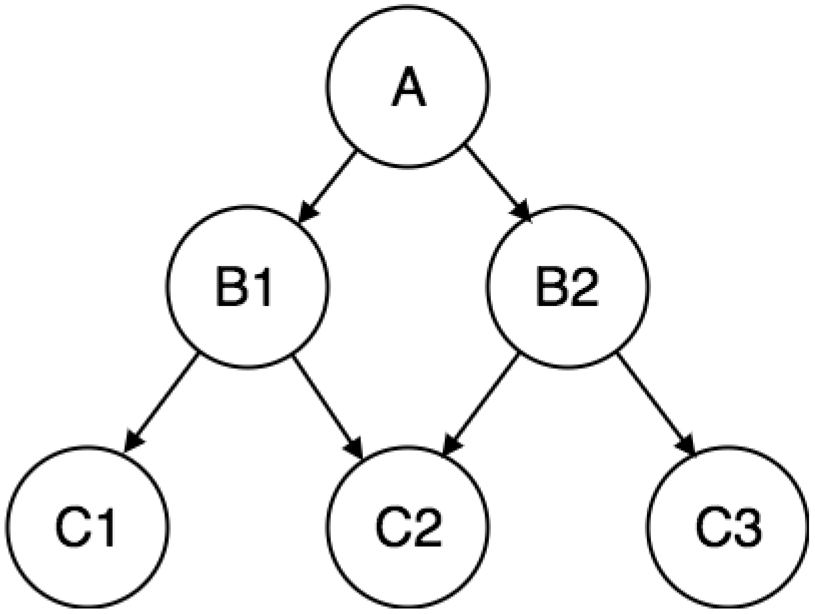
---

## Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



---

**Entity-relationship Model**

In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

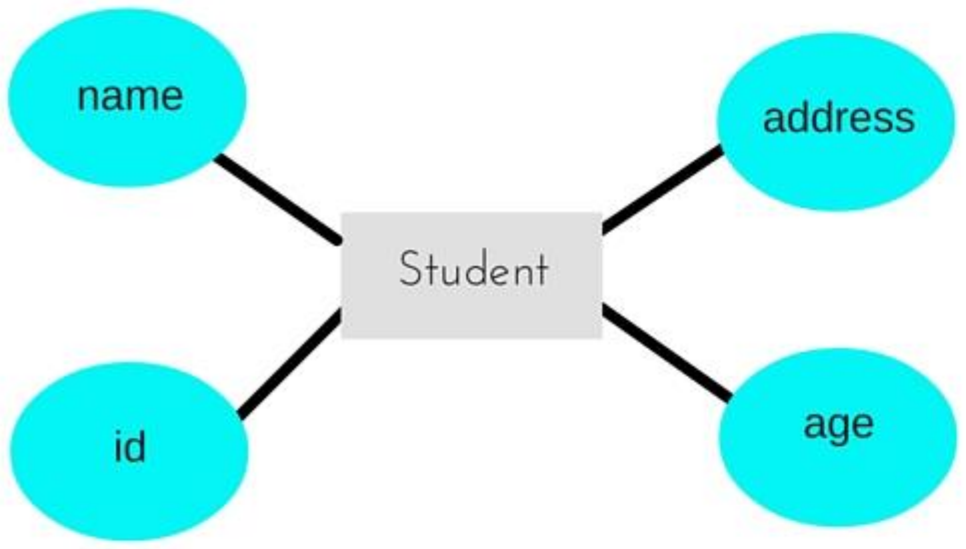
Different entities are related using relationships.

E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

This model is good to design a database, which can then be turned into tables in relational model(explained below).

Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

Relationships can also be of different types. To learn about E-R Diagrams in details, click on the link.



## Relational Model

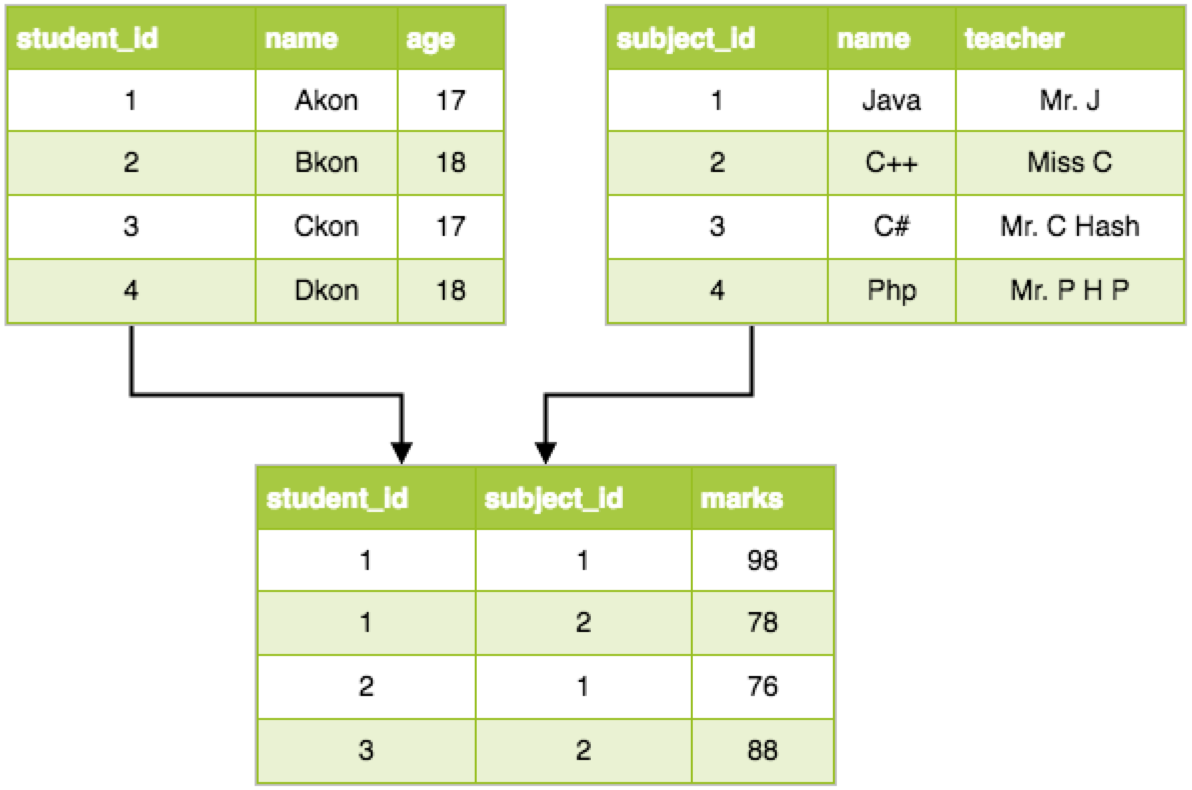
In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as **relations** in relational model.

In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.



| student_id | name | age |
|---|---|---|
| 1 | Akon | 17 |
| 2 | Bkon | 18 |
| 3 | Ckon | 17 |
| 4 | Dkon | 18 |

