Microsoft

Indexes in SQL

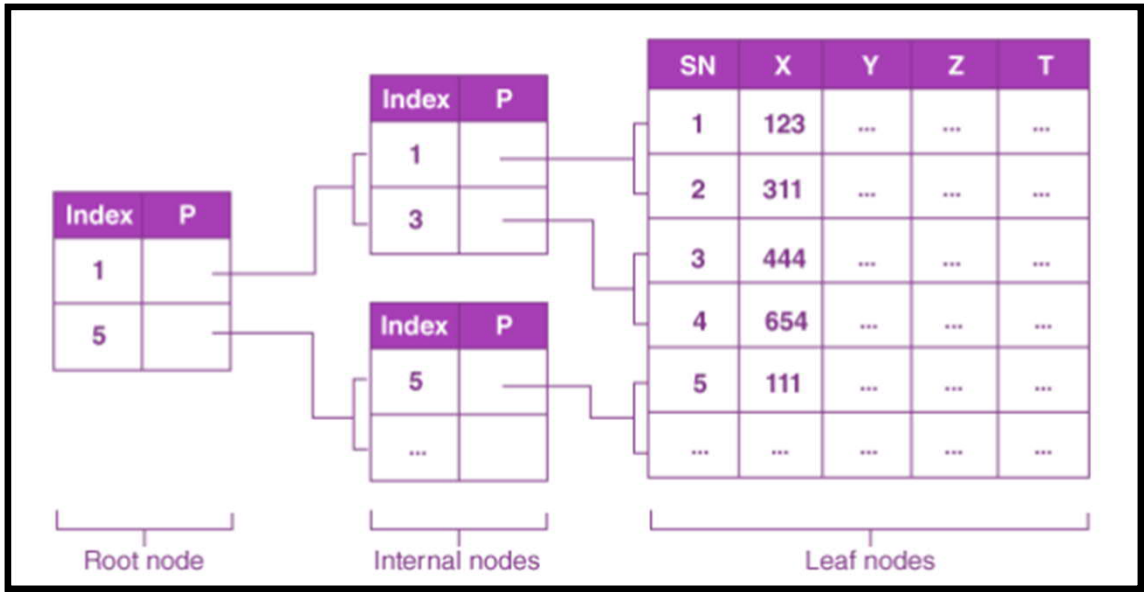
What is an Index?

Think about a regular book:

- At the end of the book, there's an index that helps to quickly locate information within the book.
- The index is a sorted list of **keywords**
- Next to each **keyword** is a set of page numbers pointing to the **pages** where each keyword can be found.

What are SQL Indexes?

- An **index** is a special data structure that improves data retrieval speed.
- Works like a **book index** → find data without scanning the whole table.
- Stored separately from the main table but linked to it.
- Does **not change data**, only speeds up queries.



Why is indexing important?

- Speeds up data retrieval and query execution.
- Supports **efficient access types**:
  - Find specific values
  - Find values in a range
- Reduces **access time** for searches.
- Makes **inserts and deletes** faster, but requires extra work to update the index after changes
- Uses **extra space** to improve performance - stores additional structures (like pointers or trees) to achieve that speed.

