Microsoft

# Non-Clustered Indexes

# Recap - Clustered & Non-Clustered Index

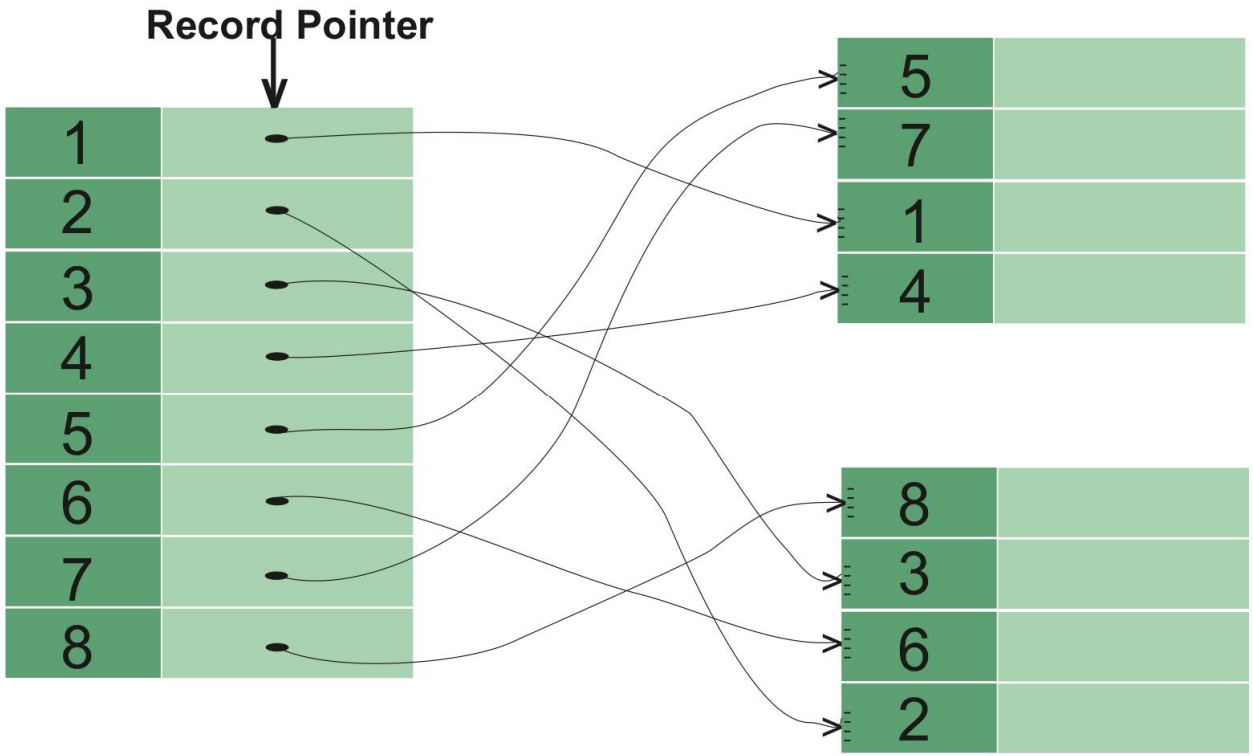
- With a clustered index the rows are stored physically on the disk in the same order as the index. Therefore, **there can be only one clustered index.**
- With a non-clustered index there is a **second list** that has pointers to the physical rows.
  - You can have many non clustered indices, although each new index will increase the time it takes to write new records.
- It is generally faster to read from a clustered index if you want to get back all the columns. You do not have to go first to the index and then to the table.
- Writing to a table with a clustered index can be slower, if there is a need to rearrange the data.

# Non-Clustered Index (Heap Tables)

A **table** without a clustered index is called a heap.

- Data is stored without any defined order, and rows are located using a **Row Identifier (RID)**, which includes the file number, data page number, and slot on the page.
- To guarantee the order of rows returned from a heap in response to a SELECT query, use the ORDER BY clause.
  - To specify a permanent logical order for storing the rows, create a clustered index on the table, so that the table is not a heap.