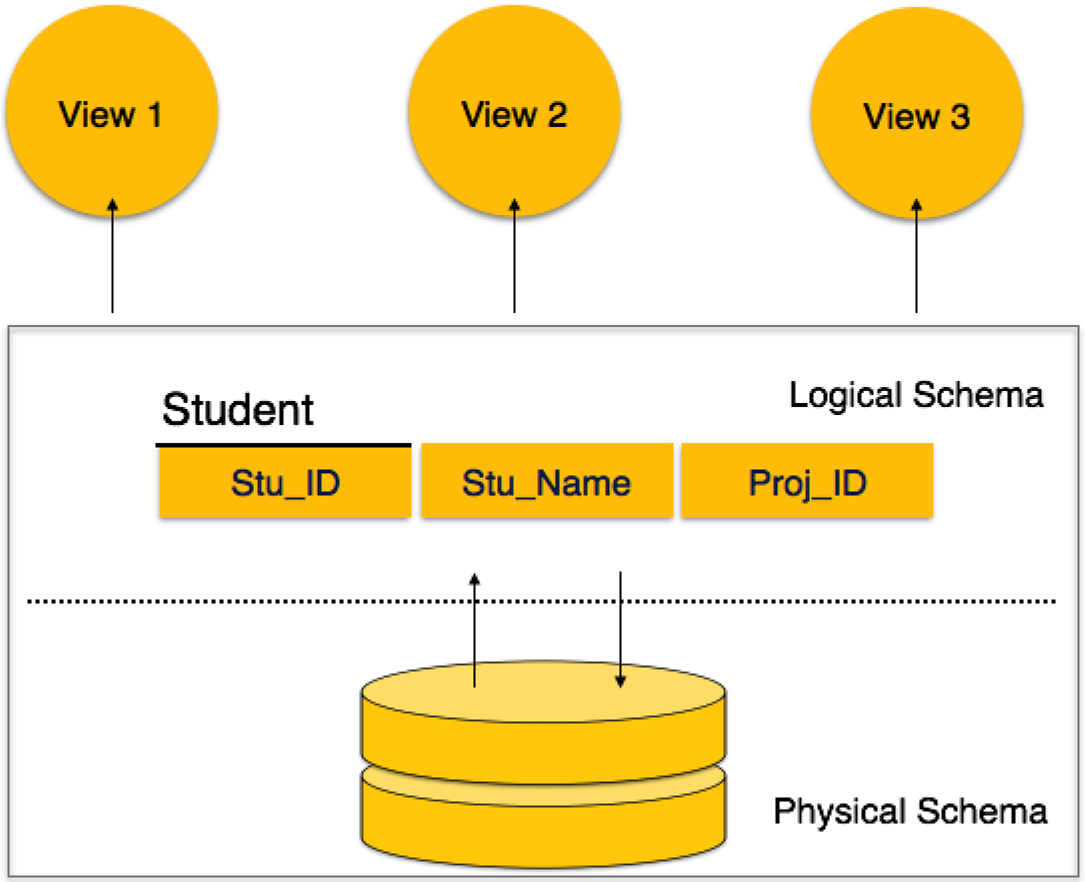
| subject_id | name | teacher |
|---|---|---|
| 1 | Java | Mr. J |
| 2 | C++ | Miss C |
| 3 | C# | Mr. C Hash |
| 4 | Php | Mr. P H P |

| student_id | subject_id | marks |
|---|---|---|
| 1 | 1 | 98 |
| 1 | 2 | 78 |
| 2 | 1 | 76 |
| 3 | 2 | 88 |

**Database Schema**

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

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories −

- **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

### Database Instance

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

## ER MODEL:

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

### Entity

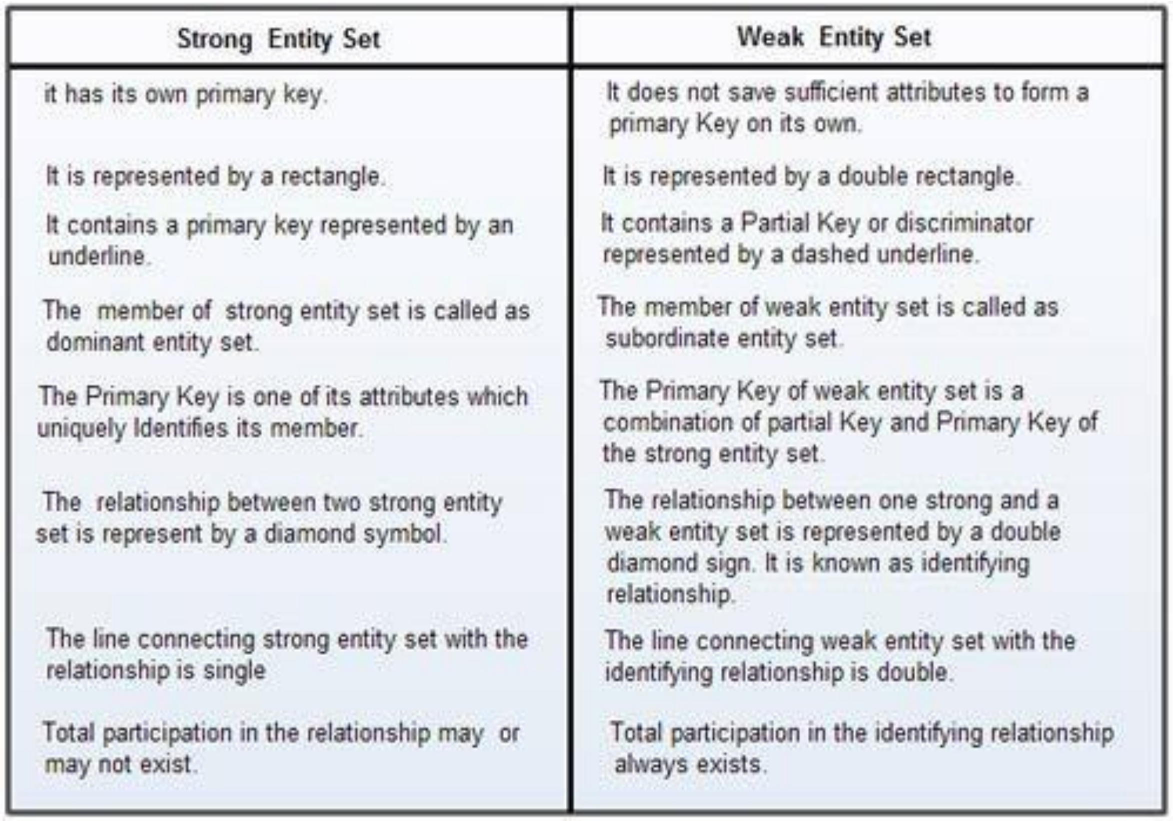
An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

### Definition of Strong Entity

The **Strong Entity** is the one whose existence does not depend on the existence of any other entity in a schema. It is denoted by a **single rectangle**. A strong entity always has the **primary key** in the set of attributes that describes the strong entity. It indicates that each entity in a strong entity set can be uniquely identified.

**Definition of Weak Entity**

A **Weak entity** is the one that depends on its owner entity i.e. a strong entity for its existence. A weak entity is denoted by the **double rectangle**. Weak entity do **not** have the **primary key** instead it has a **partial key** that uniquely discriminates the weak entities.



| Strong Entity Set | Weak Entity Set |
|---|---|
| it has its own primary key. | It does not save sufficient attributes to form a primary Key on its own. |
| It is represented by a rectangle. | It is represented by a double rectangle. |
| It contains a primary key represented by an underline. | It contains a Partial Key or discriminator represented by a dashed underline. |
| The member of strong entity set is called as dominant entity set. | The member of weak entity set is called as subordinate entity set. |
| The Primary Key is one of its attributes which uniquely Identifies its member. | The Primary Key of weak entity set is a combination of partial Key and Primary Key of the strong entity set. |
| The relationship between two strong entity set is represent by a diamond symbol. | The relationship between one strong and a weak entity set is represented by a double diamond sign. It is known as identifying relationship. |
| The line connecting strong entity set with the relationship is single | The line connecting weak entity set with the identifying relationship is double. |
| Total participation in the relationship may or may not exist. | Total participation in the identifying relationship always exists. |

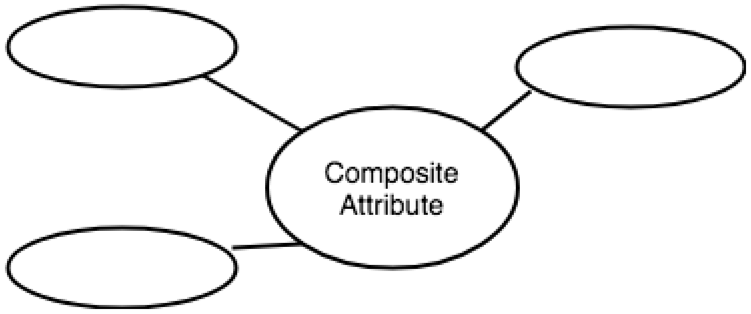
## Attributes

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

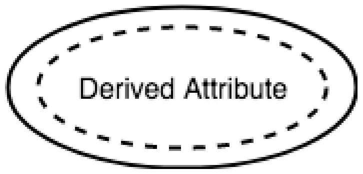
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

## Types of Attributes

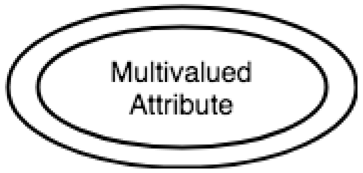
- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.



- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.



- **Single-value attribute** − Single-value attributes contain single value. For example − Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.



These attribute types can come together in a way like −

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

## **Introduction to Database Keys**

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

A Key can be a single attribute or a group of attributes, where the combination may act as a key.

.

Let's try to understand about all the keys using a simple example.

| **student_id** | **name** | **phone** | **age** |
|---|---|---|---|
| 1 | Akon | 9876723452 | 17 |
| 2 | Akon | 9991165674 | 19 |
| 3 | Bkon | 7898756543 | 18 |
| 4 | Ckon | 8987867898 | 19 |
| 5 | Dkon | 9990080080 | 17 |

Let's take a simple **Student** table, with fields student_id, name, phone and age.

---

Super Key

**Super Key** is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

In the table defined above super key would include student_id, (student_id, name), phoneetc.

Candidate Key

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key.

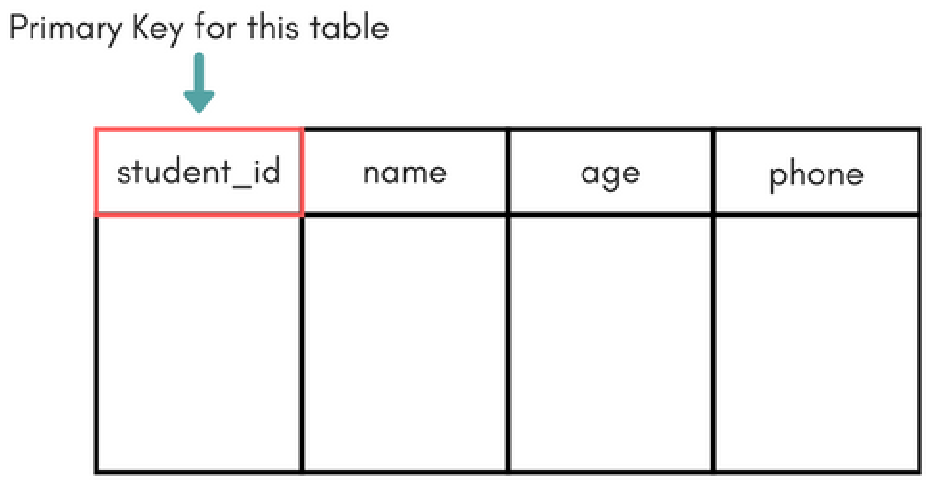
In our example, student_id and phone both are candidate keys for table **Student**.

- A candiate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one columns(attributes).

---

## Primary Key

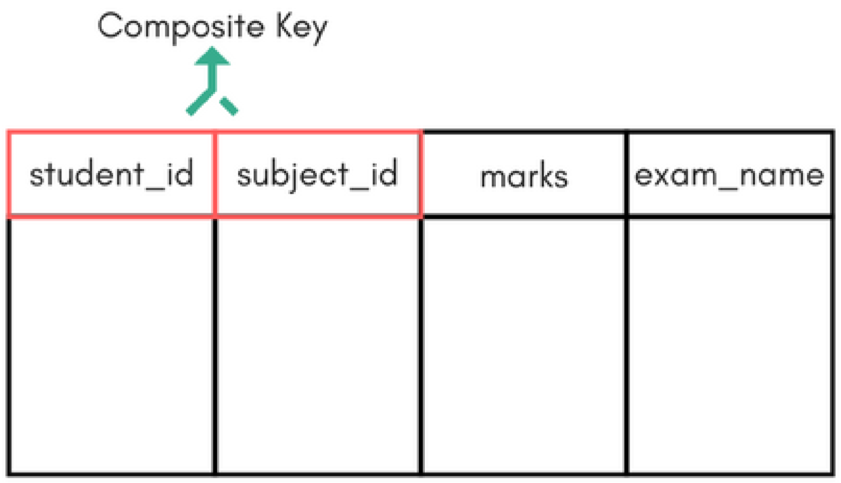
Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.



For the table **Student** we can make the student_id column as the primary key.

**Composite Key**

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independentely or individually.



Score Table - To save scores of the student for various subjects.

In the above picture we have a **Score** table which stores the marks scored by a student in a particular subject.

In this table student_id and subject_id together will form the primary key, hence it is a composite key.

**Secondary or Alternative key**

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

## Non-key Attributes

**Non-key** attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

## Non-prime Attributes

**Non-prime** Attributes are attributes other than **Primary Key attribute(s).**.

## FOREIGN KEY

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables.

In simple words you can say that, a foreign key in one table used to point primary key in another table.

Let us take an example to explain it:

Here are two tables first one is students table and second is orders table.

Here orders are given by students.

**First table:**

| S_Id | LastName | FirstName | CITY |
|---|---|---|---|
| 1 | MAURYA | AJEET | ALLAHABAD |

|  |  |  |  |
|---|---|---|---|
| 2 | JAISWAL | RATAN | GHAZIABAD |
| 3 | ARORA | SAUMYA | MODINAGAR |

**Second table:**

| O_Id | OrderNo | S_Id |
|---|---|---|
| 1 | 99586465 | 2 |
| 2 | 78466588 | 2 |
| 3 | 22354846 | 3 |
| 4 | 57698656 | 1 |

- The "S_Id" column in the "Students" table is the PRIMARY KEY in the "Students" table.
- The "S_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The foreign key constraint is generally prevents action that destroy links between tables.

It also prevents invalid data to enter in foreign key column.

## Relationship

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

## Relationship Set

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

## Degree of Relationship

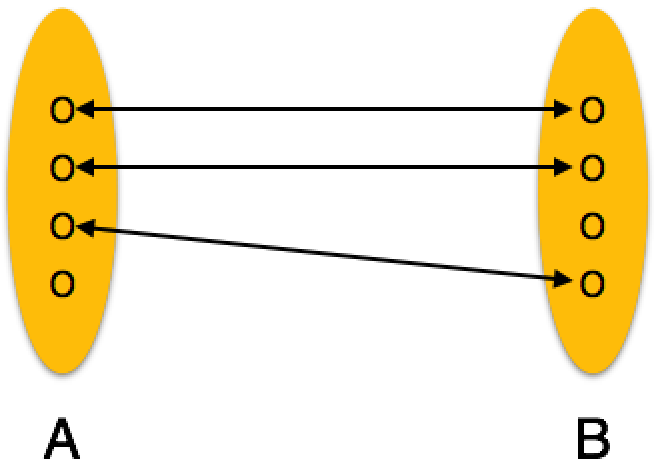
The number of participating entities in a relationship defines the degree of the relationship.

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

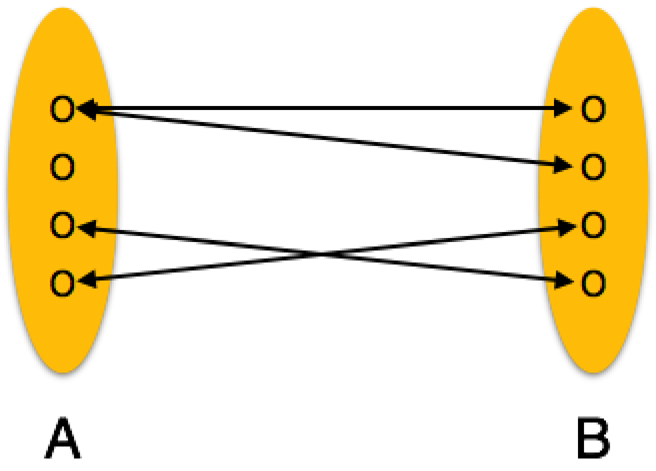
## Mapping Cardinalities

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

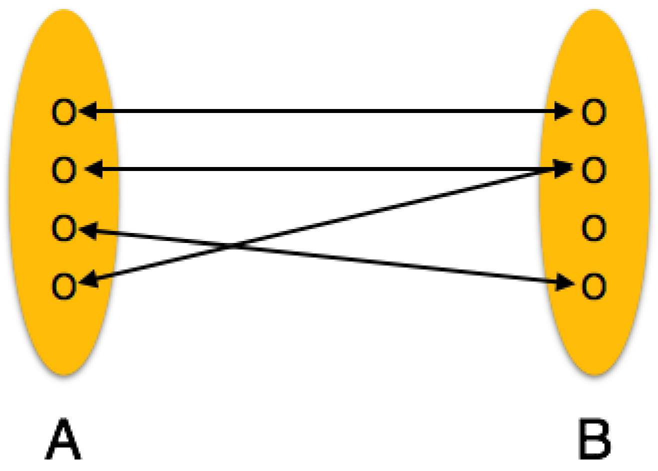
- **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



