UNIT IV

Types of dependencies :

Dependencies in DBMS is a relation between two or more attributes. It has the following types in DBMS –

- Functional Dependency
- Fully-Functional Dependency
- Transitive Dependency
- Multivalued Dependency
- Partial Dependency

Let us start with Functional Dependency –

Functional Dependency

If the information stored in a table can uniquely determine another information in the same table, then it is called **Functional Dependency**. Consider it as an association between two attributes of the same relation.

If P functionally determines Q, then

P -> Q

Let us see an example –

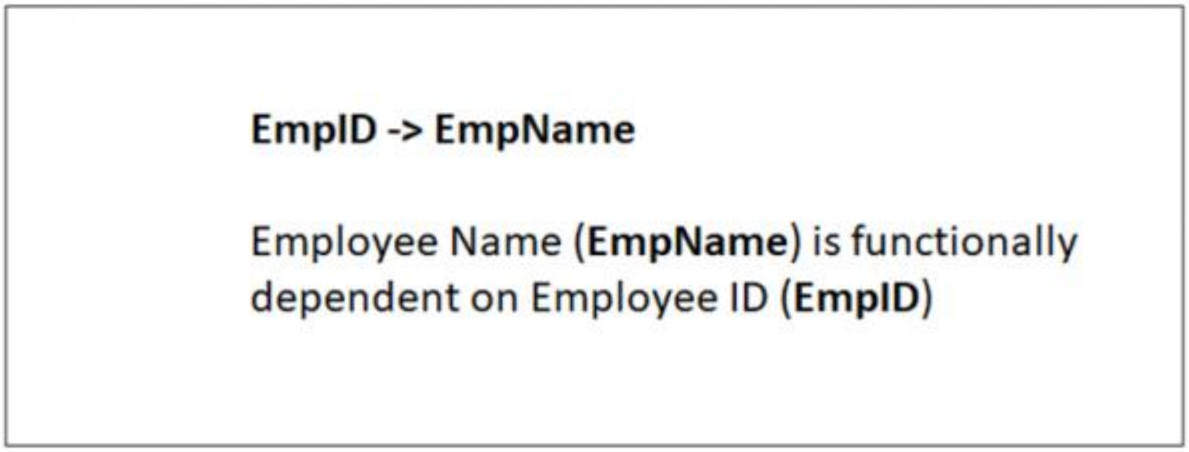
<Employee>

EmpID	**EmpName**	**EmpAge**
E01	Amit	28
E02	Rohit	31

In the above table, **EmpName** is functionally dependent on **EmpID** because **EmpName** can take only one value for the given value of **EmpID:**

EmpID -> EmpName

The same is displayed below –


EmpID -> EmpName

Employee Name (**EmpName**) is functionally dependent on Employee ID (**EmpID**)

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.

Let us see an example –

<ProjectCost>

ProjectID	**ProjectCost**
001	1000
002	5000

<EmployeeProject>

EmpID	**ProjectID**	**Days** (spent on the project)
E099	001	320
E056	002	190

The above relations states:

EmpID, ProjectID, ProjectCost -> Days

However, it is not fully functional dependent.

Whereas the subset **{EmpID, ProjectID}** can easily determine the **{Days}** spent on the project by the employee.

This summarizes and gives our fully functional dependency –
{EmpID, ProjectID} -> (Days)

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

In database management systems (DBMS), multivalued dependencies are a type of functional dependency that describe relationships between attributes in a relation (table) when certain tuples (rows) have multiple values for a particular attribute. Multivalued dependencies are used to ensure that a relation is in a specific normal form, known as Fourth Normal Form (4NF).

Let's start with a brief explanation of multivalued dependencies and then provide an example.

Multivalued Dependency (MVD): An MVD exists between two sets of attributes X and Y in a relation R if, for every value of X, there are one or more corresponding sets of values of Y that are independent of each other. In other words, if you have a tuple with a certain value in X, there can be multiple sets of values in Y that are associated with that X value, and these sets in Y are unrelated to each other.