# Non-clustered Index



# Non-Clustered Indexes

- Speeds up **exact match queries**
  - Example: **WHERE ManagerID = 5**.
- Useful for **JOIN, GROUP BY, and aggregate operations**.
- Multiple indexes can optimize **different query patterns**.

**Problem Domain:**

- Avoid creating too many indexes on frequently updated tables , as they slow down insert, update, and delete operations.
- Every insert, update, or delete must also **update all related indexes**, which adds extra overhead and slows performance.

# Non-Clustered Indexes - Implementation

The implementation of the nonclustered index depends on whether the data pages of a table are managed as a Heap or as a clustered index.

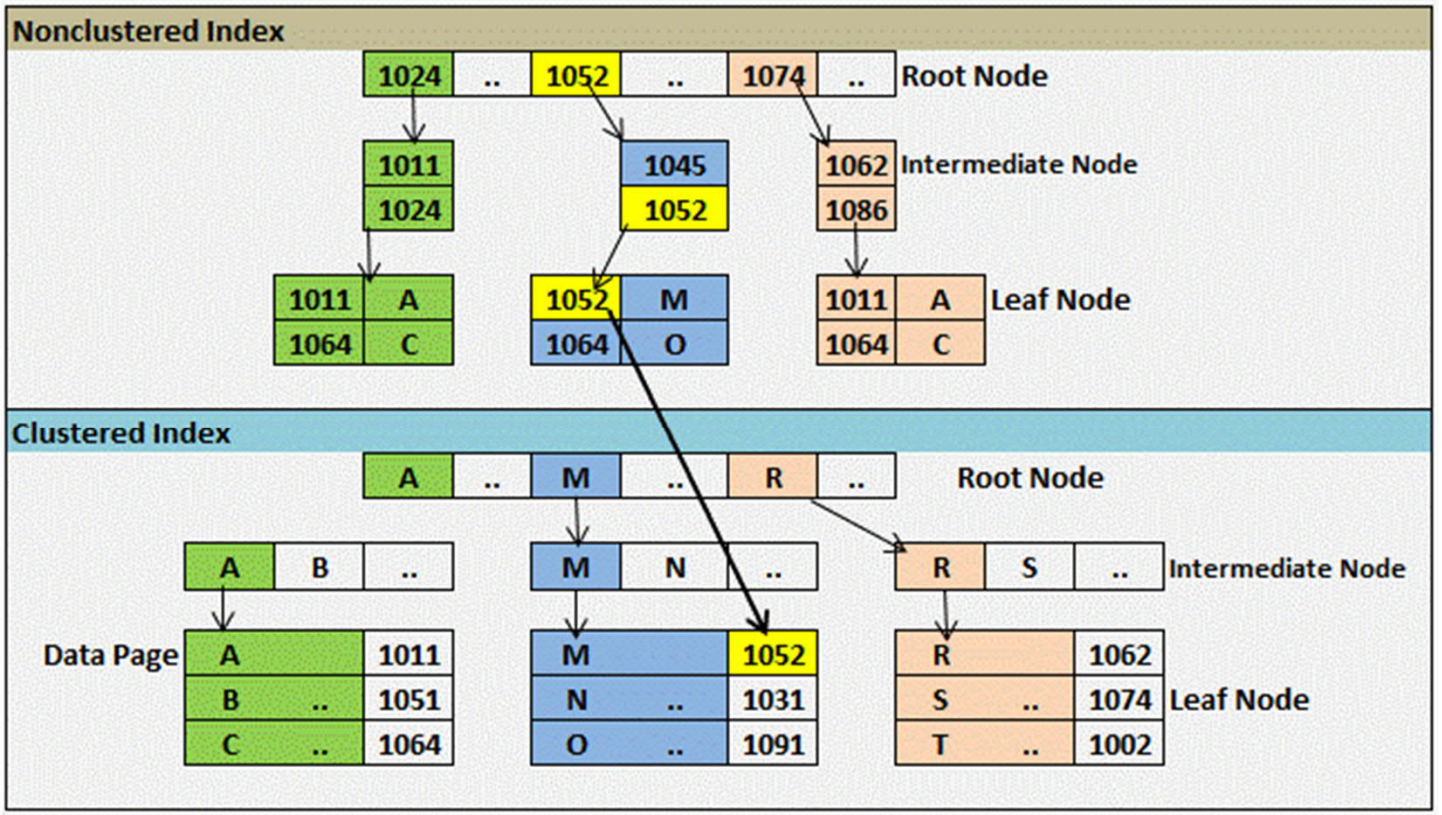
**Heap based Non-clustered Index structure in SQL Server:**

- If non-clustered index is built in a heap then SQL Server uses pointers in the leaf node index pages that point to a row in the data pages.