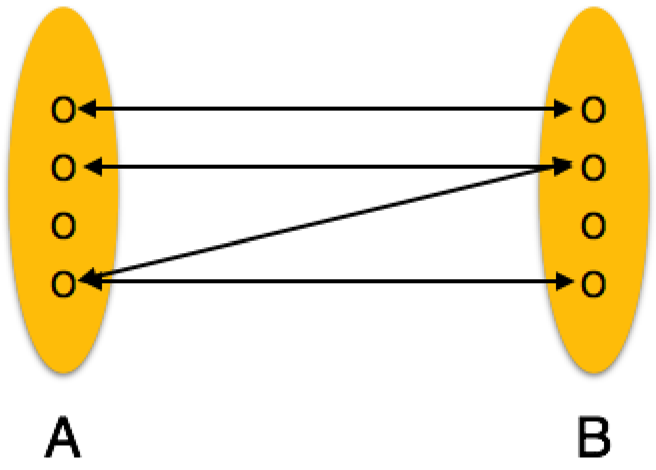
- **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



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

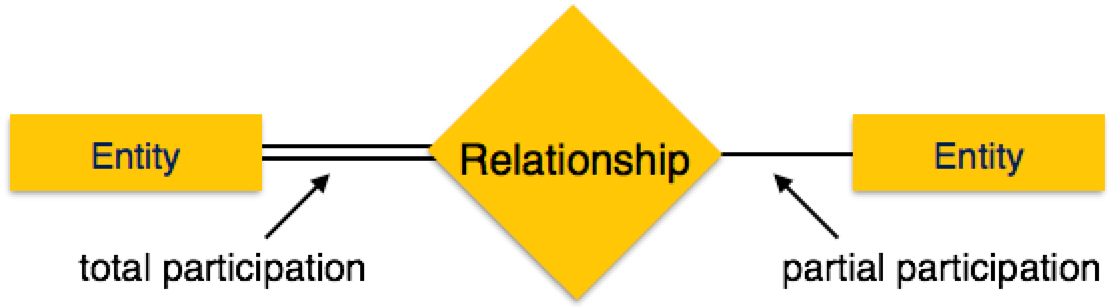


- **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



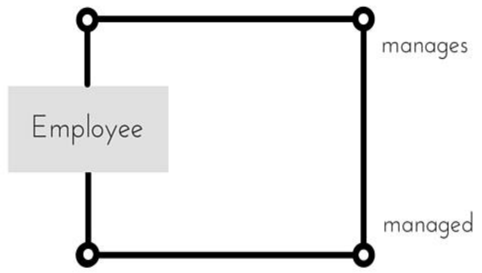
**Participation Constraints**

- **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.



**Recursive Relationship**

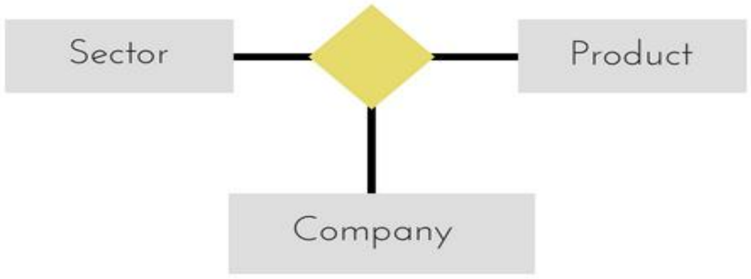
When an Entity is related with itself it is known as **Recursive** Relationship.



**Ternary Relationship**

Relationship of degree three is called Ternary relationship.

A Ternary relationship involves three entities. In such relationships we always consider two entites together and then look upon t



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

For example, in the diagram above, we have three related entities, **Company**, **Product** and **Sector**. To understand the relationship better or to define rules around the model, we should relate two entities and then derive the third one.

A **Company** produces many **Products**/ each product is produced by exactly one company.

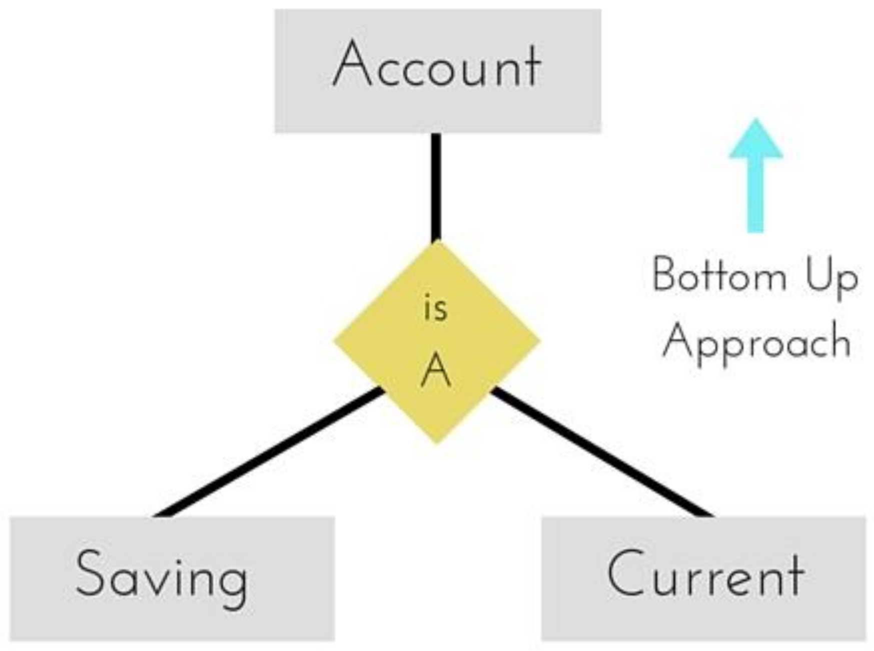
A **Company** operates in only one **Sector** / each sector has many companies operating in it.

Considering the above two rules or relationships, we see that although the complete relationship involves three entities, but we are looking at two entities at a time.

**Generalization**

**Generalization** is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

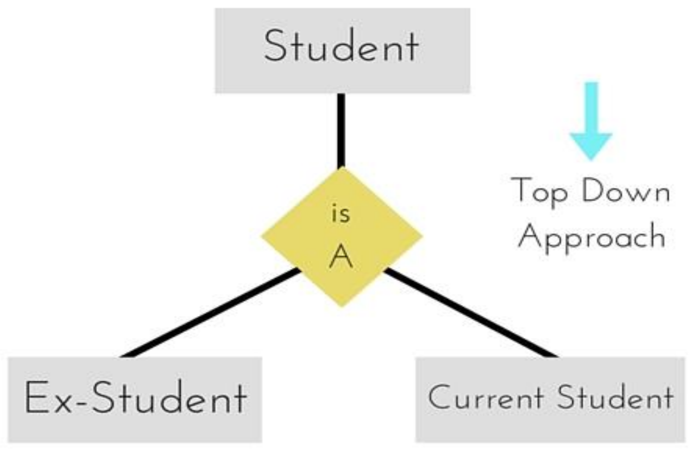
It's more like Superclass and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalised entity, in other words, sub-classes are combined to form a super-class.



For example, **Saving** and **Current** account types entities can be generalised and an entity with name **Account** can be created, which covers both.