Here's an example to illustrate multivalued dependencies:

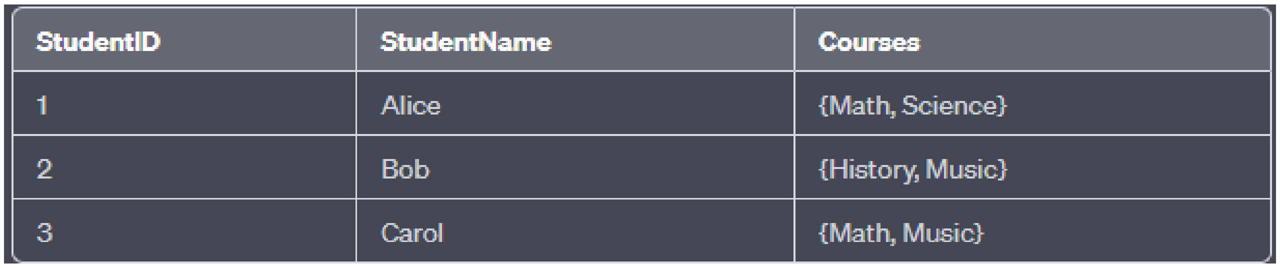
Consider a relation "StudentCourses" that stores information about students and the courses they are enrolled in. The attributes of this relation might be:

- StudentID (Student's unique identifier)

- StudentName (Student's name)

- Courses (A multivalued attribute representing the courses a student is enrolled in)

Now, let's say we have the following data in the "StudentCourses" relation:



StudentID	StudentName	Courses
1	Alice	{Math, Science}
2	Bob	{History, Music}
3	Carol	{Math, Music}

In this example, the "Courses" attribute is a multivalued attribute because it can contain multiple values for each student. Here are the multivalued dependencies:

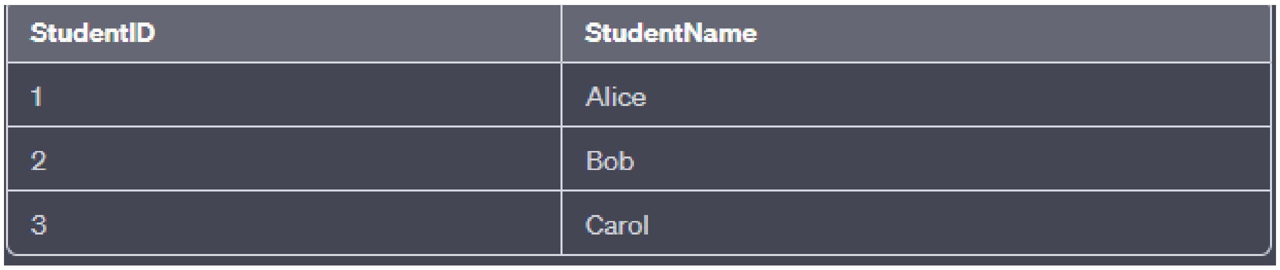
1. For StudentID 1 (Alice), the "Courses" attribute has two sets of values: {Math, Science}. These sets are independent of each other because enrolling in Math has no influence on enrolling in Science and vice versa.

2. For StudentID 2 (Bob), the "Courses" attribute has two sets of values: {History, Music}. Like in the previous case, these sets are independent of each other.

3. For StudentID 3 (Carol), the "Courses" attribute also has two sets of values: {Math, Music}, which are independent of each other.

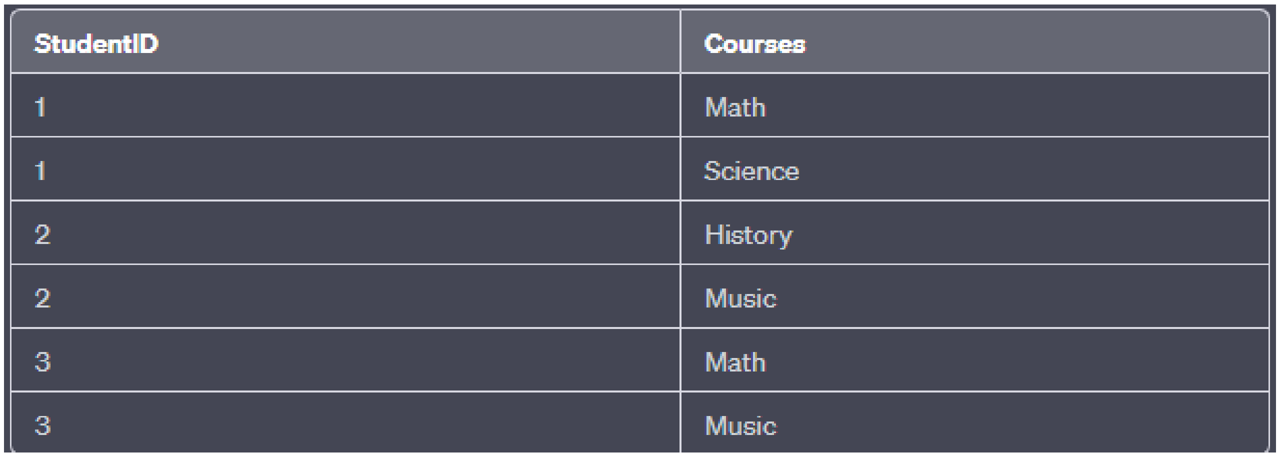
To represent this relation without violating Fourth Normal Form (4NF), you can create a separate relation for the multivalued attribute "Courses." This would result in two relations:

Student Information:



StudentID	StudentName
1	Alice
2	Bob
3	Carol

StudentCourses Information:



StudentID	Courses
1	Math
1	Science
2	History
2	Music
3	Math
3	Music

By splitting the multivalued attribute into a separate relation, you eliminate multivalued dependencies and ensure that the database is in Fourth Normal Form (4NF).

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.

Normalization :

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is 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 undesirable characteristics like Insertion, Update, and Deletion Anomalies.

- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

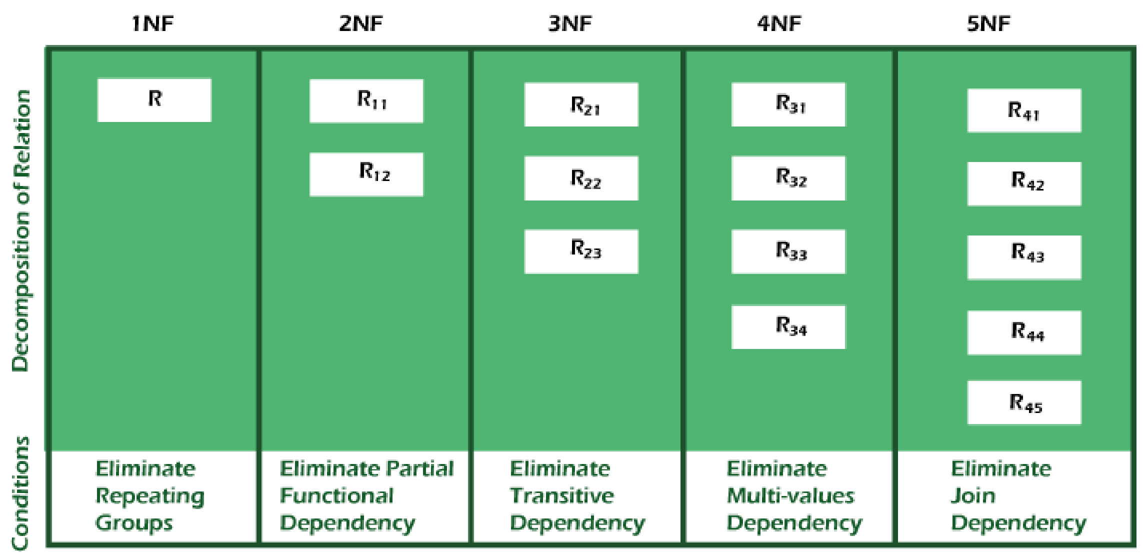
Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:


1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R11
R12
R21
R22
R23
R31
R32
R33
R34
R41
R42
R43
R44
R45
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

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.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.

- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

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

Sixth Normal Form (6NF):

6NF deals with additional decompositions based on join dependencies and information preservation.

Achieving 6NF typically involves complex scenarios and is less common in practice. It deals with cases where even 5NF doesn't eliminate all redundancy and dependency issues.

Organization of Database System:

Introduction of file, file types :

Introduction to File Types: Serial and Sequential Files

In the realm of computer science and data storage, files can be categorized into various types based on their structure and how data is organized within them. Two common types of files are "serial files" and "sequential files." Let's explore each of these file types:

1. **Serial Files**:

- **Definition**: Serial files, also known as flat files, are a type of data file where records are stored one after another with no specific order or structure. Each record is typically separated by a delimiter, such as a newline character or a comma.

