Indexing in DBMS

Overview

Indexing in DBMS is a technique that uses data structures to optimize the searching time of a database query. It helps in faster query results and quick data retrieval from the database. Indexing makes database performance better. It also consumes lesser space in the main memory.

Scope

The article contains topics such as Indexing and its types, Index Table, Attributes of Indexing, and Advantages of Indexing.

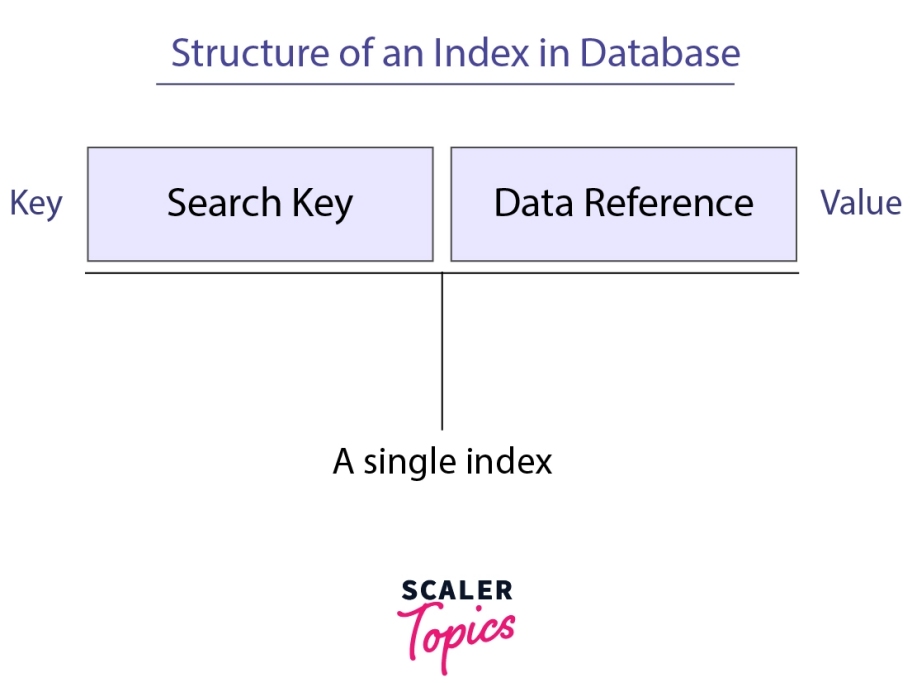
Each of the topics is explained clearly with diagrams and examples wherever necessary.

What is Indexing in DBMS?

Indexing is used to quickly retrieve particular data from the database. Formally we can define Indexing as a technique that uses data structures to optimize the searching time of a database query. Indexing reduces the number of disks required to access a particular data by internally creating an index table.

Indexing is achieved by creating **Index-table** or **Index**.

Index usually consists of two columns which are a key-value pair. The two columns of the index table(i.e., the key-value pair) contain copies of selected columns of the tabular data of the database.



Here, **Search Key** contains the copy of the *Primary Key* or the *Candidate Key* of the database table. Generally, we store the selected Primary or Candidate keys in a sorted manner so that we can reduce the overall query time or search time(from linear to binary).

Data Reference contains a set of pointers that holds the address of the disk block. The pointed disk block contains the actual data referred to by the Search Key. Data Reference is also called **Block Pointer** because it uses block-based addressing.

Indexing Attributes

Let's discuss the various indexing attributes:

Standard (B-tree) and Bitmap

B-tree-indexing is one of the most popular and commonly used indexing techniques. B-tree is a type of tree data structure that contains 2 things namely: **Index Key** and its corresponding **disk address**. Index Key refers to a certain disk address and that disk further contains rows or tuples of data.

On the other hand, Bitmap indexing uses strings to store the address of the tuples or rows. A bitmap is a mapping from one system to the other such as integers to bits.