**Specialization**

**Specialization** is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible.


Student
is
A
Top Down
Approach
Ex-Student
Current Student

Difference:

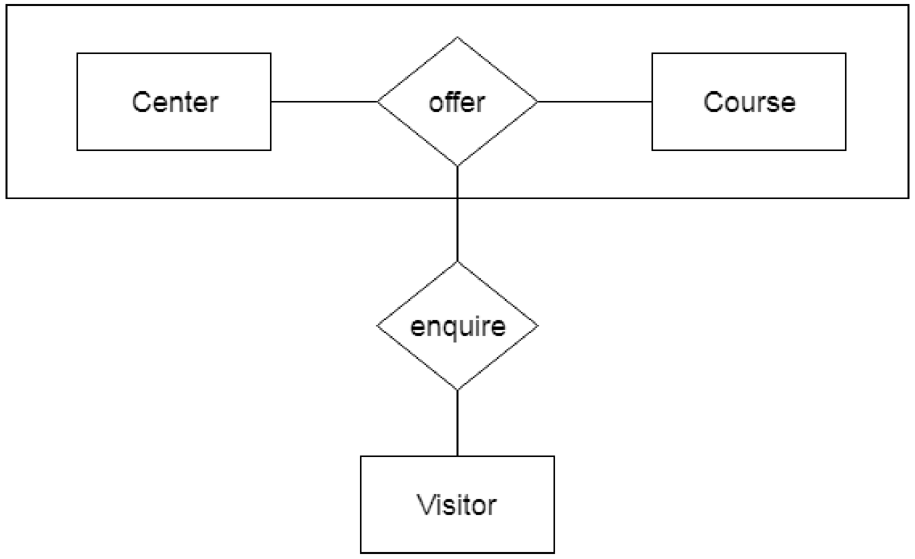
| BASIS FOR COMPARISON | GENERALIZATION | SPECIALIZATION |
|---|---|---|
| Basic | It proceeds in a bottom-up manner. | It proceeds in a top-down manner. |
| Function | Generalization extracts the common features of multiple entities to form a new entity. | Specialization splits an entity to form multiple new entities that inherit some feature of the splitting entity. |
| Entities | The higher level entity must have lower level entities. | The higher level entity may not have lower level entities. |
| Size | Generalization reduces the size of a schema. | Specialization increases the size of a schema. |

| BASIS FOR COMPARISON | GENERALIZATION | SPECIALIZATION |
|---|---|---|
| Application | Generalization entities on group of entities. | Specialization is applied on a single entity. |
| Result | Generalization results in forming a single entity from multiple entities. | Specialization results in forming the multiple entity from a single entity. |

## 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.



.

## Constraints

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

- Key constraints
- Domain constraints
- Referential integrity constraints

### Key Constraints

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

Key constraints force that −

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

Key constraints are also referred to as Entity Constraints.

### Domain Constraints

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

### Referential integrity Constraints

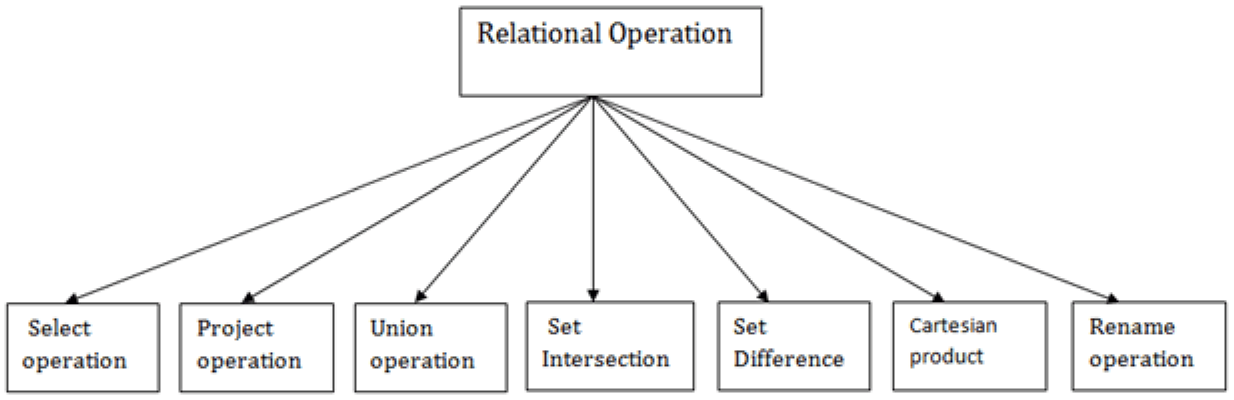
Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

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

## Relational Algebra

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

### 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).

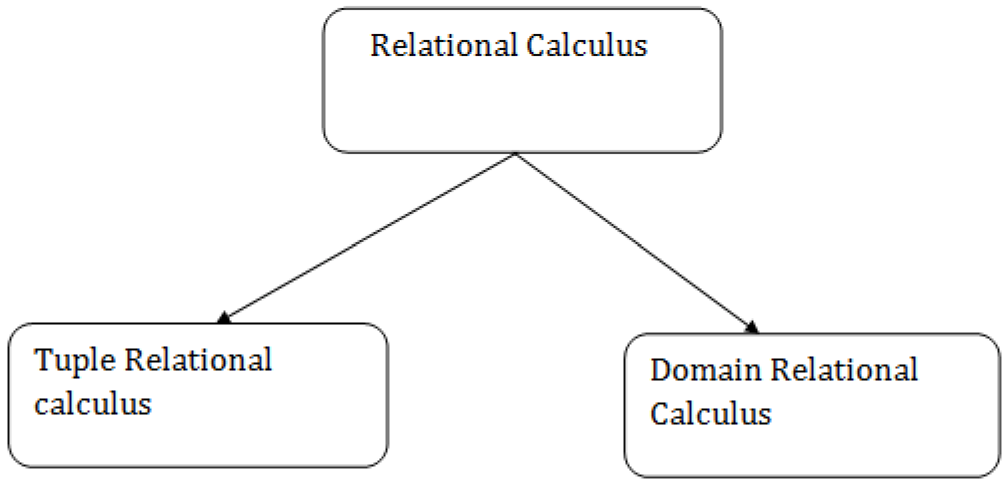
**Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

# 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.

**Functional Dependency**

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

## Types of Functional dependency

### 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.

## 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

### Fully-functionally Dependency

An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

For example, an attribute Q is fully functional dependent on another attribute P, if it is Functionally Dependent on P and not on any of the proper subset of P.

### Transitive Dependency

When an indirect relationship causes functional dependency it is called Transitive Dependency.

If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

**Multivalued Dependency**

When existence of one or more rows in a table implies one or more other rows in the same table, then the Multi-valued dependencies occur.

If a table has attributes P, Q and R, then Q and R are multi-valued facts of P.

It is represented by double arrow:

```
->->
```

For our example:

```
P->->Q
Q->->R
```

In the above case, Multivalued Dependency exists only if Q and R are independent attributes.

**Partial Dependency**

Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.

The 2nd Normal Form (2NF) eliminates the Partial Dependency. Let us see an example:

**<StudentProject>**

| StudentID | ProjectNo | StudentName | ProjectName |
|---|---|---|---|
| S01 | 199 | Katie | Geo Location |
| S02 | 120 | Ollie | Cluster Exploration |

In the above table, we have partial dependency; let us see how:

The prime key attributes are **StudentID** and **ProjectNo.**

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID** that makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectID**, which that the relation Partial Dependent.

## Inference Rule (IR)/Closer property/Amstrong Axioums:

- 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.

If $X \supseteq Y$ then $X \rightarrow Y$

## 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.

If $X \rightarrow Y$ then $XZ \rightarrow YZ$

## 3. Transitive Rule ($IR_3$)

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

If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

## 4. Union Rule ($IR_4$)

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

If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

## 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.

If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

## 6. Pseudo transitive Rule ($IR_6$)

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

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

## **Normalization of Database**

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating reduntant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

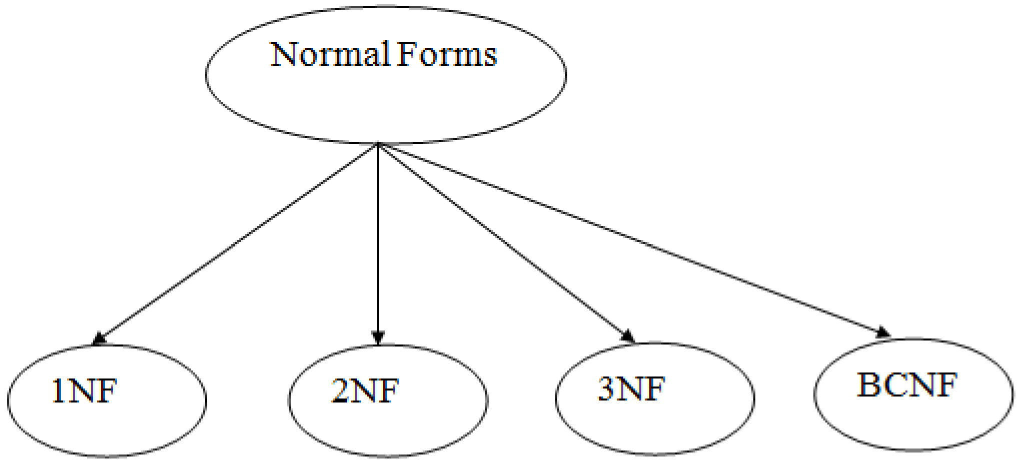
- **Update anomalies** − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

