# Indexes

## Key Terms



### Index

> a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure

---

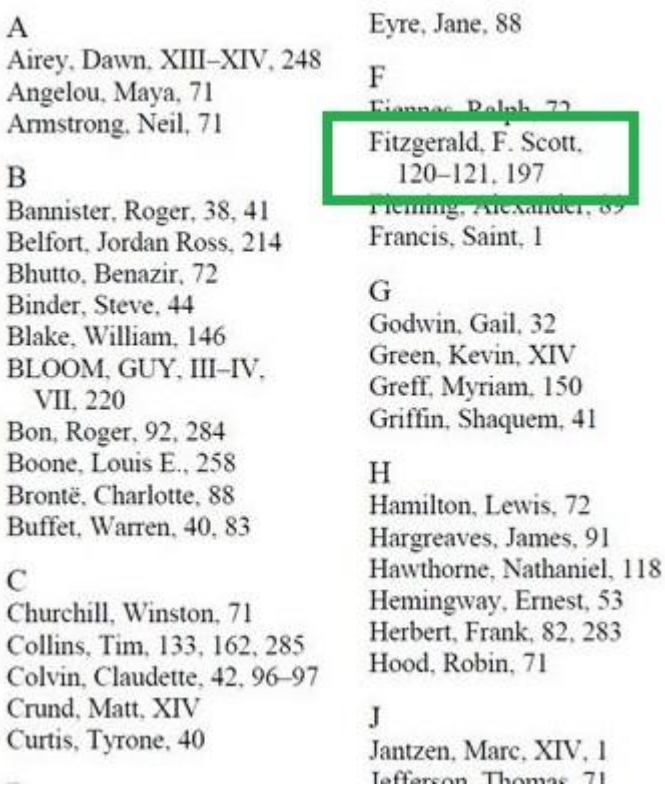
## Database Indexes

Data stored in a database is persisted to disk. This means that the data is stored in a file on the disk. To fetch a single row of the data, the database needs to know the address of the row (or the block). A basic way to do this is to iterate over all the blocks and find the one that matches the key. This has two issues:

- The database has to iterate over all the blocks. Worst case complexity is O(n) where n is the number of blocks in the database.
- Large number of I/O operations are required to read the data. I/O operations are much slower than accessing data in memory.

In order to reduce I/O operations, the database can use indexes to speed up the lookup process. An index is a data structure that stores the address of the block that contains the data. Thus, to fetch a single row of data, the database only needs to look up the index and then read the data from the block.

> You have 10 minutes to find information about an author in a 300-page book; unless you are The Flash, you cannot read the entire book





> At the very end of the book, the Index section has an ordered list of important words where you can easily find what you are looking for (in our scenario, author names). Next to each topic, there are some numbers that tell you in which pages of the book the author is mentioned

## Types of indexes

| | **Unique** | **Non-unique** |
|---|---|---|
| **Ordered** | Primary Index | Clustered Index |
| **Unordered** | Secondary Index | Secondary Index |

**Primary Index**

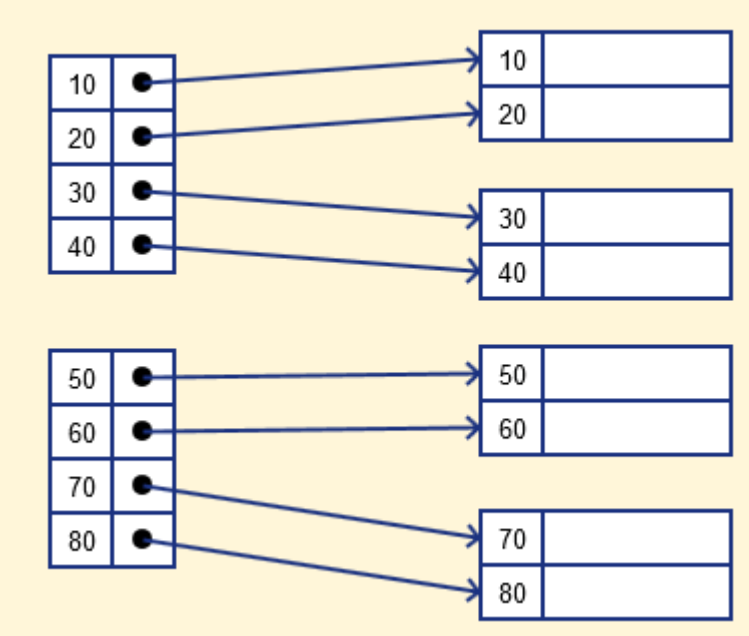
Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

A primary key of a table always has a primary index created.

