Relational database design: Functional Dependencies -Normalization and its normal forms-Denormalization- Data Storage: RAID – Tertiary Storage – File organization – Organization of records in files. Query Processing - Query optimization.

Functional Dependency

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

1. X → Y

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

For example:

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

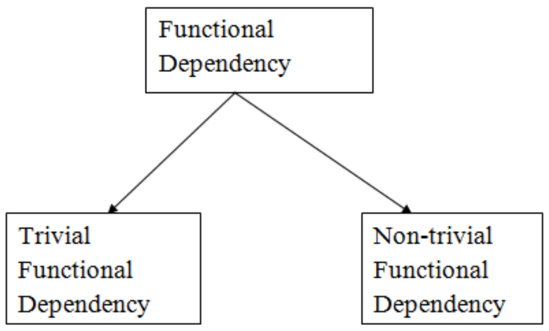
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

1. Trivial functional dependency

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

Example:

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

2. Non-trivial functional dependency

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

Example:

1. ID → Name,
2. Name → DOB

Inference Rule (IR):

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

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR1)

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

1. If X ⊇ Y then X → Y

Example:

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

2. Augmentation Rule (IR2)

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

1. If X → Y then XZ → YZ

Example:

1. For R(ABCD), **if** A → B then AC → BC

3. Transitive Rule (IR3)

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

1. If X → Y and Y → Z then X → Z

4. Union Rule (IR4)

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

1. If X → Y and X → Z then X → YZ

Proof:

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

4.XY→YZ(using IR_2 on 2 by augmentation with Y)

5. X → YZ (using IR_3 on 3 and 4)

5. Decomposition Rule (IR5)

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

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

1. If X → YZ then X → Y and X → Z

   Proof:

   1.X→YZ(given)
   2.YZ→Y(usingIR$_1$ Rule)
   3. X → Y (using IR$_3$ on 1 and 2)

6. Pseudo transitive Rule (IR6)

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

1. If X → Y and YZ → W then XZ → W

   Proof:

   1.X→Y(given)
   2.WY→Z(given)
   3.WX→WY(using IR$_2$ on 1 by augmenting with W)
   4. WX → Z (using IR$_3$ on 3 and 2)

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.

Play Video

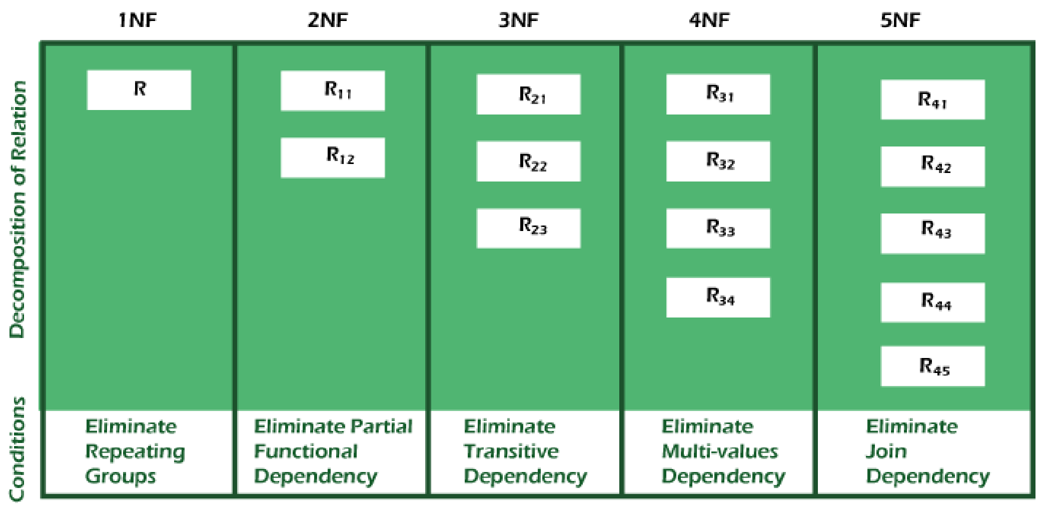
Data modification anomalies can be categorized into three types:

- **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
364	UK

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

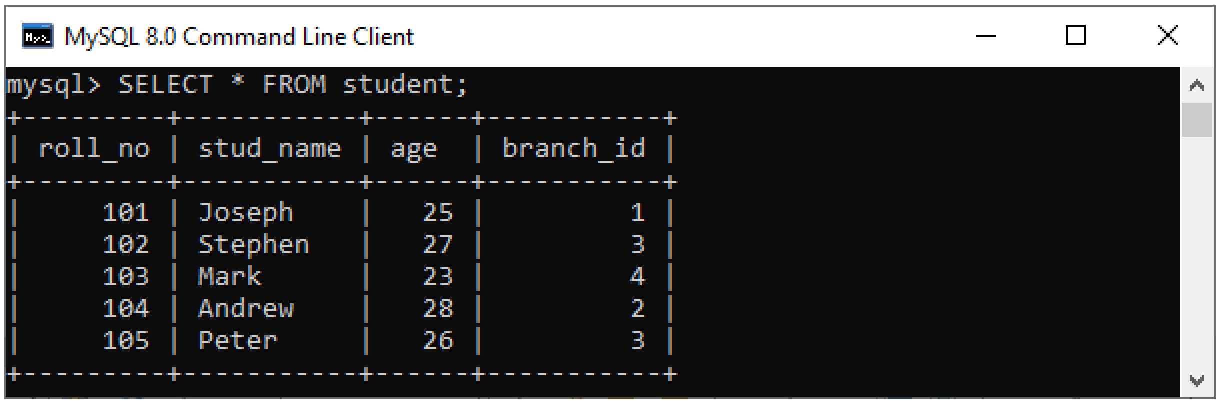
SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Denormalization in Databases

When we normalize tables, we break them into multiple smaller tables. So when we want to retrieve data from multiple tables, we need to perform some kind of join operation on them. In that case, we use the denormalization technique that eliminates the drawback of normalization.