- **Characteristics**:

  - Records are stored sequentially, one after another.

  - No specific order is imposed on the records, and they are often appended as new data arrives.

  - Commonly used for simple data storage and transfer, such as logs or basic data exports.

  - Retrieving specific records may require reading through the entire file.

2. **Sequential Files**:

- **Definition**: Sequential files are a type of file where data is organized in a specific order, often based on a primary key or another criterion. Records are stored one after another, but they follow a predefined sequence or sorting order.

- **Characteristics**:

  - Records are arranged in a specific order, such as alphabetical, numerical, or chronological.

  - Well-suited for applications where data needs to be processed in a systematic manner, such as in accounting or inventory management.

- Efficient for searching and retrieving records based on the predefined order.
- Adding or modifying records may require reorganizing the entire file to maintain the sequence.

To better understand the difference between serial and sequential files, consider the following examples:

- **Serial File Example**: A log file that stores timestamped log entries generated by a web server. Each log entry is appended to the file as new events occur, and there is no specific order imposed on the entries other than their chronological order of occurrence.

- **Sequential File Example**: A customer database file where customer records are stored in alphabetical order based on their last names. This sequential arrangement allows for efficient searching and retrieval of customer information based on last names.

In summary, serial files are characterized by the absence of a specific order, making them suitable for simple data storage and transfer, while sequential files maintain a predefined order, making them valuable for applications that require systematic organization and retrieval of data based on a specific criterion. The choice between these file types depends on the specific needs of the application and how data will be accessed and processed.

organization of file- heap file organization :

Heap File Organization:

Heap file organization is a method of storing and managing data records in a database or file system. It is characterized by its simplicity and lack of any specific order or structure imposed on the data records within the file. In a heap file, records are inserted into the file as they arrive, and they are not sorted or organized in any particular way. This results in a more straightforward data storage approach, but it can also make data retrieval less efficient compared to other file organization methods.

Key characteristics of heap file organization:

1. **No Specific Order**: In a heap file, data records are stored in the order they are added, without any regard for a specific sequence or sorting criteria. This means that the records can be in any order within the file.

2. **Insertion Flexibility**: Heap files are well-suited for scenarios where new records are frequently added to the file. Since there is no requirement to maintain a particular order, inserting new records is a straightforward process.

3. **No Indexing**: Unlike some other file organization methods, such as indexed or sequential files, heap files do not have built-in indexes to speed up record retrieval. To find a specific record, you may need to scan through the entire file, which can be inefficient for large files.

4. **Random Access**: While heap files lack a specific order, they do allow for random access to records. This means you can read or modify any record in the file without having to navigate through the entire file sequentially.

5. **Simple Implementation**: Heap file organization is relatively simple to implement, making it a good choice for scenarios where data organization requirements are minimal, and performance considerations are less critical.

Use Cases of Heap File Organization:

Heap file organization is suitable for certain use cases, including:

1. **Logging**: When recording logs or events, a heap file can be used to store log entries as they occur, without any specific order.

2. **Temporary Data**: In situations where data is temporary and does not need to be stored in a structured manner, heap files can be used for temporary storage.

3. **Data Loading**: Heap files are often used when bulk loading data into a database before applying sorting and indexing processes.

4. **Data Warehousing**: In data warehousing scenarios where raw data is initially collected before being transformed and organized, heap files can be employed for storing the raw data.

However, it's important to note that heap file organization is not well-suited for applications that require efficient searching, sorting, or retrieval of records based on specific criteria. In such cases, other file organization methods like sequential or indexed files may be more appropriate to optimize data access performance.

index sequential file, random access file (direct access file) :

Index Sequential File, Random Access File (Direct Access File):

Index Sequential File and Random Access File, also known as Direct Access File, are two different methods of organizing and accessing data within computer files. These methods offer distinct advantages and are suitable for specific use cases:

1. **Index Sequential File**:

- **Definition**: An index sequential file is a file organization method that combines aspects of both sequential and indexed file structures. It's designed to provide efficient access to records based on a key field (often called the index field) while maintaining the physical order of records.