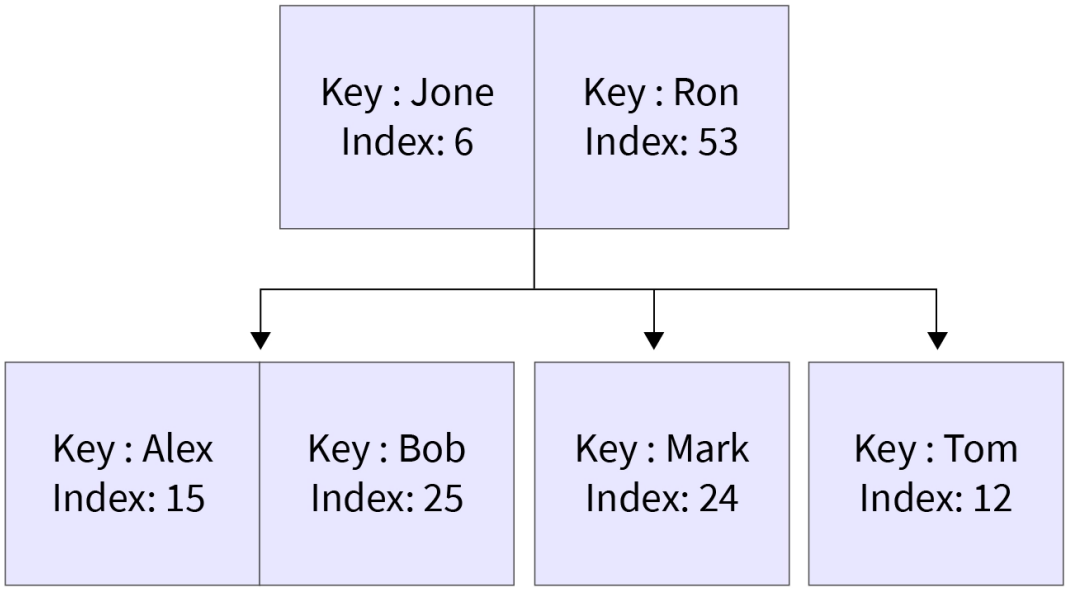
Bitmap has an advantage over B-tress as bitmap performs faster retrieval of certain data (Bitmap is made according to a certain data, hence retrieves faster). Bitmaps are also more compact than B-trees.

There is a drawback with bit mapping, bit mapping requires more overhead during tuple operations on the table. Hence, bit maps are mainly used in data warehouse environments.

Example - We want to store this three-column table in the database.

Name	Marks	Age
Jone	5	28
Alex	32	45
Tom	37	23
Ron	87	13
Mark	20	48
Bob	89	32

The B-tree representation will be like this:



Note: Oracle Database uses Bitmap and B-trees.

Ascending and Descending

As we have discussed above, columns of the index are stored in some sorted manner. Generally, we store these Search Keys in ascending order. These sorted keys allow us to search data the data fastly. We can change the sort order from ascending to descending or something different according to the most frequent queries on the database.

Syntax

Lets see the syntax to store indexing in descending order-

```
CREATE INDEX index_name ON table-name (column-name_1, column-name_2 DESC);
```

By default Sorting Order:

- Character Data: Sorted by ASCII values of the characters.
- Numeric Data: Smallest to largest numbers.
- Date: Earliest date to the latest date.

Column and Functional:

Generally, we prepare the index table with certain column values of the actual database but sometimes we can also use predefined SQL functions like UPPER() or LOWER() or MAX(), etc. to prepare the Search Keys.

Example - We can convert all values in a column to uppercase and stored these results in the index.

Syntax:

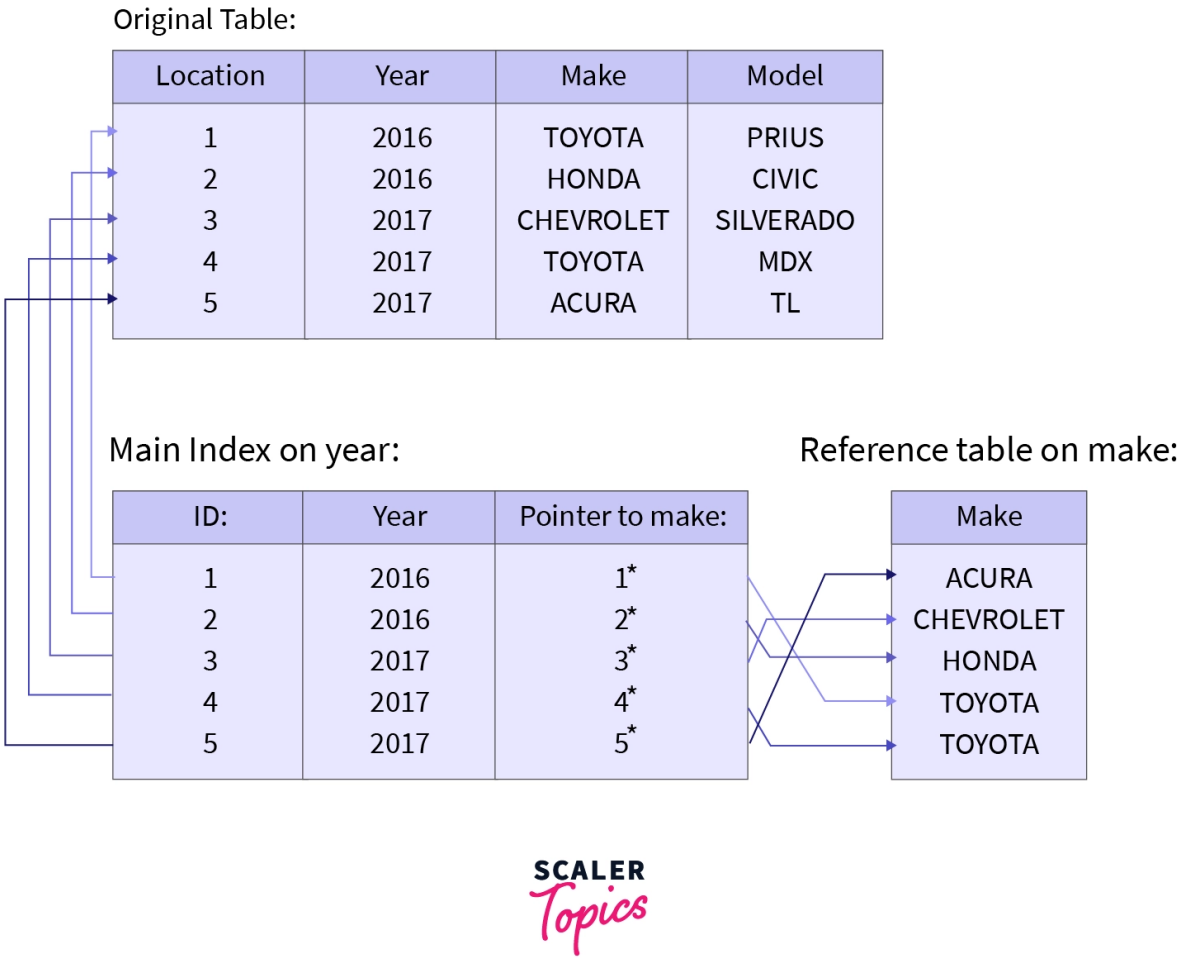
```
CREATE INDEX index-name
ON members(UPPER(target-column));
```

> **Note:** The index table formed used columns values are also termed as Column Index or Column Index-table.

Single-Column and Concatenated

We can create a single-column index table or multi-column index table. Concatenated indexes are made according to certain WHERE clauses(WHERE clause related to the most frequent SQL Queries), hence making the searching or data retrieval faster.

Example - Let us take an example of a multi-column index table:



We can use the primary key to create multiple index tables such as indexing based on year (grouping years) or indexing based on model-name etc. This multi-table indexing will help in getting

specific query results faster.

Note: The multi-column is also termed a Concatenated Index.