Denormalization is a technique used by database administrators to optimize the efficiency of their database infrastructure. This method allows us to add redundant data into a normalized database to alleviate issues with database queries that merge data from several tables into a single table. The denormalization concept is based on the definition of normalization that is defined as arranging a database into tables correctly for a particular purpose.

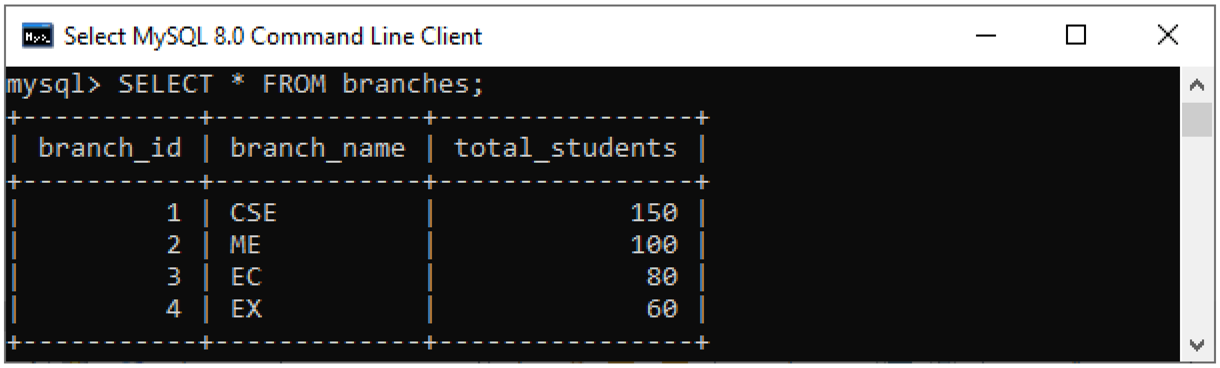
For Example, We have two table students and branch after performing normalization. The student table has the attributes roll_no, stud-name, age, and branch_id.



```
mysql> SELECT * FROM student;
+---------+-----------+------+-----------+
| roll_no | stud_name | age  | branch_id |
+---------+-----------+------+-----------+
|     101 | Joseph    |   25 |         1 |
|     102 | Stephen   |   27 |         3 |
|     103 | Mark      |   23 |         4 |
|     104 | Andrew    |   28 |         2 |
|     105 | Peter     |   26 |         3 |
+---------+-----------+------+-----------+
```

Additionally, the branch table is related to the student table with branch_id as the student table's foreign key.

Play Video



```
mysql> SELECT * FROM branches;
+-----------+-------------+----------------+
| branch_id | branch_name | total_students |
+-----------+-------------+----------------+
|         1 | CSE         |            150 |
|         2 | ME          |            100 |
|         3 | EC          |             80 |
|         4 | EX          |             60 |
+-----------+-------------+----------------+
```

A JOIN operation between these two tables is needed when we need to retrieve all student names as well as the branch name. Suppose we want to change the student name only, then it is great if the table is small. The issue here is that if the tables are big, joins on tables can take an excessively long time.

In this case, we'll update the database with denormalization, redundancy, and extra effort to maximize the efficiency benefits of fewer joins. Therefore, we can add the branch name's data from the Branch table to the student table and optimizing the database.

Pros of Denormalization

The following are the advantages of denormalization:

1. **Enhance Query Performance**

Fetching queries in a normalized database generally requires joining a large number of tables, but we already know that the more joins, the slower the query. To overcome this, we can add redundancy to a database by copying values between parent and child tables, minimizing the number of joins needed for a query.

2. Make database more convenient to manage

A normalized database is not required calculated values for applications. Calculating these values on-the-fly will take a longer time, slowing down the execution of the query. Thus, in denormalization, fetching queries can be simpler because we need to look at fewer tables.

3. Facilitate and accelerate reporting

Suppose you need certain statistics very frequently. It requires a long time to create them from live data and slows down the entire system. Suppose you want to monitor client revenues over a certain year for any or all clients. Generating such reports from live data will require "searching" throughout the entire database, significantly slowing it down.

Cons of Denormalization

The following are the disadvantages of denormalization:

- It takes large storage due to data redundancy.
- It makes it expensive to updates and inserts data in a table.
- It makes update and inserts code harder to write.
- Since data can be modified in several ways, it makes data inconsistent. Hence, we'll need to update every piece of duplicate data. It's also used to measure values and produce reports. We can do this by using triggers, transactions, and/or procedures for all operations that must be performed together.

How is denormalization different from normalization?

The denormalization is different from normalization in the following manner:

- Denormalization is a technique used to merge data from multiple tables into a single table that can be queried quickly. Normalization, on the other hand, is used to delete redundant data from a database and replace it with non-redundant and reliable data.

- Denormalization is used when joins are costly, and queries are run regularly on the tables. Normalization, on the other hand, is typically used when a large number of insert/update/delete operations are performed, and joins between those tables are not expensive.

RAID

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

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

RAID technology

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

These levels contain the following characteristics:

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

Standard RAID levels

RAID 0

- RAID level 0 provides data stripping, i.e., a data can place across multiple disks. It is based on stripping that means if one disk fails then all data in the array is lost.
- This level doesn't provide fault tolerance but increases the system performance.