- **Deletion anomalies** − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** − We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

## Types of Normal Forms

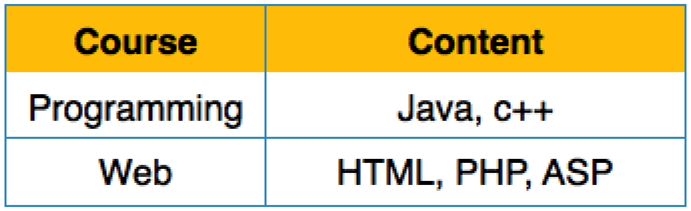
There are the four types of normal forms:



| 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. |

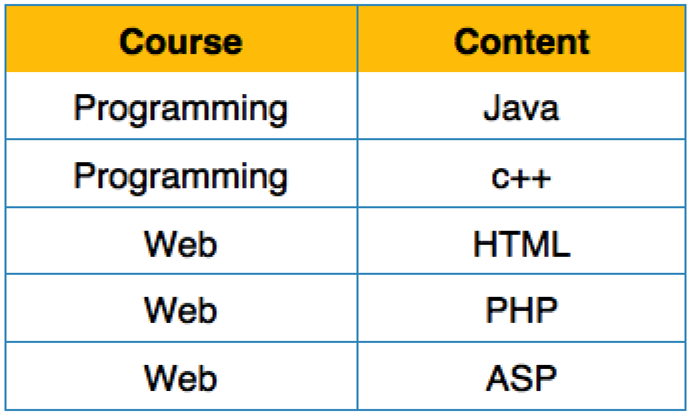
**First Normal Form**

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.



| Course | Content |
|---|---|
| Programming | Java, c++ |
| Web | HTML, PHP, ASP |

We re-arrange the relation (table) as below, to convert it to First Normal Form.



| Course | Content |
|---|---|
| Programming | Java |
| Programming | c++ |
| Web | HTML |
| Web | PHP |
| Web | ASP |

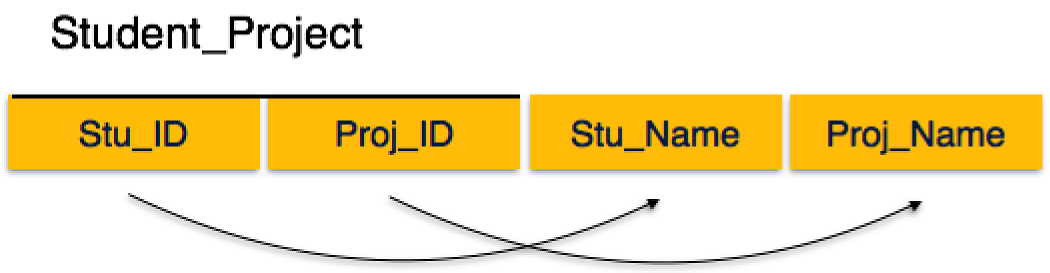
Each attribute must contain only a single value from its pre-defined domain.

## Second Normal Form

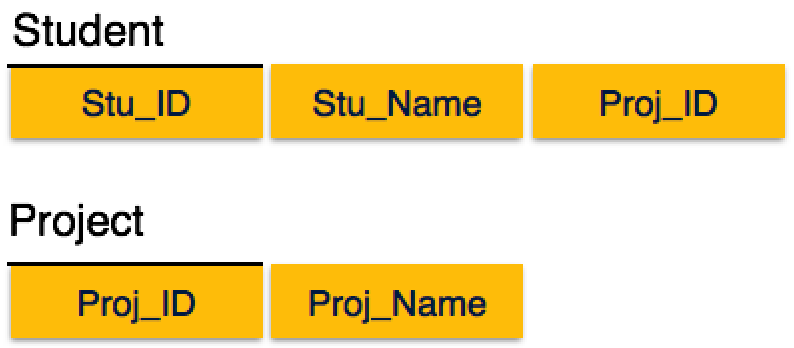
Before we learn about the second normal form, we need to understand the following −

- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

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



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

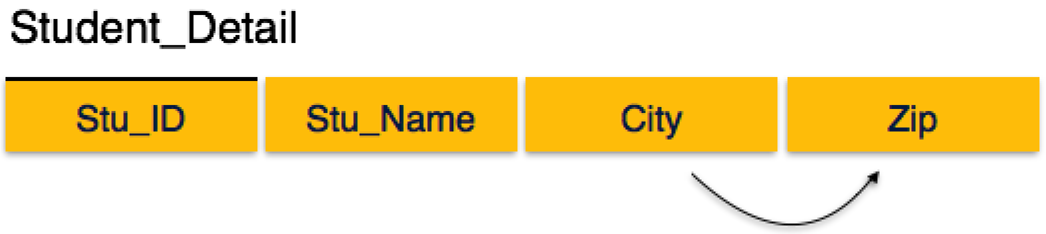


We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

## Third Normal Form

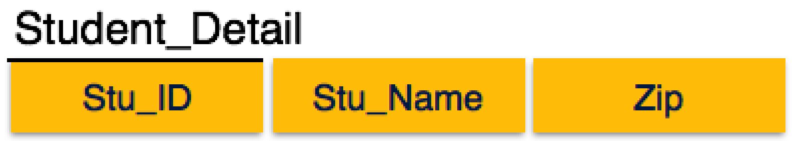
For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy −

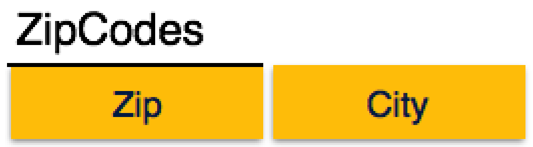
- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
    - X is a superkey or,
    - A is prime attribute.



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

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





## Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that −

- For any non-trivial functional dependency, X → A, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

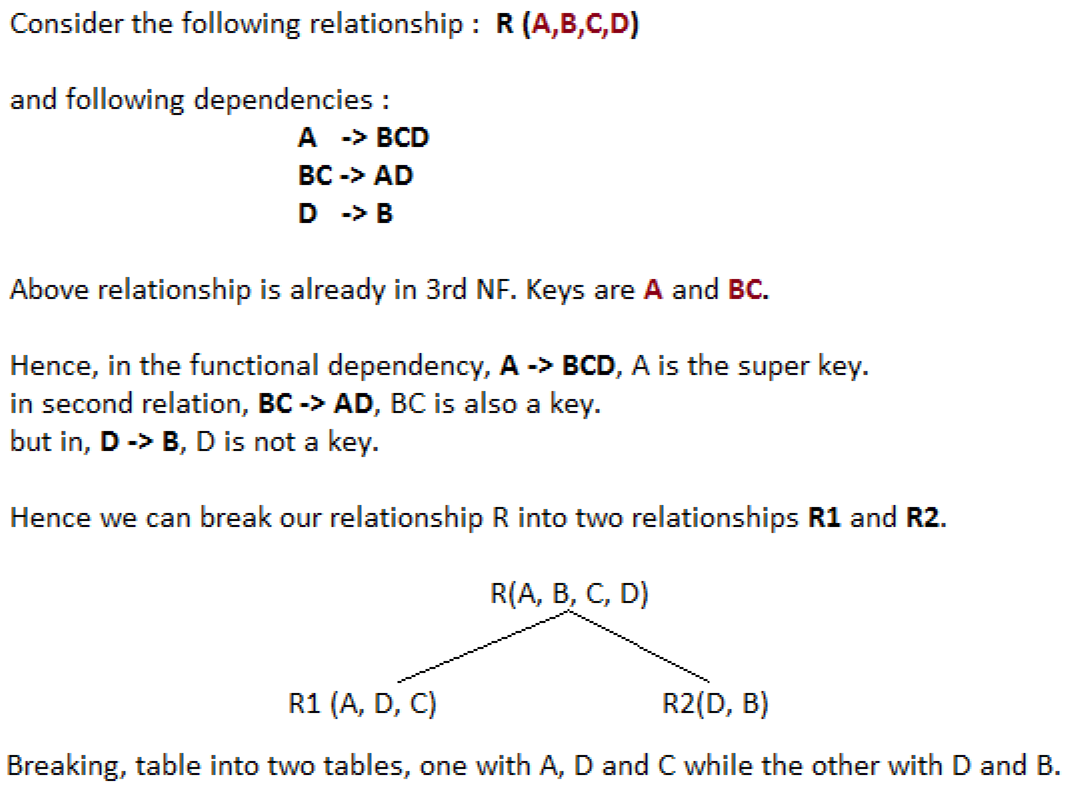
and

Zip → City

Which confirms that both the relations are in BCNF.