Microsoft

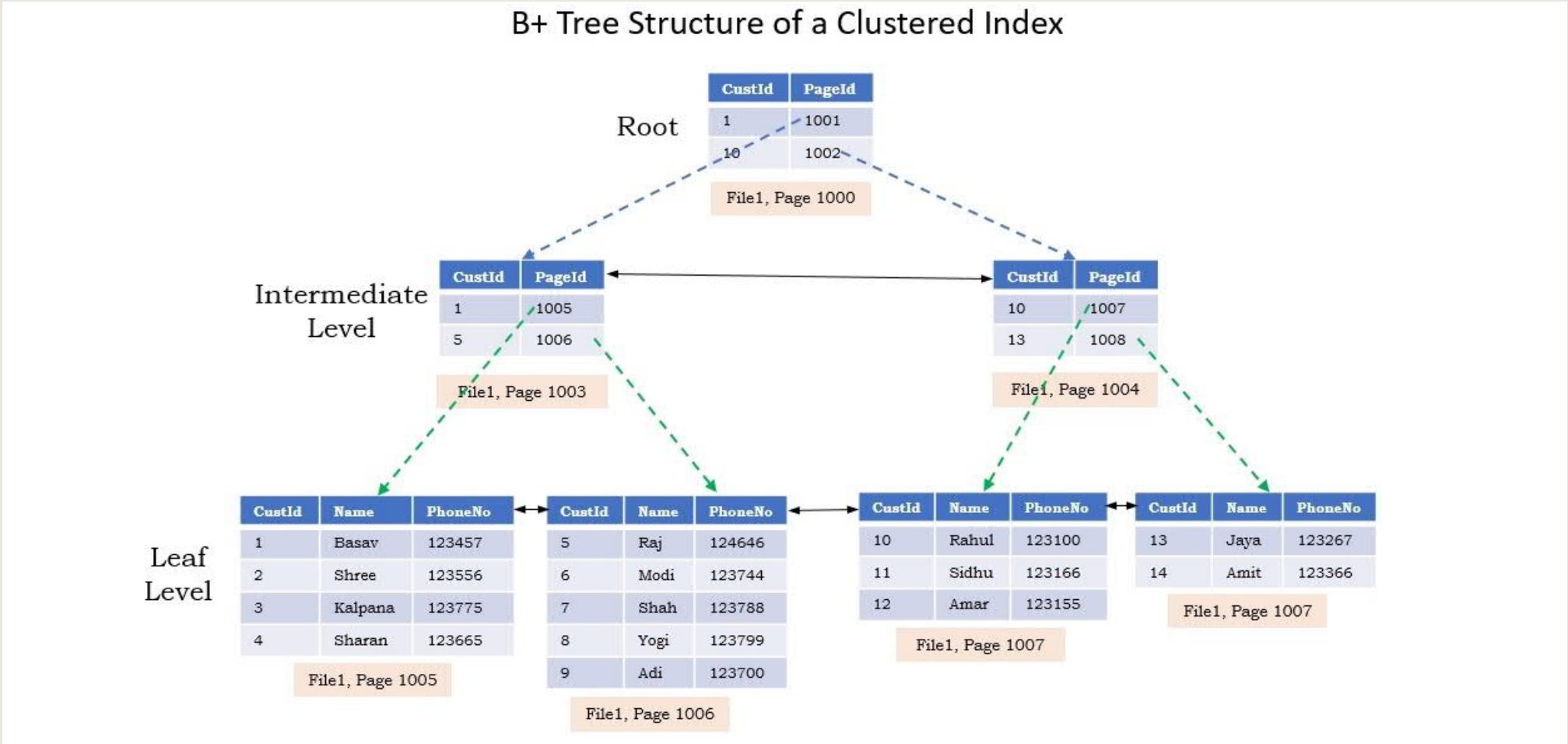
Index Types

Clustered Index
Non-Clustered Index

Microsoft

Clustered Index

Clustered Index



Clustered Index (cont...)

- Clustered indexes, implemented as a B+ Trees, sort and store the data rows in the table or view based on their key values.
  - These key values are the columns included in the index definition.
  - There can be only one clustered index per table, because the data rows themselves can be stored in only one order.
- The only time the data rows in a table are stored in sorted order is when the table contains a clustered index.
  - When a table has a clustered index, the table is called a clustered table.
  - If a table has no clustered index, its data rows are stored in an unordered structure called a heap.