Example:

Disk 0	Disk 1	Disk 2	Disk 3
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35

In this figure, block 0, 1, 2, 3 form a stripe.

In this level, instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one.

Disk 0	Disk 1	Disk 2	Disk 3
20	22	24	26
21	23	25	27
28	30	32	34
29	31	33	35

In this above figure, there is no duplication of data. Hence, a block once lost cannot be recovered.

Pros of RAID 0:

- In this level, throughput is increased because multiple data requests probably not on the same disk.
- This level full utilizes the disk space and provides high performance.
- It requires minimum 2 drives.

Cons of RAID 0:

- It doesn't contain any error detection mechanism.
- The RAID 0 is not a true RAID because it is not fault-tolerance.
- In this level, failure of either disk results in complete data loss in respective array.

RAID 1

This level is called mirroring of data as it copies the data from drive 1 to drive 2. It provides 100% redundancy in case of a failure.

Example:

Disk 0	**Disk 1**	**Disk 2**	**Disk 3**
A	A	B	B
C	C	D	D
E	E	F	F
G	G	H	H

Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.

Pros of RAID 1:

- The main advantage of RAID 1 is fault tolerance. In this level, if one disk fails, then the other automatically takes over.
- In this level, the array will function even if any one of the drives fails.

Cons of RAID 1:

- In this level, one extra drive is required per drive for mirroring, so the expense is higher.

RAID 2

- RAID 2 consists of bit-level striping using hamming code parity. In this level, each data bit in a word is recorded on a separate disk and ECC code of data words is stored on different set disks.
- Due to its high cost and complex structure, this level is not commercially used. This same performance can be achieved by RAID 3 at a lower cost.

Pros of RAID 2:

- This level uses one designated drive to store parity.
- It uses the hamming code for error detection.

Cons of RAID 2:

- It requires an additional drive for error detection.

RAID 3

- RAID 3 consists of byte-level striping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
- In case of drive failure, the parity drive is accessed, and data is reconstructed from the remaining devices. Once the failed drive is replaced, the missing data can be restored on the new drive.
- In this level, data can be transferred in bulk. Thus high-speed data transmission is possible.

Disk 0	Disk 1	Disk 2	Disk 3
A	B	C	P(A, B, C)
D	E	F	P(D, E, F)
G	H	I	P(G, H, I)
J	K	L	P(J, K, L)

Pros of RAID 3:

- In this level, data is regenerated using parity drive.
- It contains high data transfer rates.
- In this level, data is accessed in parallel.

Cons of RAID 3:

- It required an additional drive for parity.
- It gives a slow performance for operating on small sized files.

RAID 4

- RAID 4 consists of block-level stripping with a parity disk. Instead of duplicating data, the RAID 4 adopts a parity-based approach.
- This level allows recovery of at most 1 disk failure due to the way parity works. In this level, if more than one disk fails, then there is no way to recover the data.
- Level 3 and level 4 both are required at least three disks to implement RAID.

Disk 0	Disk 1	Disk 2	Disk 3
A	B	C	P0
D	E	F	P1
G	H	I	P2
J	K	L	P3

In this figure, we can observe one disk dedicated to parity.

In this level, parity can be calculated using an XOR function. If the data bits are 0,0,0,1 then the parity bits is XOR(0,1,0,0) = 1. If the parity bits are 0,0,1,1 then the parity bit is XOR(0,0,1,1)= 0. That means, even number of one results in parity 0 and an odd number of one results in parity 1.

C1	C2	C3	C4	Parity
0	1	0	0	1
0	0	1	1	0

Suppose that in the above figure, C2 is lost due to some disk failure. Then using the values of all the other columns and the parity bit, we can recompute the data bit stored in C2. This level allows us to recover lost data.

RAID 5

- RAID 5 is a slight modification of the RAID 4 system. The only difference is that in RAID 5, the parity rotates among the drives.
- It consists of block-level striping with DISTRIBUTED parity.

- Same as RAID 4, this level allows recovery of at most 1 disk failure. If more than one disk fails, then there is no way for data recovery.

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

This figure shows that how parity bit rotates.

This level was introduced to make the random write performance better.

Pros of RAID 5:

- This level is cost effective and provides high performance.
- In this level, parity is distributed across the disks in an array.
- It is used to make the random write performance better.

Cons of RAID 5:

- In this level, disk failure recovery takes longer time as parity has to be calculated from all available drives.
- This level cannot survive in concurrent drive failure.

RAID 6

- This level is an extension of RAID 5. It contains block-level stripping with 2 parity bits.
- In RAID 6, you can survive 2 concurrent disk failures. Suppose you are using RAID 5, and RAID 1. When your disks fail, you need to replace the failed disk because if simultaneously another disk fails then you won't be able to recover any of the

data, so in this case RAID 6 plays its part where you can survive two concurrent disk failures before you run out of options.