### Explanation

In the picture below, we have tried to explain BCNF in terms of relations.



Consider the following relationship : **R (A,B,C,D)**

and following dependencies :

**A -> BCD**
**BC -> AD**
**D -> B**

Above relationship is already in 3rd NF. Keys are **A** and **BC.**

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2.**

$$\begin{tikzcd}
 & R(A, B, C, D) \arrow[dl, dash] \arrow[dr, dash] & \\
R1 (A, D, C) & & R2(D, B)
\end{tikzcd}$$

Breaking, table into two tables, one with A, D and C while the other with D and B.

<u>**Key Differences Between 3NF and BCNF**</u>

1. 3NF states that no non-prime attribute must be transitively dependent on the candidate key of the relation. On the other hands, BCNF states that if a trivial functional dependency X -> Y exist for a relation; then X must be a super key.
2. 3NF can be obtained without sacrificing the dependency of relation. However, dependency may not be preserved while obtaining BCNF.
3. 3NF can be achieved without loosing any information from the old table whereas, while obtaining BCNF we may loose some information from the old table.

## **Why bcnf is stronger than 3nf?**

A relation R is in 3NF if and only if every dependency A->B satisfied by R meets at least ONE of the following criteria:

1. A->B is trivial (i.e. B is a subset of A)

2. A is a superkey

3. B is a subset of a candidate key

BCNF doesn't permit the third of these options. Therefore BCNF is said to be stronger than 3NF because 3NF permits some dependencies which BCNF does not.

# 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 |

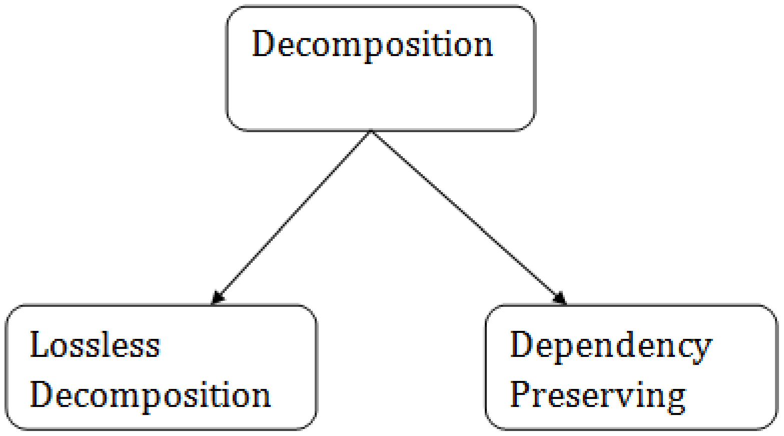
**P3**

| 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).

## Join

**Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

We will briefly describe various join types in the following sections.

### Theta (θ) Join

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

**Notation**

$R1 \bowtie_{\theta} R2$

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don’t have anything in common, that is $R1 \cap R2 = \Phi$.

Theta join can use all kinds of comparison operators.

| **Student** | | |
|---|---|---|
| **SID** | **Name** | **Std** |
| 101 | Alex | 10 |
| 102 | Maria | 11 |

| Subjects | |
|---|---|
| **Class** | **Subject** |
| 10 | Math |
| 10 | English |
| 11 | Music |
| 11 | Sports |

Student_Detail −

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

| Student_detail | | | | |
|---|---|---|---|---|
| **SID** | **Name** | **Std** | **Class** | **Subject** |
| 101 | Alex | 10 | 10 | Math |

| | | | | |
|---|---|---|---|---|
| 101 | Alex | 10 | 10 | English |
| 102 | Maria | 11 | 11 | Music |
| 102 | Maria | 11 | 11 | Sports |

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

**Natural Join (⋈)**

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

| **Courses** | | |
|---|---|---|
| **CID** | **Course** | **Dept** |

| | | |
|---|---|---|
| CS01 | Database | CS |
| ME01 | Mechanics | ME |
| EE01 | Electronics | EE |

**HoD**

| **Dept** | **Head** |
|---|---|
| CS | Alex |
| ME | Maya |
| EE | Mira |

**Courses ⋈HoD**

| **Dept** | **CID** | **Course** | **Head** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| CS | CS01 | Database | Alex |
| ME | ME01 | Mechanics | Maya |
| EE | EE01 | Electronics | Mira |

**Outer Joins**

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

**Left Outer Join(R ⟕ S)**

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

| **Left** | |
|---|---|
| **A** | **B** |

| | |
|---|---|
| 100 | Database |
| 101 | Mechanics |
| 102 | Electronics |

**Right**

| **A** | **B** |
|---|---|
| 100 | Alex |
| 102 | Maya |
| 104 | Mira |

**Courses ⟕ HoD**

| **A** | **B** | **C** | **D** |
|---|---|---|---|

|  |  |  |  |
|---|---|---|---|
| 100 | Database | 100 | Alex |
| 101 | Mechanics | --- | --- |
| 102 | Electronics | 102 | Maya |

**<u>Right Outer Join: ( R ⟖ S )</u>**

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

| **Courses ⟖ HoD** |
|---|

| A | B | C | D |
|---|---|---|---|
| 100 | Database | 100 | Alex |
| 102 | Electronics | 102 | Maya |
| --- | --- | 104 | Mira |

**Full Outer Join: ( R ⟗ S)**

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

| **Courses ⟗ HoD** | | | |
|---|---|---|---|
| **A** | **B** | **C** | **D** |
| 100 | Database | 100 | Alex |
| 101 | Mechanics | --- | --- |

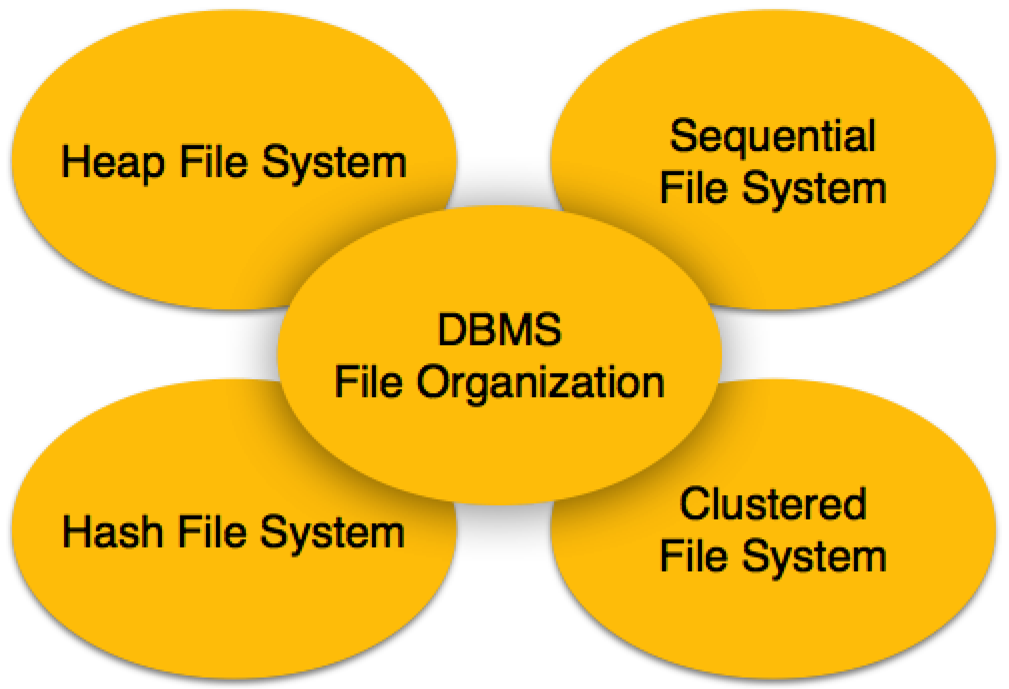
| | | | |
|---|---|---|---|
| 102 | Electronics | 102 | Maya |
| --- | --- | 104 | Mira |