A primary index can be of two types

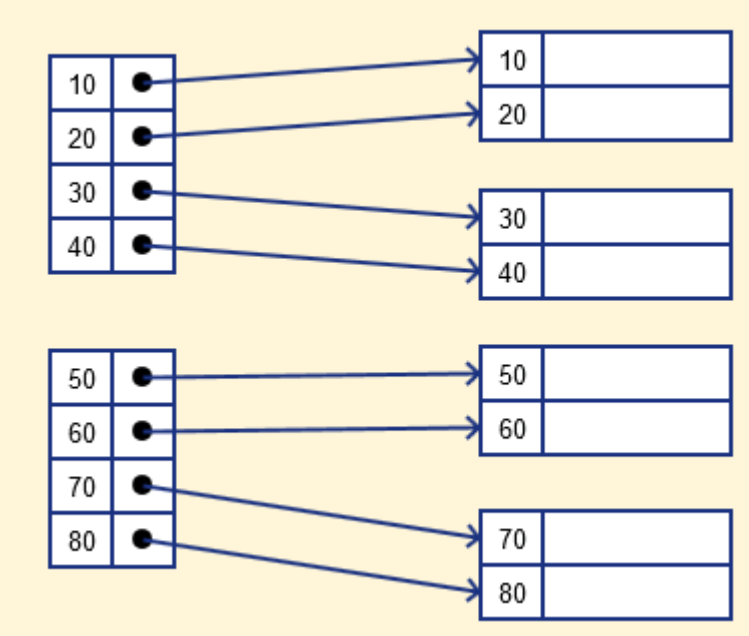
- Dense - a record is created for every search key valued in the database
- Sparse - an index record that appears for only some values

**Dense Index** In a dense index, a record is created for every search key valued in the database. This helps you to search faster but needs more space to store index records. In this Indexing, method records contain search key value and points to the real record on the disk.




**Sparse Index** It is an index record that appears for only some of the values in the file. Sparse Index helps you to resolve the issues of dense Indexing in DBMS. In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.

However, sparse Index stores index records for only some search-key values. It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.