Disk 1	**Disk 2**	**Disk 3**	**Disk 4**
A0	B0	Q0	P0
A1	Q1	P1	D1
Q2	P2	C2	D2
P3	B3	C3	Q3

Pros of RAID 6:

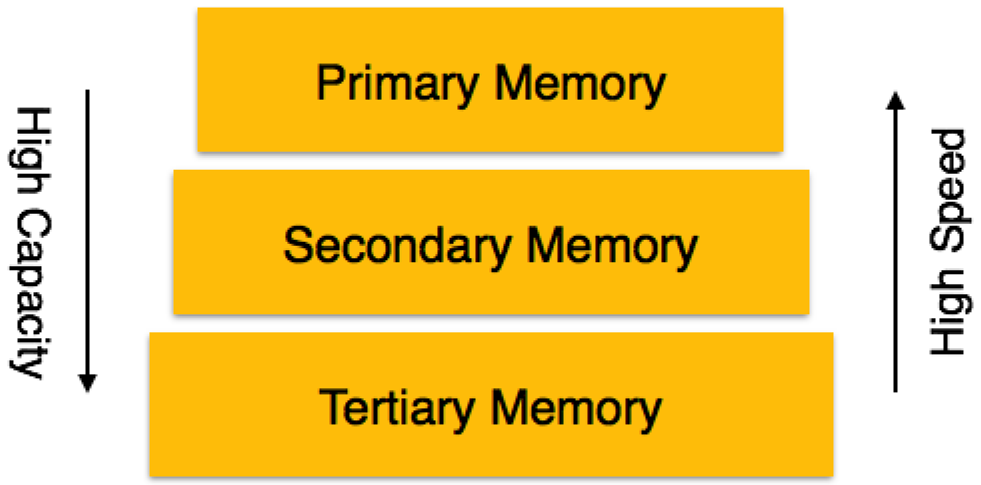
- This level performs RAID 0 to strip data and RAID 1 to mirror. In this level, stripping is performed before mirroring.
- In this level, drives required should be multiple of 2.

Cons of RAID 6:

- It is not utilized 100% disk capability as half is used for mirroring.
- It contains very limited scalability.

DBMS - Storage System

Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types −



- Primary Storage − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.

- Secondary Storage − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

- Tertiary Storage − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

Memory Hierarchy

A computer system has a well-defined hierarchy of memory. A CPU has direct access to it main memory as well as its inbuilt registers. The access time of the main memory is obviously less than the CPU speed. To minimize this speed mismatch, cache memory is introduced. Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.

The memory with the fastest access is the costliest one. Larger storage devices offer slow speed and they are less expensive, however they can store huge volumes of data as compared to CPU registers or cache memory.

Magnetic Disks

Hard disk drives are the most common secondary storage devices in present computer systems. These are called magnetic disks because they use the concept of magnetization to store information. Hard disks consist of metal disks coated with magnetizable material. These disks are placed vertically on a spindle. A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it. A magnetized spot can be recognized as 0 (zero) or 1 (one).

Hard disks are formatted in a well-defined order to store data efficiently. A hard disk plate has many concentric circles on it, called tracks. Every track is further divided into sectors. A sector on a hard disk typically stores 512 bytes of data.

File Organization

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

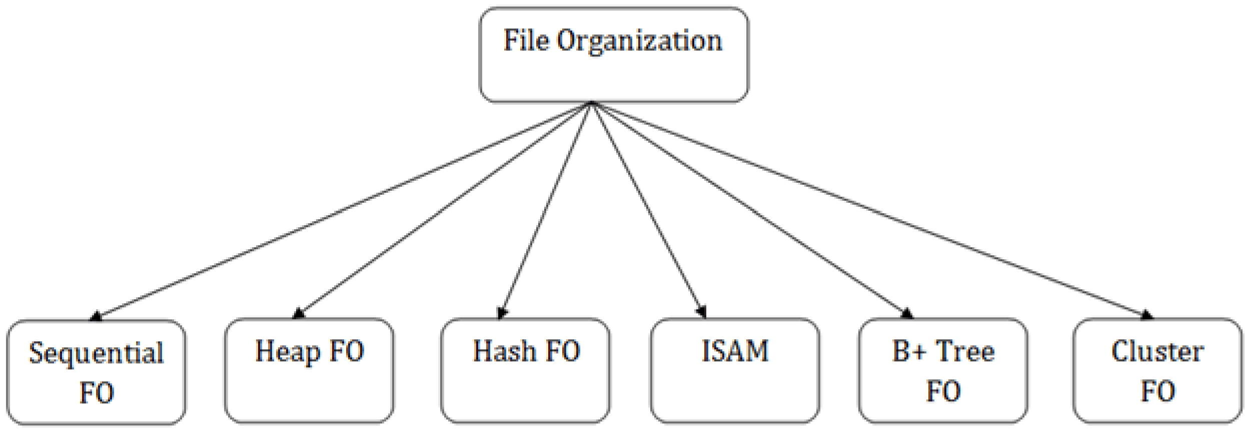
Objective of file organization

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

Types of file organization:

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

Types of file organization are as follows:



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

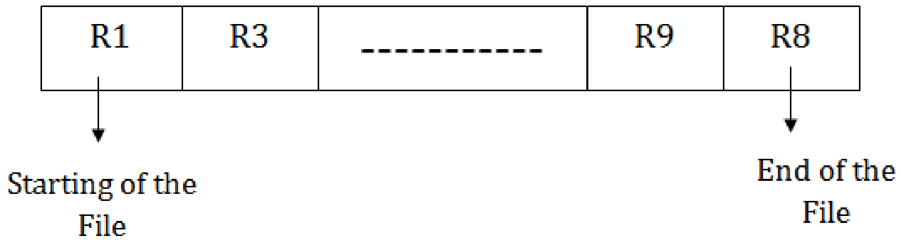
Sequential File Organization

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

1. Pile File Method:

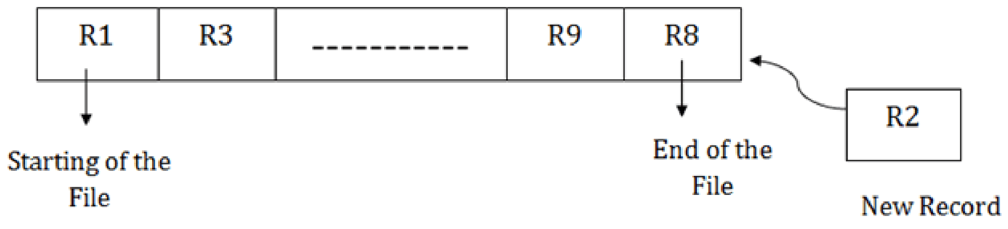
- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.

- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



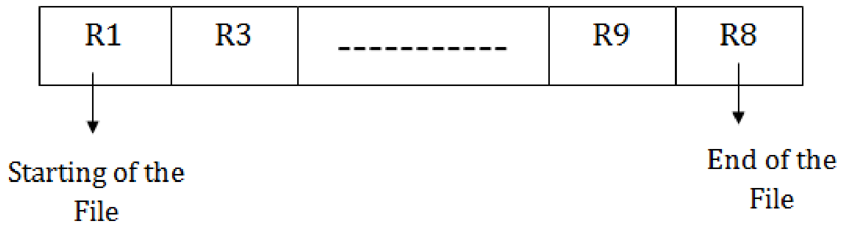
Insertion of the new record:

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



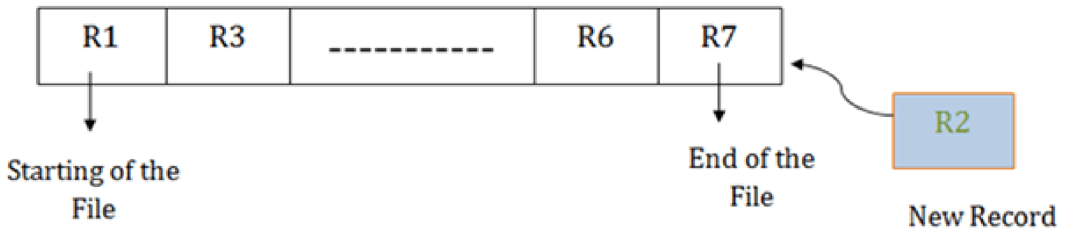
2. Sorted File Method:

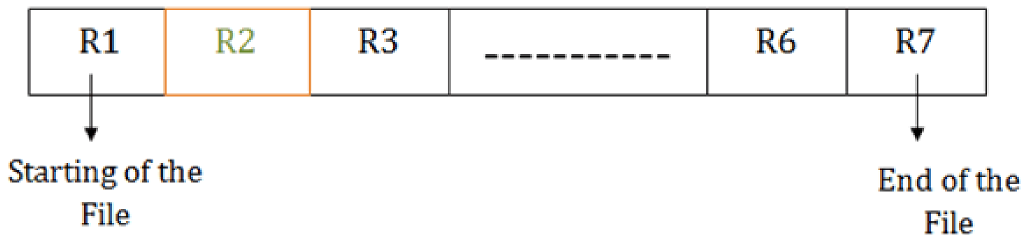
- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and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



Insertion of the new record:

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





Pros of sequential file organization

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

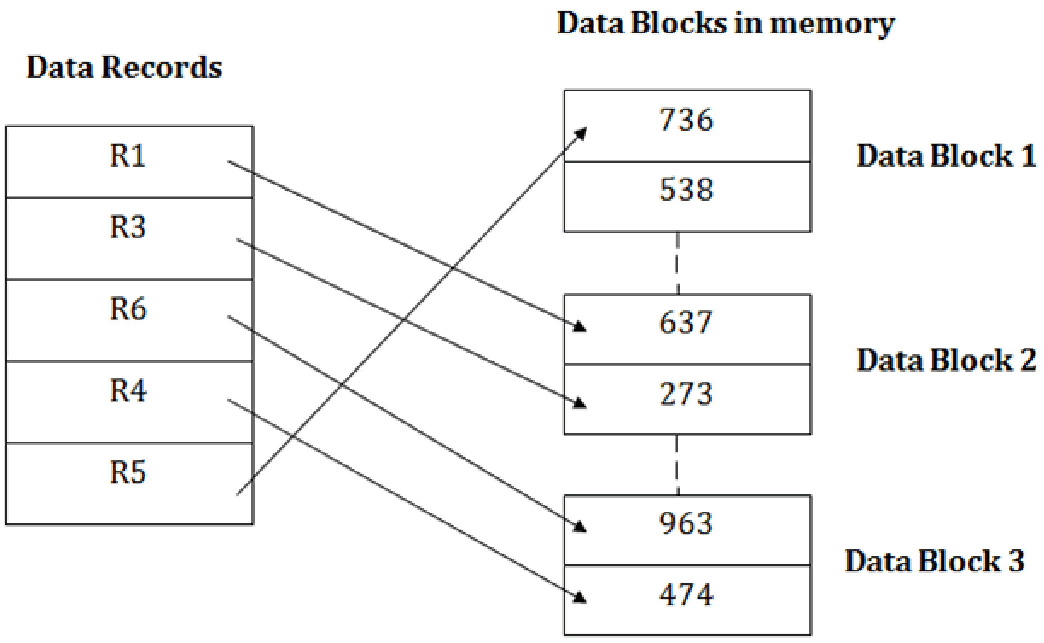
Cons of sequential file organization

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

Heap file organization

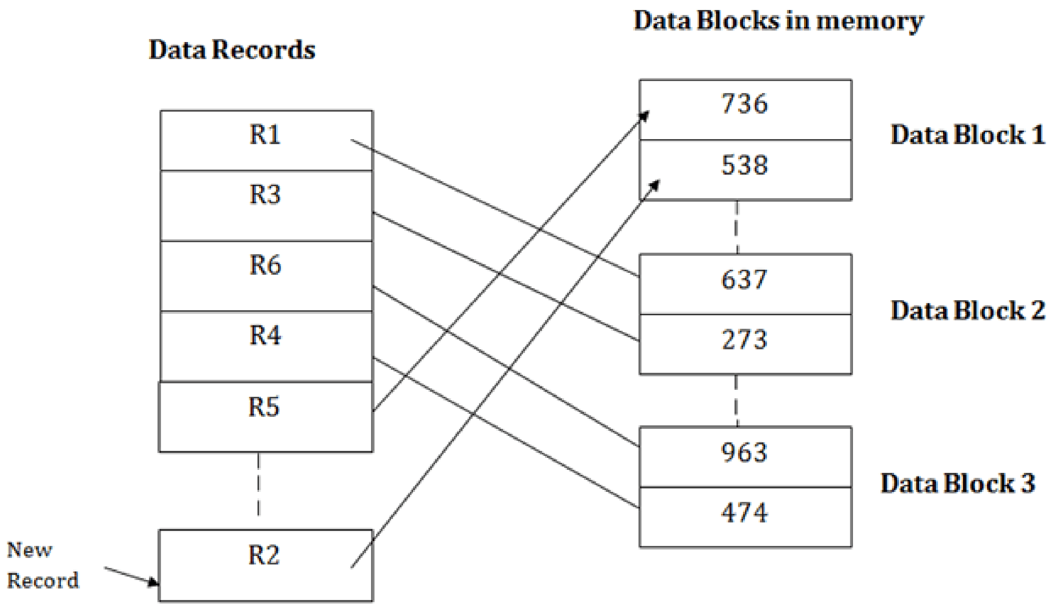
- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in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

- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

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



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

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

Pros of Heap file organization

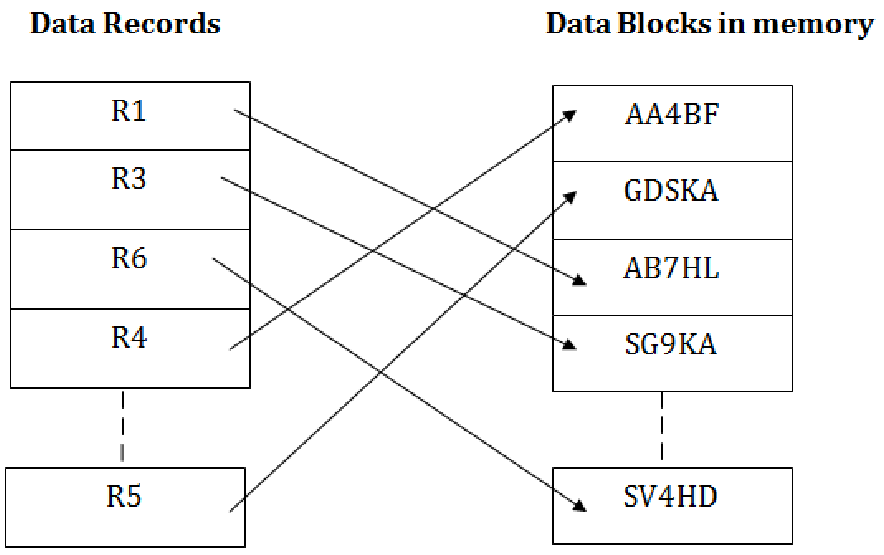
- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

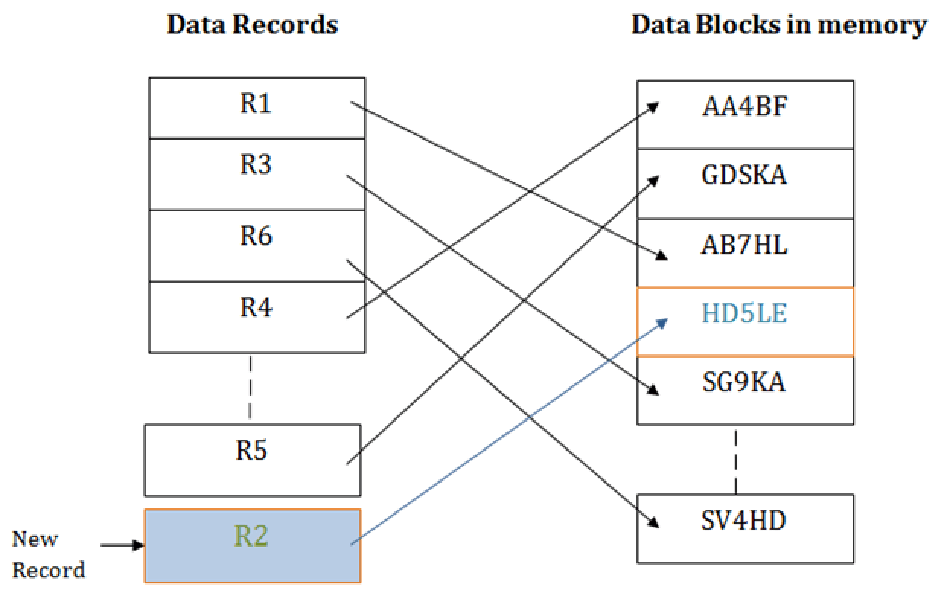
- This method is inefficient for the large database because it takes time to search or modify the record.
- This method is inefficient for large databases.