**Non-clustered Index based on Clustered Index Structure:**

- If we have a table with clustered index, SQL Server builds a non-clustered index on the top of the clustered index structure and SQL Server uses clustering keys in the leaf node index page of the non-clustered index to point the cluster index.

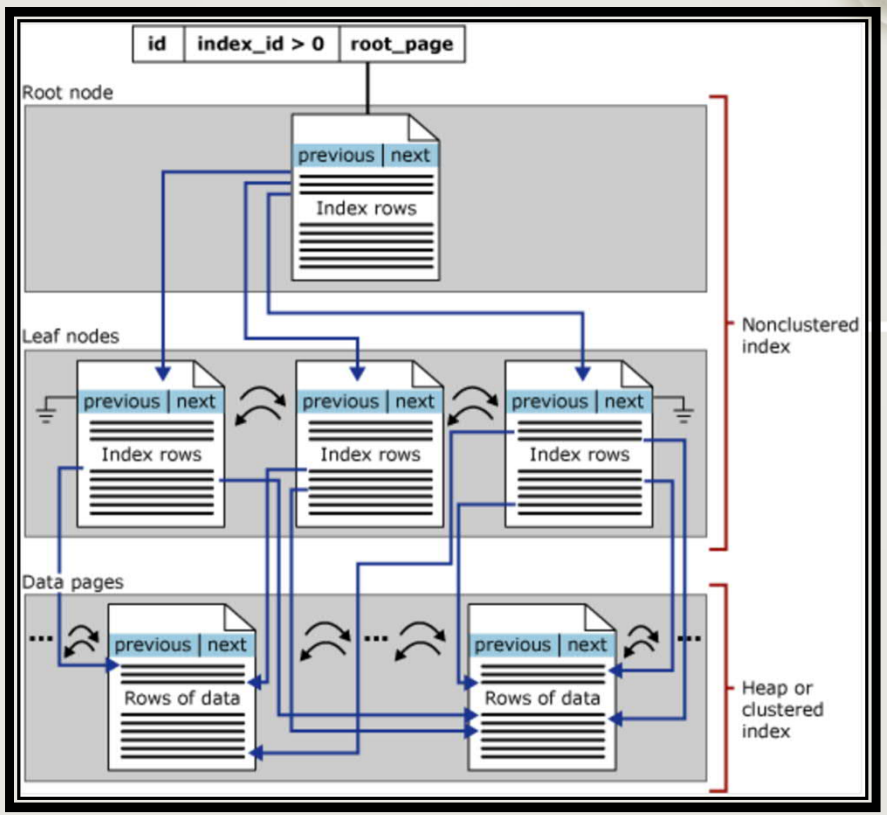
# Non-Clustered Indexes - Implementation




Nonclustered Index structure navigation on Clustered Index Structure in SQL Server

# Architecture

- Organized as a **B+ tree** for fast searching.
- **Root node:** entry point for searches.
- **Intermediate nodes:** guide the search down the tree.
- **Leaf nodes:** contain **key values, included columns, and row locators** (pointing to actual table rows).
- **Row locator:**
  - Heap → Row ID (RID) - Direct physical location of the row
  - Clustered table → Clustered index key - Use key to locate row in the clustered index
- **Inserts:** follow key order; leaf splits maintain balance.



## Example of use:

| EmployeeID | Name | Dept | ManagerID |
|---|---|---|---|
| 101 | Ali | IT | 2 |
| 102 | Sarah | CS | 7 |
| 103 | Omer | Physics | 2 |
| 104 | Ben | Math | 5 |

- **Index on ManagerID** column lets the database quickly find employees reporting to a manager.
- **Leaf nodes** store **ManagerID + row locator**, not full data.