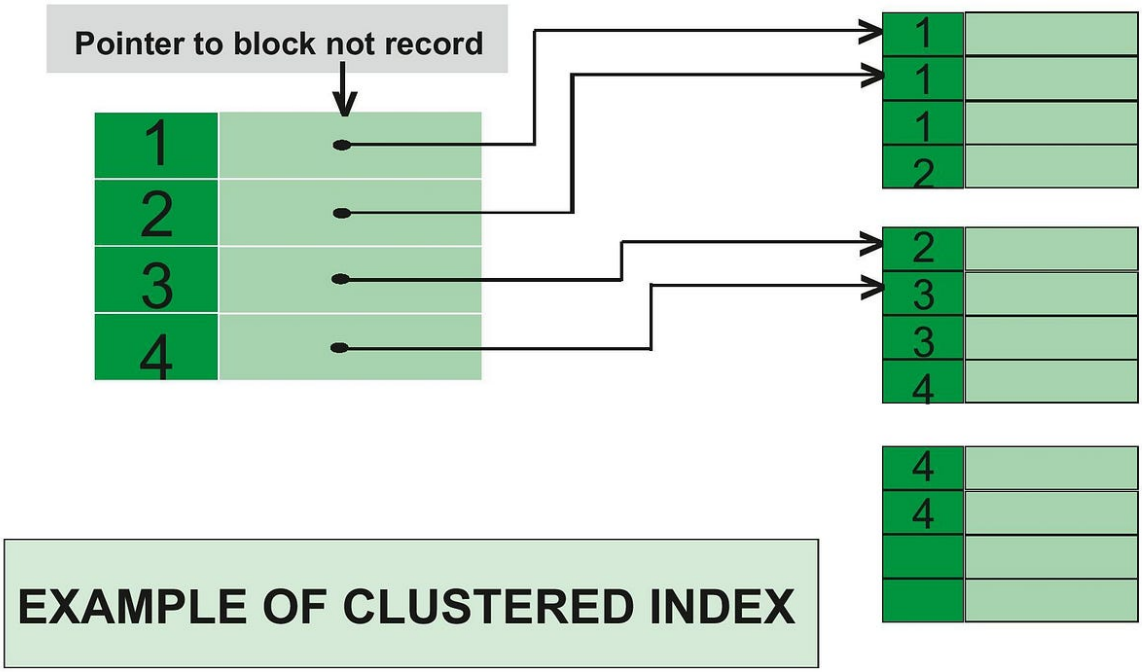



**Clustered Index**

> Cluster index is a type of index which sorts the data rows in the table on their key values.

A clustered index is used when the data is sorted but might contain duplicate values.

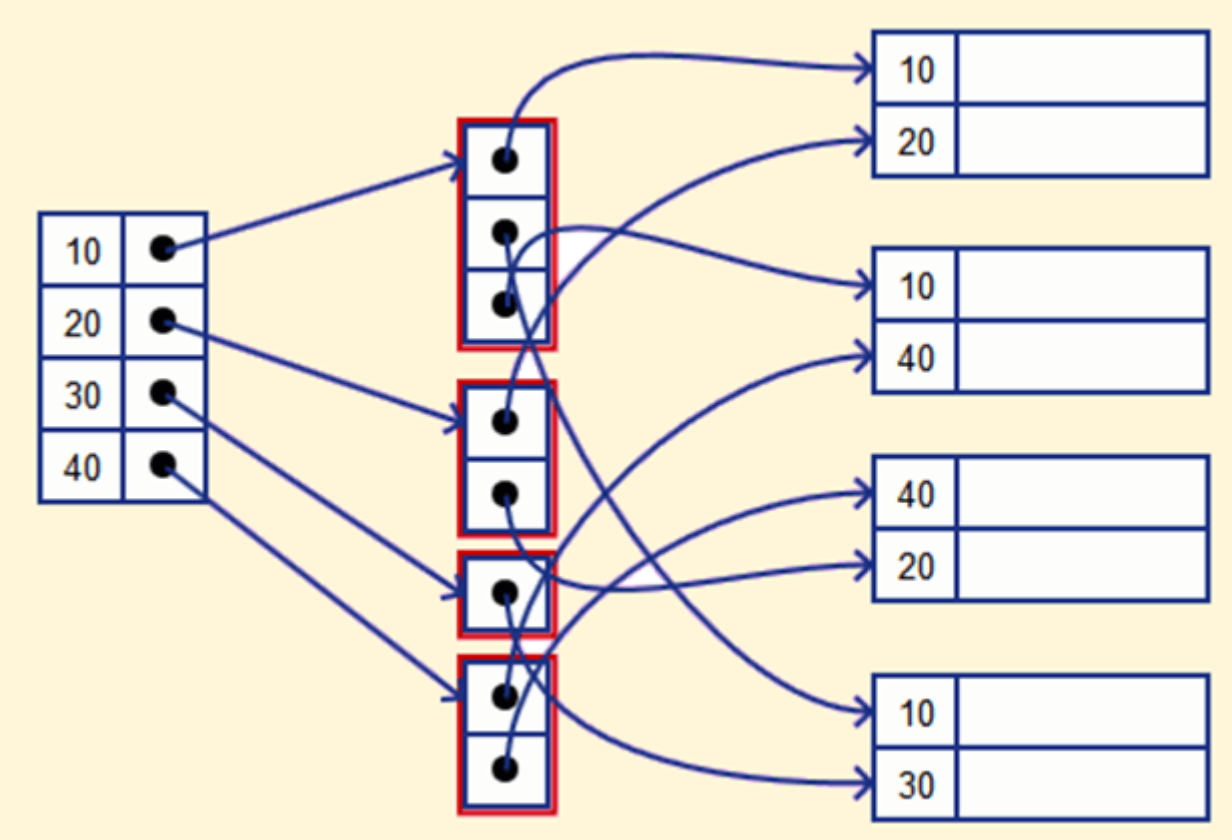
Clustered indexes work similar to the primary index expect they maintain a flag in each page to indicate whether the next block contains the same value or not due to non-unique keys.