Hash File Organization

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

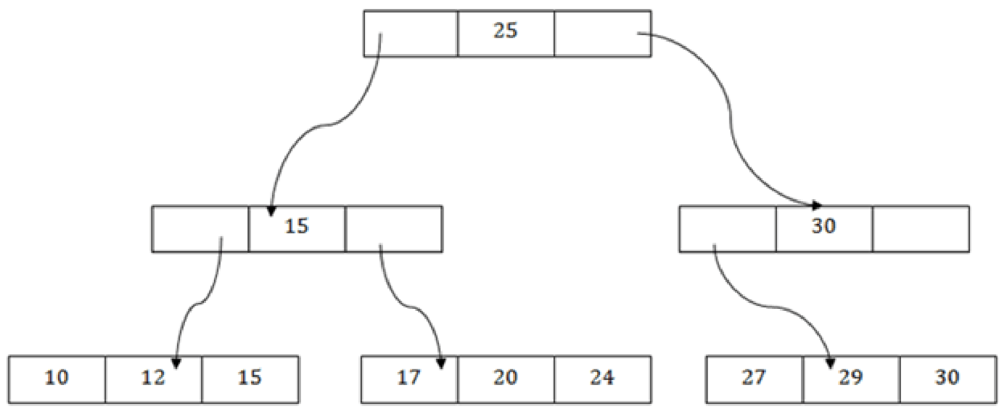


-
- When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.
- In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.

- 

B+ File Organization

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



The above B+ tree shows that:

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

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

Pros of B+ tree file organization

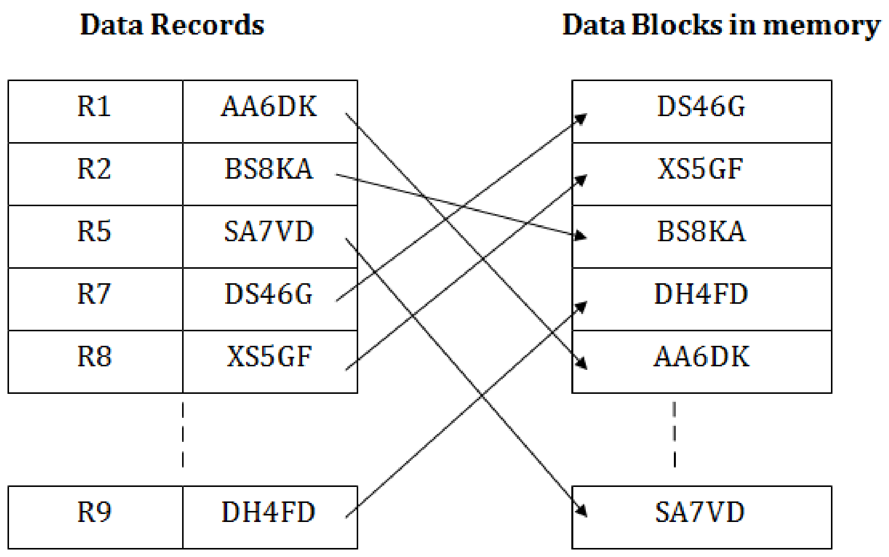
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and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

Cons of B+ tree file organization

- This method is inefficient for the static method.

Indexed sequential access method (ISAM)

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



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

Pros of ISAM:

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

Cons of ISAM

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

Cluster file organization

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

EMPLOYEE

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

DEPARTMENT

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

Cluster Key

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

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

Types of Cluster file organization:

Cluster file organization is of two types:

1. Indexed Clusters:

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

2. Hash Clusters:

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

Pros of Cluster file organization

- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.

- It provides the efficient result when there is a 1:M mapping between the tables.

Cons of Cluster file organization

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.
- This method is not suitable for a table with a 1:1 condition.

Indexing in DBMS

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

Index structure:

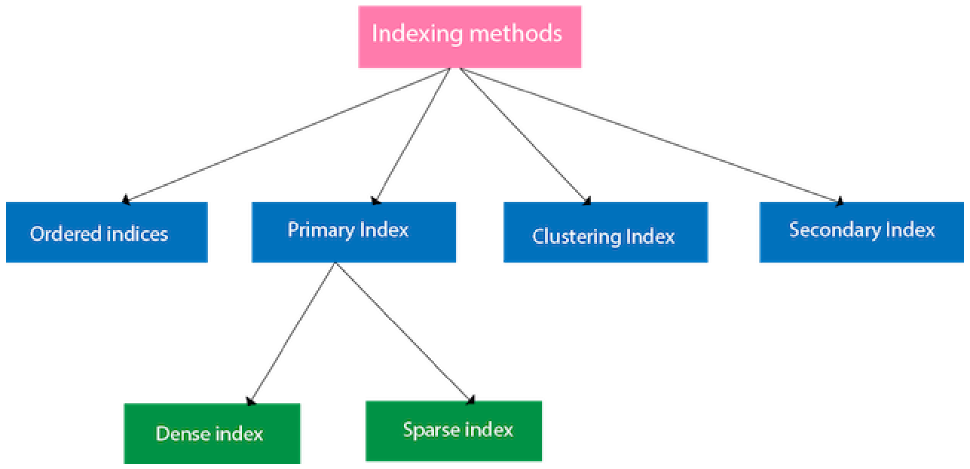
Indexes can be created using some database columns.