## 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.



**Heap File Organization**

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

**Sequential File Organization**

Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

**Hash File Organization**

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

**Clustered File Organization**

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

**Indexing :**

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

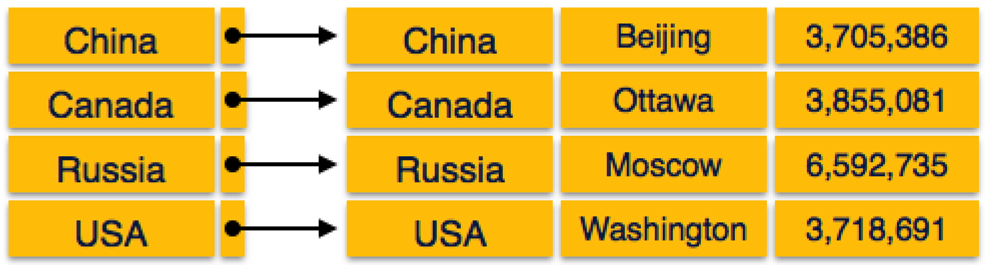
- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

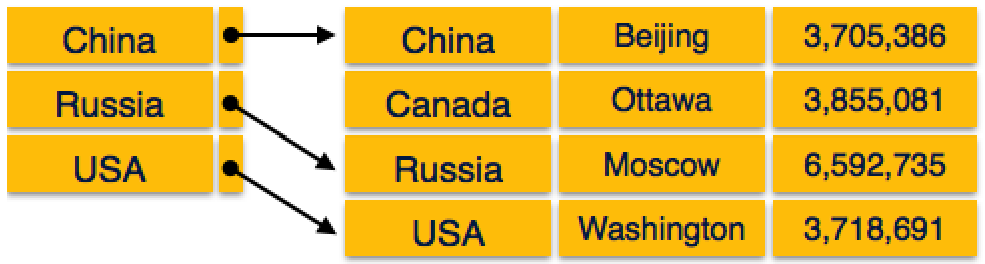
**Dense Index**

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



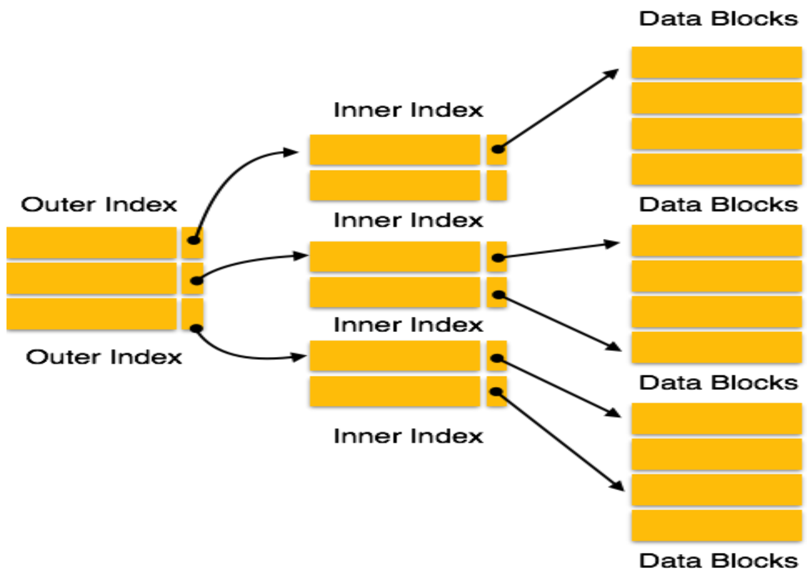
**Sparse Index**

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



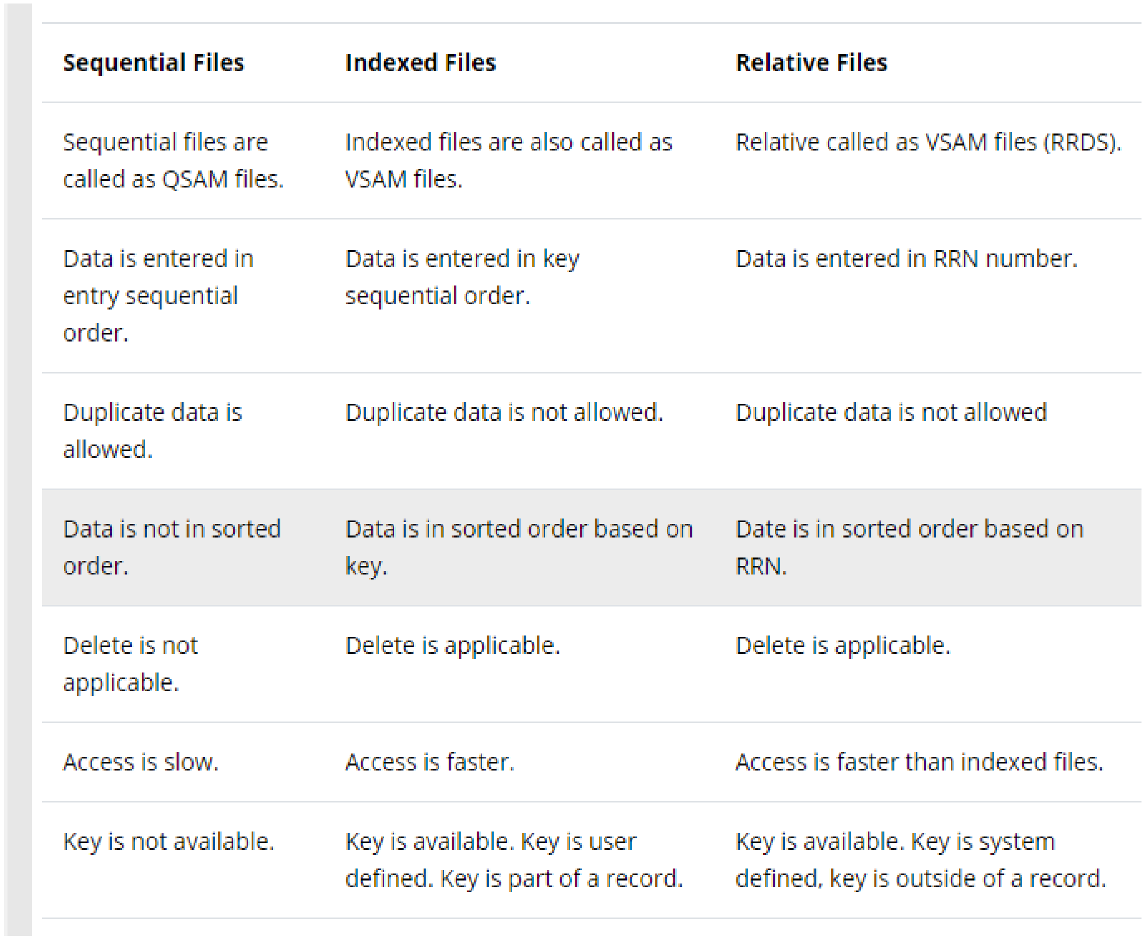
**Multilevel Index**

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

**Difference between different file :**



| Sequential Files | Indexed Files | Relative Files |
|---|---|---|
| Sequential files are called as QSAM files. | Indexed files are also called as VSAM files. | Relative called as VSAM files (RRDS). |
| Data is entered in entry sequential order. | Data is entered in key sequential order. | Data is entered in RRN number. |
| Duplicate data is allowed. | Duplicate data is not allowed. | Duplicate data is not allowed |
| Data is not in sorted order. | Data is in sorted order based on key. | Date is in sorted order based on RRN. |
| Delete is not applicable. | Delete is applicable. | Delete is applicable. |
| Access is slow. | Access is faster. | Access is faster than indexed files. |
| Key is not available. | Key is available. Key is user defined. Key is part of a record. | Key is available. Key is system defined, key is outside of a record. |