- **Characteristics**:
  - **Sequential Ordering**: Records are physically stored in sequential order on the storage medium, which can be beneficial for efficient storage and retrieval.
  - **Index**: An index structure is maintained alongside the file, containing key values and pointers to corresponding records. This index allows for rapid access to records based on the indexed field.
  - **Key Field**: Records are typically indexed based on one or more key fields, such as an employee ID or a product code.
  - **Efficient Searching**: The index enables efficient searching for specific records based on the key field. This is particularly useful for retrieval and queries.
  - **Records in Sequence**: Records are stored in sequence based on the physical storage order, which can be advantageous for operations like range scans.

- **Use Cases**: Index sequential files are often used in databases and file systems where data needs to be efficiently accessed based on specific key values, while also preserving some level of physical order.

2. **Random Access File (Direct Access File)**:

- **Definition**: A random access file, also known as a direct access file, is a file organization method that allows for direct access to any record within the file without the need to traverse preceding records. It provides efficient access to records using record numbers or byte offsets.

- **Characteristics**:
  - **Random Access**: Unlike sequential files, random access files do not require reading records sequentially. Instead, they allow direct access to a particular record using a record number or byte offset.

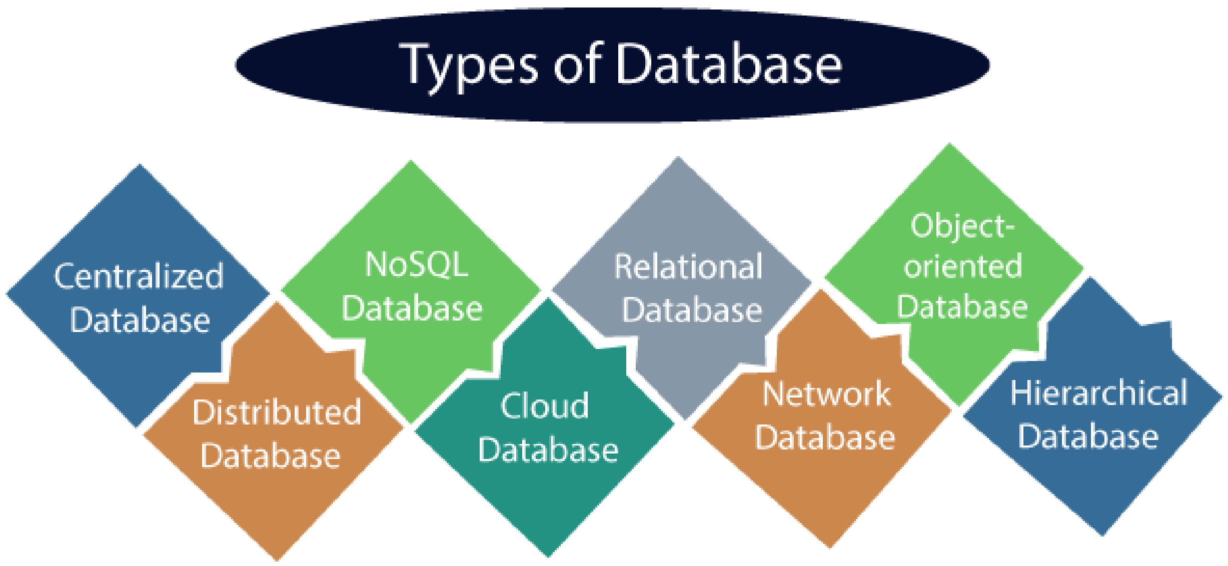
- **Efficient Record Retrieval**: Random access is highly efficient for retrieving specific records without scanning through the entire file.
- **Record Addressing**: Records can be addressed directly, which is particularly useful in situations where rapid access to specific records is critical.
- **No Natural Ordering**: Unlike sequential files, random access files do not maintain a natural order of records.

- **Use Cases**: Random access files are commonly used in applications where quick access to specific records is essential, such as databases, file systems, and indexed files.

In summary, index sequential files are a hybrid approach that combines sequential storage with an index structure for efficient access based on key fields. Random access files, on the other hand, prioritize direct access to records using record numbers or byte offsets, making them suitable for applications that require fast and direct access to specific data without the need for sequential scanning. The choice between these file organization methods depends on the specific requirements of the application and the nature of data access patterns.

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.