Search key	Data Reference

Fig: Structure of Index

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

Indexing Methods



Ordered indices

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

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

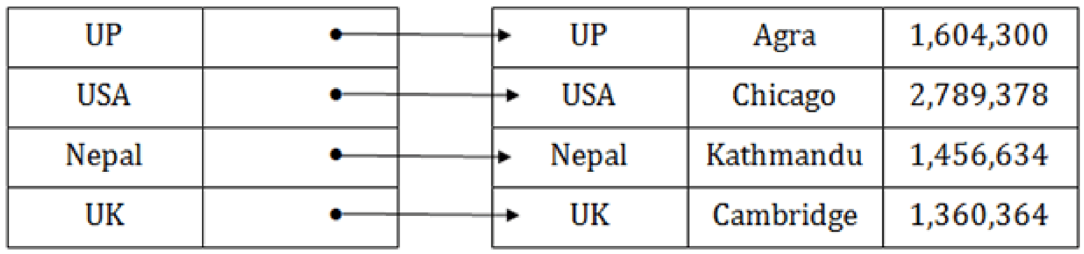
- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which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

Primary Index

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

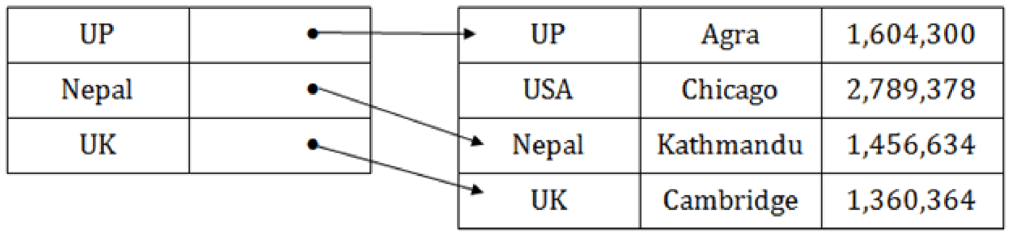
Dense index

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


UP
USA
Nepal
UK
UP | Agra | 1,604,300
USA | Chicago | 2,789,378
Nepal | Kathmandu | 1,456,634
UK | Cambridge | 1,360,364

Sparse index

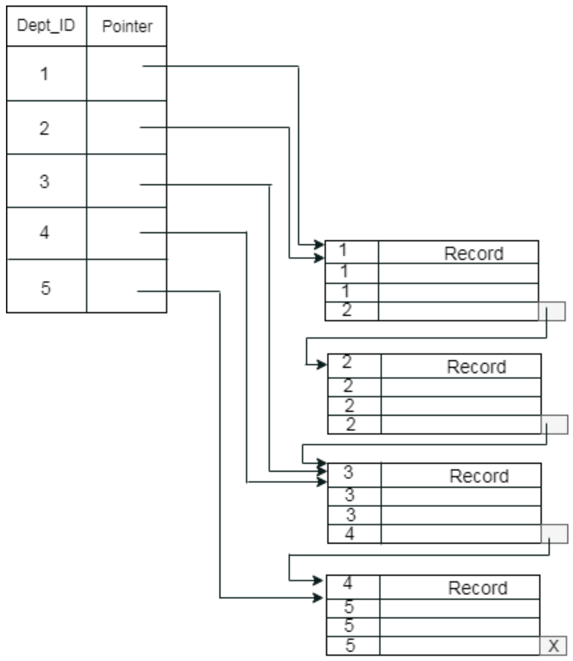
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.


UP
Nepal
UK
UP | Agra | 1,604,300
USA | Chicago | 2,789,378
Nepal | Kathmandu | 1,456,634
UK | Cambridge | 1,360,364

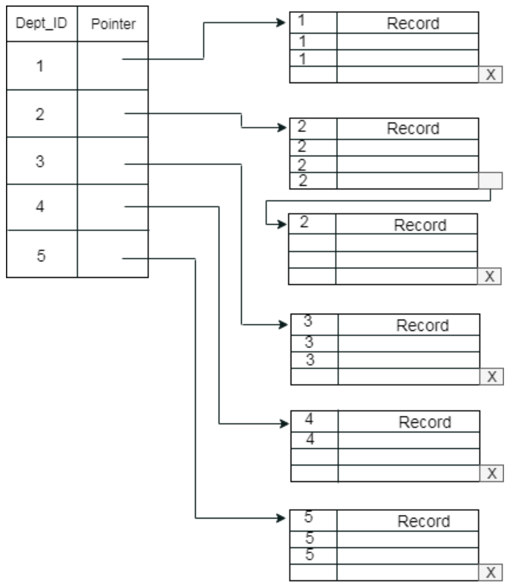
Clustering Index

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

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



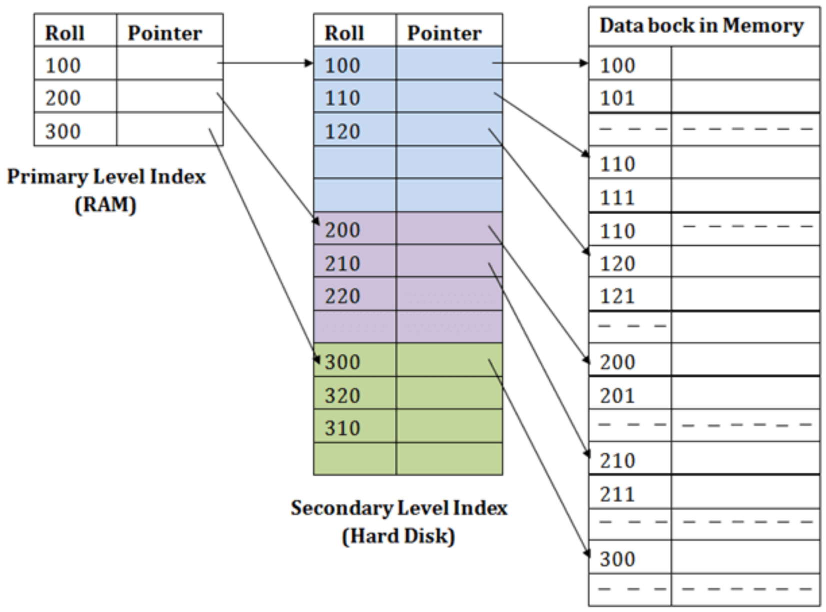
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index

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

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



For example:

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