Clustered Index

- Speeds up queries that return **ranges of data**
  - **Example**: **`WHERE EmployeeID BETWEEN 100 AND 200;`**
- Ideal for columns **frequently sorted or searched**.
  - Example: **EmployeeID**
- Good for **range queries, ORDER BY, and grouping** since rows are stored in index order

Problem Domain:

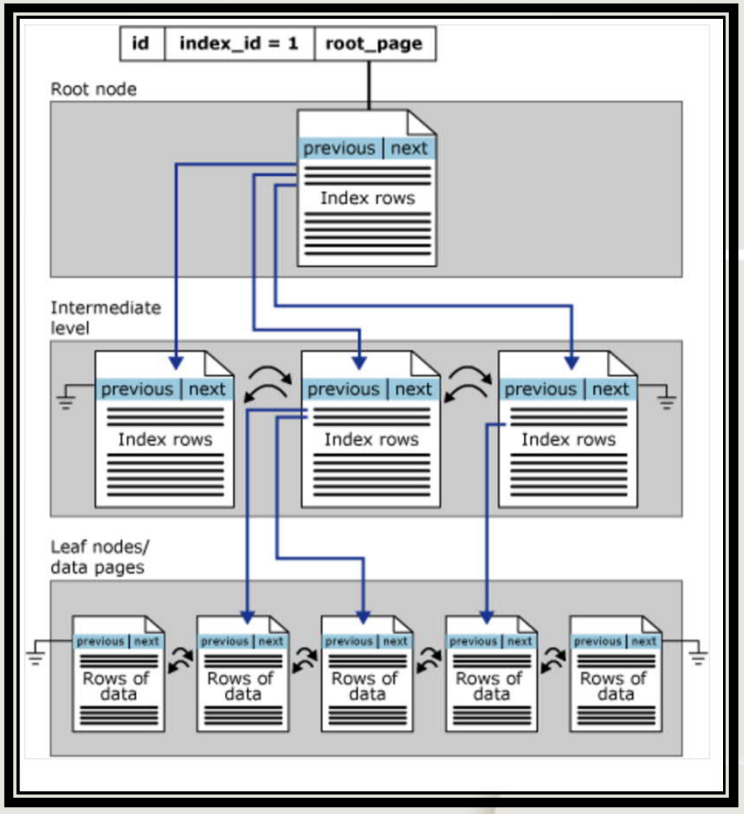
- Works best for **or mostly unique columns**.
- Not ideal for columns **frequently updated or inserted randomly** (reordering is costly).
  - SQL Server may need to **physically move rows around to maintain the sorted order** of the clustered index.
- Usually applied to **primary key columns**, as each table can have only **one clustered index**.

Indexes and constraints

- SQL Server automatically creates indexes when PRIMARY KEY and UNIQUE constraints are defined on table columns.
  - For example, when you create a table with a UNIQUE constraint, Database Engine automatically creates a nonclustered index.
  - If you configure a PRIMARY KEY, Database Engine automatically creates a clustered index, unless a clustered index already exists.
  - When you try to enforce a PRIMARY KEY constraint on an existing table and a clustered index already exists on that table, SQL Server enforces the primary key using a nonclustered index.

Architecture:

- Organized as a **B+ tree**
- **Root node**: top level for searching
- **Intermediate nodes**: guide the search
- **Leaf nodes**: contain actual table rows (physically sorted)
- **Inserts** follow key order: New rows are placed where they fit to keep the table sorted.
- **Partitions**: Large tables can be split into sections, each with its own B+ tree for faster queries.
- Supports **variable-length columns**: Large or flexible-size data is managed without breaking the index structure.



Usage

In this table, the rows are physically stored on disk in the order of **EmployeeID**.

EmployeeID	Name	Dept	Salary
101	Ali	IT	5000
102	Sarah	CS	3400
103	Omer	Physics	4500
104	Ben	Math	6000

- Table rows are **physically sorted by EmployeeID**.
- **Leaf nodes** of the index contain the actual table data.
- Only **one clustered index per table**; usually on the **primary key**.

Query:

```
SELECT *
FROM Employees
WHERE EmployeeID BETWEEN
101 AND 103;
```

- Database uses the **clustered index B+ tree** to find **EmployeeID 101** quickly.
- Reads **consecutive rows** in order until EmployeeID 103.
- **No need to scan the entire table**, so the query is faster.

Clustered Index is like a Dictionary

- The words (data rows) are stored in alphabetical order (index key order).
- You can quickly find a word because you know where it should be.
- The dictionary itself is the data—no need to look elsewhere.