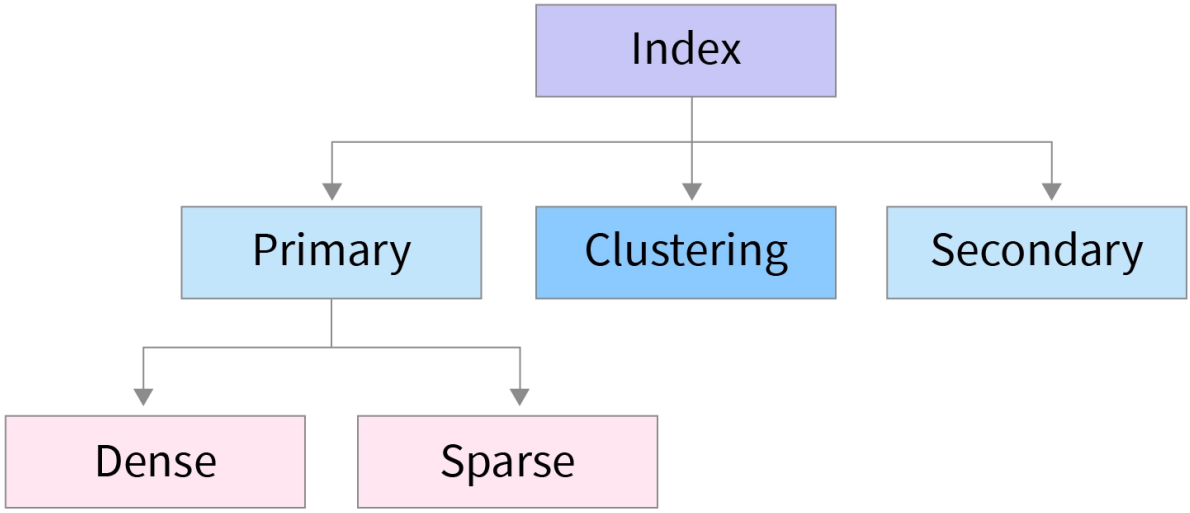
Non-Partitioned and Partitioned

As we know index points to a certain table or block of data but sometimes the data itself is partitioned in a certain manner, so we need to partition the index table as well. Generally, we use the same table partition schema for the partition of the index table which is known as the **Local Partition Index**. We use the same schema so that the data retrieval speed is maintained. However, we can also create our non-partitioned index. This is known as **Global Index** of the partitioned table.

Example - Suppose we have a table namely a student table. If the student table is partitioned according to the roll number(primary key) then the index table of the student table should be partitioned according to roll number as well. This type of partition will help in the grouping of similar data and faster query results.

Types of Indexes

According to the attributes defined above, we divide indexing into three types:



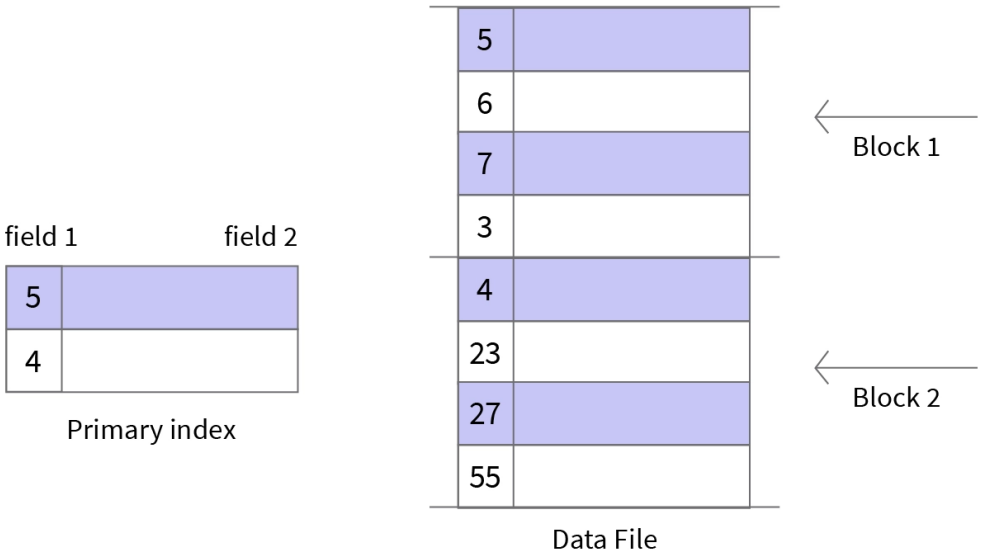
Single Level Indexing

It is somewhat like the index (or the table of contents) found in a book. Index of a book contains topic names along with the page number similarly the index table of the database contains keys and their corresponding block address.