- **Row locator** points to the actual row in database

| ManagerID | Row Locator |
|---|---|
| 7 | RID -> points to row 102 |
| 5 | RID -> points to row 104 |

**Query:**

```
SELECT Name, Department
FROM Employees
WHERE ManagerID = 5;
```

- Query WHERE ManagerID = 5:
  1. Searches index leaf nodes.
  2. Finds all matching rows.
  3. Follows row locators to fetch Name and Department.
- Multiple rows with the same ManagerID are **sorted in leaf nodes by key**.

# Example of how Index is formed in Microsoft SQL

```sql
CREATE TABLE Products_table (
    ProductID INT CONSTRAINT PK_Products PRIMARY KEY NONCLUSTERED,
    -- PK but nonclustered
    ProductName NVARCHAR(50),
    Price DECIMAL(10,2)
);

-- Now add clustered index manually
CREATE CLUSTERED INDEX IX_Prod_ID
ON Products_table(ProductID);

-- Add nonclustered index
CREATE NONCLUSTERED INDEX IX_Prod_Price
ON Products_table(Price);
```

This block **creates a table with a nonclustered primary key, then adds a clustered index on ProductID and a nonclustered index on Price.**

```sql
SELECT name, type_desc
FROM sys.indexes
WHERE object_id = OBJECT_ID('Products_table');
```

This query **retrieves the names and types (clustered/nonclustered) of all indexes that exist on the table Products_table.**

Results | Messages

| | name | type_desc |
|---|---|---|
| 1 | IX_Prod_ID | CLUSTERED |
| 2 | PK_Products | NONCLUSTERED |
| 3 | IX_Prod_Price | NONCLUSTERED |

# Without records inserted

Results | Messages

| | IndexName | IndexType | index_id | TotalPages | TotalRecords | SizeKB | SizeMB |
|---|---|---|---|---|---|---|---|
| 1 | IX_Prod_ID | CLUSTERED | 1 | 0 | 0 | 0 | 0.000000 |
| 2 | PK_Products | NONCLUSTERED | 2 | 0 | 0 | 0 | 0.000000 |
| 3 | IX_Prod_Price | NONCLUSTERED | 3 | 0 | 0 | 0 | 0.000000 |

- Index structures are created when you define **PRIMARY KEY / CLUSTERED / NONCLUSTERED**.
- But until rows exist in the table → **no records are stored in the indexes**.
- A record = one **row** of your table

**In SQL Server B+ Tree Index**

- **Root Page** → the topmost page (only one).
- **Intermediate Pages** → the middle levels (if index is large enough).
- **Leaf Pages** → the bottom level:
  - For a **clustered index** → leaf pages = **actual data rows**.
  - For a **nonclustered index** → leaf pages = **index key + pointer to data**.

# With records inserted

```
INSERT INTO Products_table (ProductID, ProductName, Price)
VALUES (1, 'Mouse', 25.00),
       (2, 'Keyboard', 45.00),
       (3, 'Monitor', 250.00),
       (4, 'Laptop', 1200.00);
```

Results | Messages

| | IndexName | IndexType | index_id | TotalPages | TotalRecords | SizeKB | SizeMB |
|---|---|---|---|---|---|---|---|
| 1 | IX_Prod_ID | CLUSTERED | 1 | 1 | 4 | 8 | 0.007812 |
| 2 | PK_Products | NONCLUSTERED | 2 | 1 | 4 | 8 | 0.007812 |
| 3 | IX_Prod_Price | NONCLUSTERED | 3 | 1 | 4 | 8 | 0.007812 |

**IX_Prod_ID (Clustered Index)**

- Stores the **actual table rows**.
- 4 rows fit into **1 page (8 KB)**.

**PK_Products (Nonclustered on ProductID)**

- Stores **ProductID + pointer** to clustered index row.
- 4 entries, fits in **1 page (8 KB)**.

**IX_Prod_Price (Nonclustered on Price)**

- Stores **Price + pointer** to clustered index row.
- 4 entries, also **1 page (8 KB)**.

# Non-Clustered Index is like a index in the back of a book …

- It lists topics and page numbers (pointers).
- You still need to flip to the page to read the actual content.