**Secondary Index**

> The secondary Index in DBMS is an indexing method whose search key specifies an order different from the sequential order of the file

You might have several use cases in your application where you don't query the database with a primary key. In our example phone_no is the primary key but we may need to query the database with pan_no, or name. In such cases you need secondary indices on these columns if the frequency of such queries is very high.

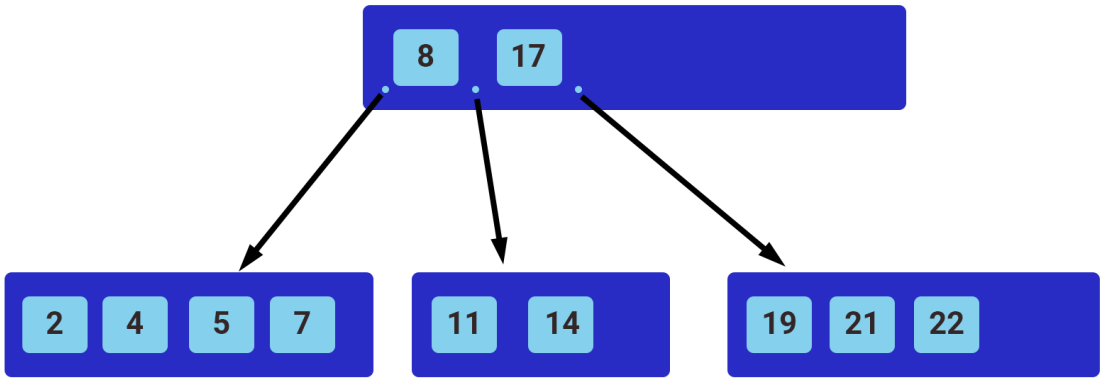



Since the values are not ordered and unique, maintaining a lookup table like clustered index is not possible. Secondary indexes create an intermediate table where the data is sorted, and it contains the address of the data block where it is stored. This is an example of a dense lookup table.

Now, similar to clustered indexes, a primary lookup table is created as a sparse table which contains the addresses of the intermediate table.

### Implementation

The most common indexes use a BALANCED TREE behind the scenes to speed up queries. Most database engines use either a balanced tree or a variation of a balanced tree like a B+ TREE. The structure of a general balanced tree is shown below




The top node is the root, and those below it are either child nodes or leaf nodes. We always start searching for our row from the root node and compare if the value we're searching for is less than or greater than the value in the node at hand. The result of the comparison tells us which way to go, left or right, depending on the result of our comparison. In the example above, all values lower than 8 take us to the left, while values greater than 8 take us to the right, and so on.

The B+ Tree is the most common index data structure in database systems. It has several important properties:

- Perfectly balanced search tree (all leaf nodes are the same depth)
- It generalizes the binary search tree from two children to M children Where M is a fixed constant for any particular B+ Tree
- Every inner node other than the root must be at least half capacity (has m children where $\lceil M/2 \rceil \leq m \leq M$)
- Every inner node with m children has $m-1$ keys
- Every leaf node must be at least half full (holds k keys where $\lceil (M-1)/2 \rceil \leq k \leq M-1$)
- Insert and delete operations will rebalance the tree to maintain these properties

### When to use indexes?

- To find the rows matching a WHERE clause quickly.
- To eliminate rows from consideration.
- To retrieve rows from other tables when performing joins.
- To find the MIN() or MAX() value for a specific indexed column.
- To sort or group a table (under certain conditions).
- To optimize queries using only indexes without consulting the data rows.

### When Should Indexes Be Avoided?

- Indexes should not be used on small tables.

- Indexes should not be used on columns that return a high percentage of data rows when used as a filter condition in a query's WHERE clause.
- Tables that have frequent, large batch update jobs run can be indexed. However, the batch job's performance is slowed considerably by the index.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed. Maintenance on the index can become excessive.

## Some queries to get you started

Create the students relation with an index on phone number

```
CREATE TABLE `students` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `age` int DEFAULT NULL,
  `phone` int DEFAULT NULL,
  `email` varchar(255) NOT NULL,
  `address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `phone_idx` (`phone`)
)
```

Or modify the students relation to have an index on phone number

```
ALTER TABLE `students` ADD INDEX `phone_idx` (`phone`)
```

## References

- Indexes

## Further reading

- Indexes
- Indexes Data Structure