Single Level Indexing is further divided into three categories:

1. **Primary Indexing:** The indexing or the index table created using Primary keys is known as Primary Indexing. It is defined on ordered data. As the index is comprised of primary keys, they are unique, not null, and possess one to one relationship with the data blocks.

Example:

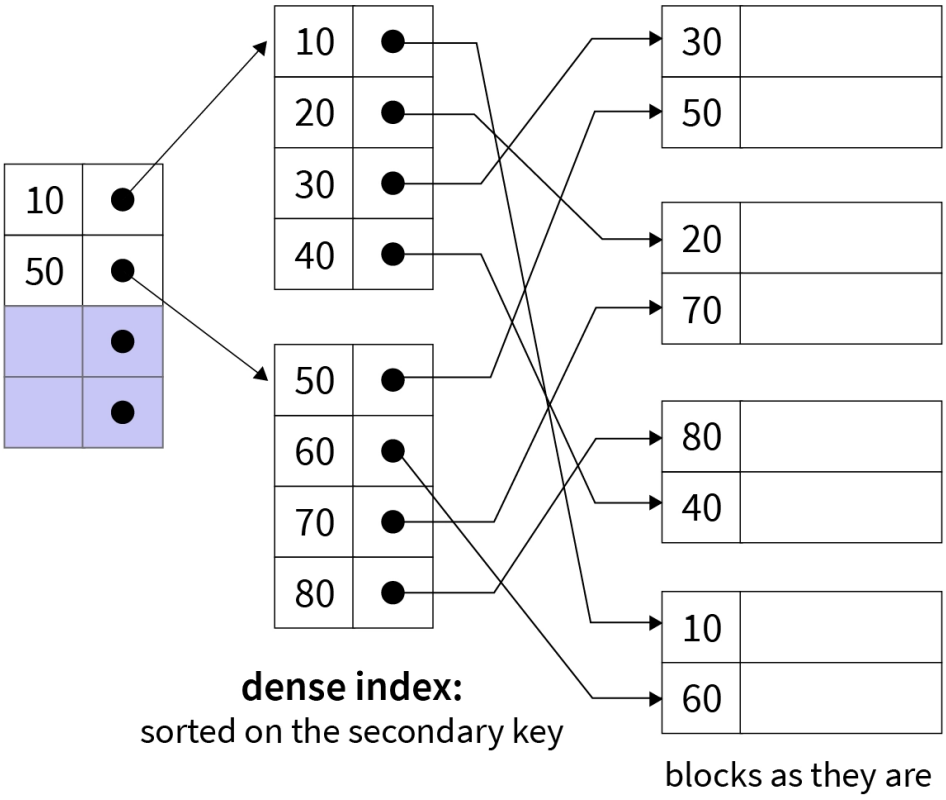


Characteristics of Primary Indexing:

- Search Keys are unique.
- Search Keys are in sorted order.
- Search Keys cannot be null as it points to a block of data.
- Fast and Efficient Searching.

2. **Secondary Indexing:** It is a two-level indexing technique used to reduce the mapping size of the primary index. The secondary index points to a certain location where the data is to be found but the actual data is not sorted like in the primary indexing. Secondary Indexing is also known as non-clustered Indexing.

Example:



Characteristics of Secondary Indexing:

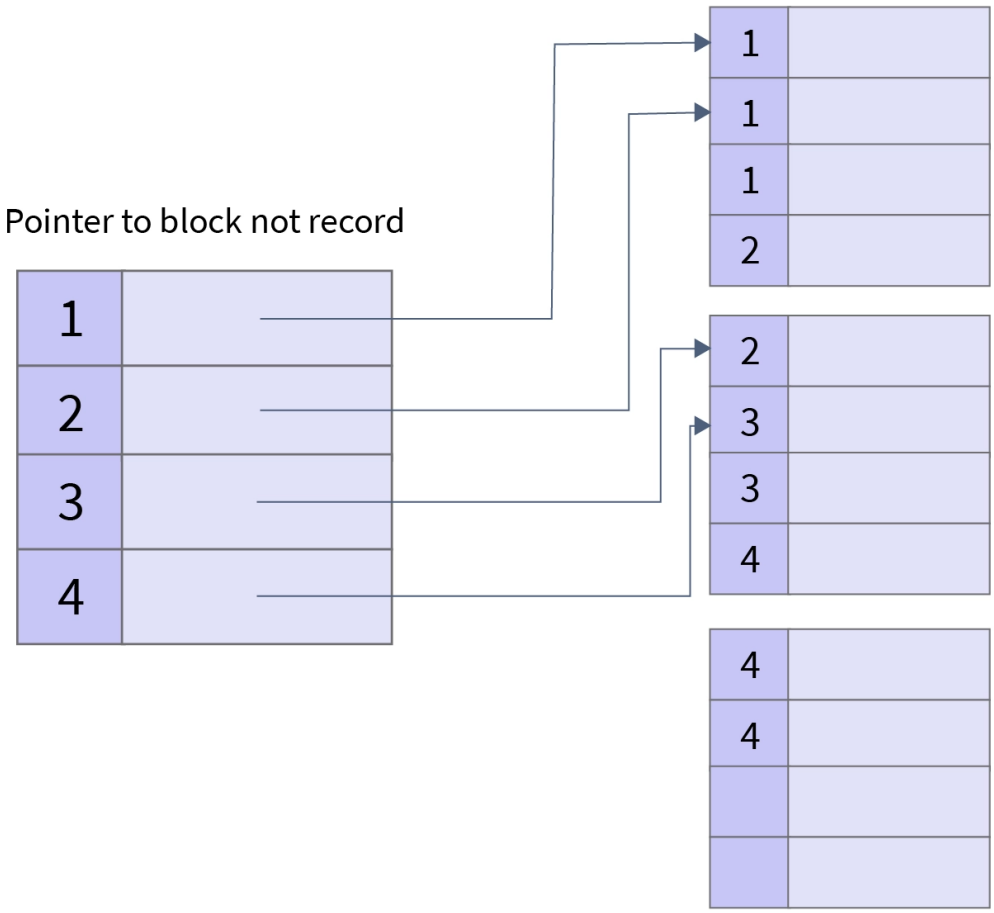
- Search Keys are Candidate Keys.
- Search Keys are sorted but actual data may or may not be sorted.
- Requires more time than primary indexing.
- Search Keys cannot be null.

- Faster than clustered indexing but slower than primary indexing.

3. **Cluster Indexing:** Clustered Indexing is used when there are multiple related records found at one place. It is defined on ordered data. The important thing to note here is that the index table of clustered indexing is created using *non-key* values which may or may not be unique. To achieve faster retrieval, we group columns having similar characteristics. The indexes are created using these groups and this process is known as **Clustering Index**.

Example:

EXAMPLE OF CLUSTERED INDEX -



Characteristics of Clustered Indexing:

- Search Keys are non-key values.
- Search Keys are sorted.
- Search Keys cannot be null.
- Search Keys may or may not be unique.

- Requires extra work to create indexing.

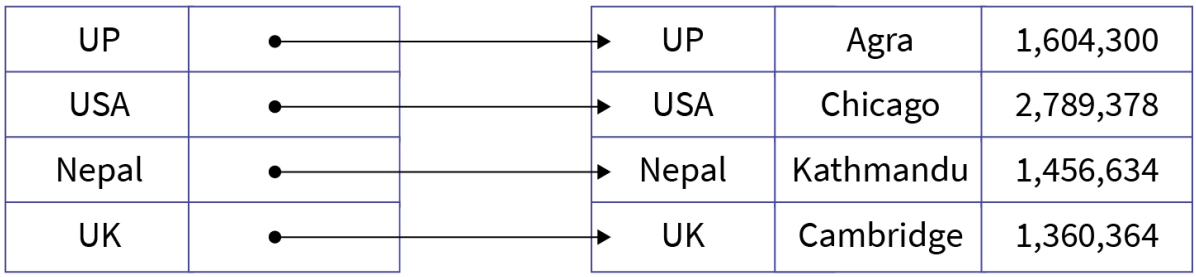
Ordered Indexing:

Ordered indexing is the traditional way of storing that gives fast retrieval. The indices are stored in a sorted manner hence it is also known as ordered indices.

Ordered Indexing is further divided into two categories:

1. **Dense Indexing:** In dense indexing, the index table contains records for every search key value of the database. This makes searching faster but requires a lot more space. It is like primary indexing but contains a record for every search key.