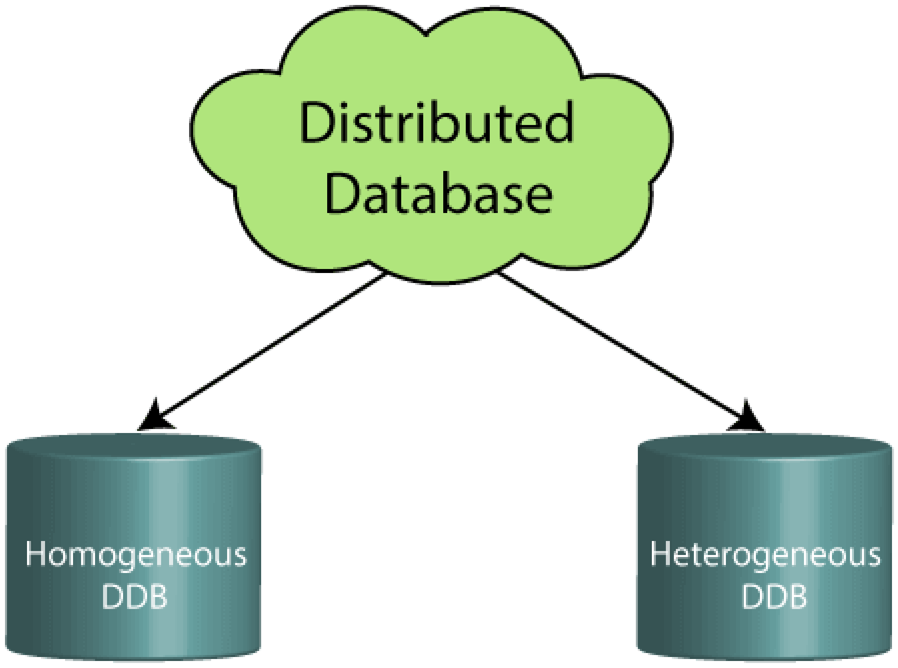
Disadvantages of Centralized Database

- 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:

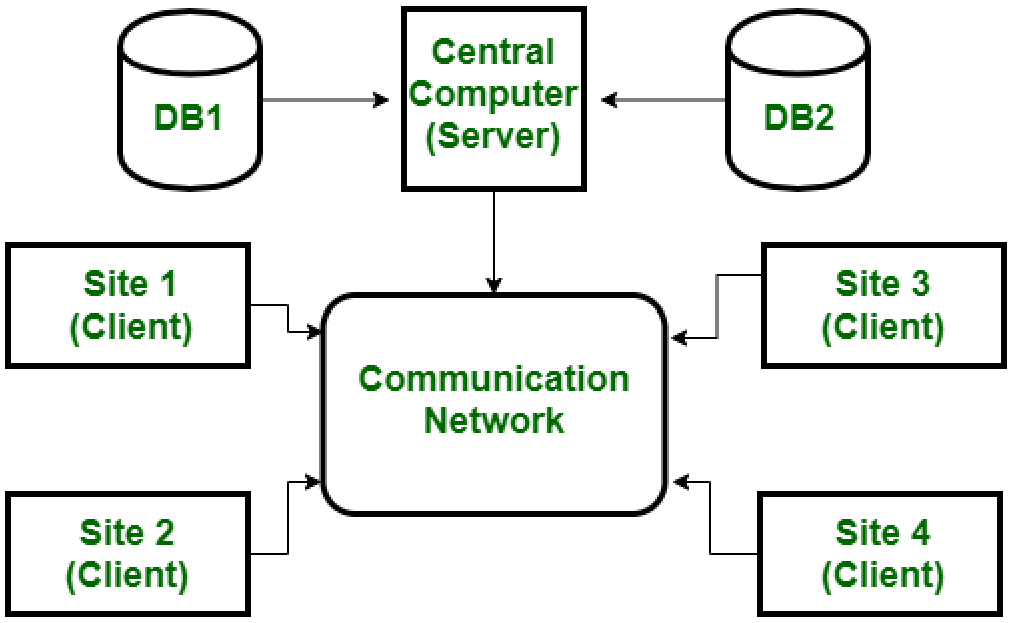


- **Homogeneous DDB:** Those database systems which execute on the same operating system and use the same application process and carry the same hardware devices.
- **Heterogeneous DDB:** Those database systems which execute on different operating systems under different application procedures, and carries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

Client Server Database System :


DB1
Central Computer (Server)
DB2
Site 1 (Client)
Site 3 (Client)
Communication Network
Site 2 (Client)
Site 4 (Client)