Example:

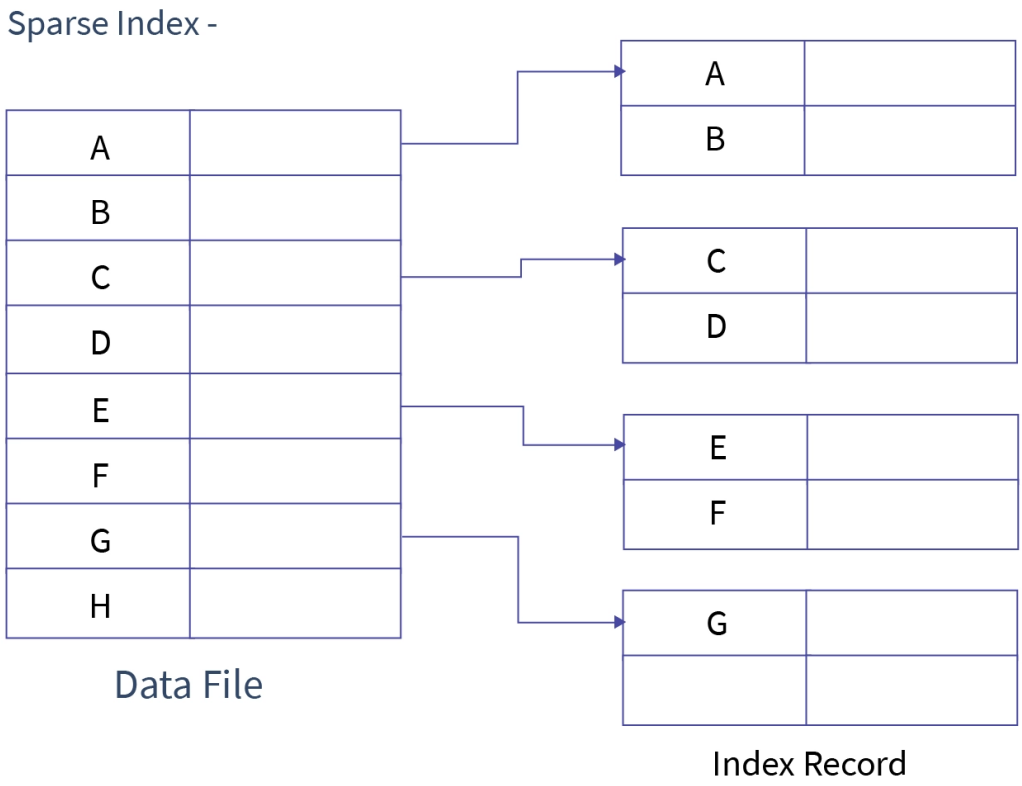


2. **Sparse Indexing:** Sparse indexing consumes lesser space than dense indexing, but it is a bit slower as well. We do not include a search key for every record despite that we store a

Search key that points to a block. The pointed block further contains a group of data.

Sometimes we have to perform double searching this makes sparse indexing a bit slower.

Example:



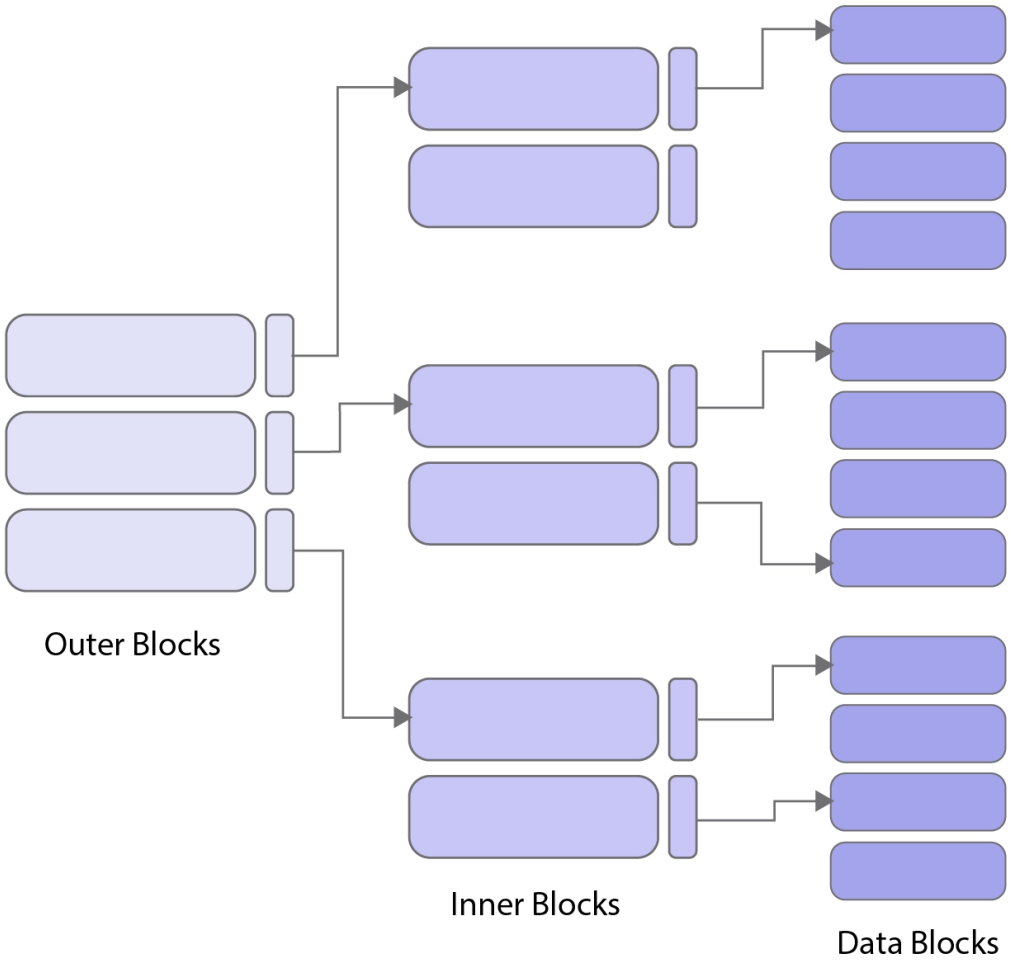
For very few search value in a Data File, there is an Index Record.
Hence the name Sparse Index.

Multi-Level Indexing

Since the index table is stored in the main memory, single-level indexing for a huge amount of data requires a lot of memory space. Hence, multilevel indexing was introduced in which we divide the

main data block into smaller blocks. This makes the outer block of the index table small enough to be stored in the main memory.

Example:



We use the B+ Tree data structure for multilevel indexing. The leaf nodes of the B+ tree contain the actual data pointers. The leaf nodes are themselves in the form of a linked list. This linked list representation helps in both sequential and random access.

Refer to this article to learn more about B and B+ Trees.

Advantages of Indexing

- Indexing helps in faster query results or quick data retrieval.
- Indexing helps in faster sorting and grouping of records
- Some Indexing uses sorted and unique keys which helps to retrieve sorted queries even faster.
- Index tables are smaller in size so require lesser memory.
- As Index tables are smaller in size, they are stored in the main memory.
- Since CPU speed and secondary memory speed have a large difference, the CPU uses this main memory index table to bridge the gap of speeds.
- Indexing helps in better CPU utilization and better performance.

Conclusion

- Indexing is a technique that uses data structures to optimize the searching time of a database query.
- Index table contains two columns namely Search Key and Data Pointer or Reference.
- There are three types of indexing namely Ordered, Single-level, and multi-level.

- Single Level Indexing is divided into three types namely Primary(index table is created using primary keys), Secondary(index table is created using candidate keys), and Clustered(index table is created using non-key values).
- Ordered Indexing is divided into two types namely dense(index table contains records for every search key value of the database) and sparse(do not include search key for every record).
- Multi-level Indexing uses B+ Trees to store data pointers.
- Indexing helps in faster data retrieval and better performance.
- Most commonly used indexing attributes are: Standard (B-trees) and Bitmaps, Ascending and Descending attribute, Column and Functional attribute, Single-Column, and Concatenated attribute, Non-partitioned and Partitioned attribute.