# Indexing in DBMS

## What is Indexing

Indexing is a **shortcut** to find data faster in a database.
Instead of searching the entire table, the database uses the **index** (a data structure) to jump directly to the location of the required data on disk.

Think of it like a **book index** → instead of reading all pages, you look at the index and jump to the exact page number.

## Why Indexing?

- Reduces **disk access**
- Helps in **fast read operations** (SELECT, WHERE, JOIN etc.)
- Works as a **secondary access path** (primary path is the actual table)

## Index Structure

An index has:

### 1.Search Key

- A copy of **Primary key / Candidate key / Any column**
- Used to search inside the index.

### 2.Data Reference (Pointer)

- Contains address of the disk block
- Points to the actual row.

Note:

**Index file is always sorted** (for fast searching).

## Real-World Example

Think of **contact numbers** in an old phone diary.

- Names sorted alphabetically → index
- Each name pointing to phone number → pointer
- Diary pages → actual data file

Instead of flipping all pages, you go straight to “S” for “Shyam”.

That’s indexing.

## Types of Indexing

## 1.Primary Index (Clustering Index)

#### Condition:

Data file must be **sorted** on the search key.

->Search key may or may not be the primary key.

(Using primary key as index name is wrong — nonstandard usage.)

### Dense Index

- Has an index entry **for every** search-key value.
- Each entry =
  **Search key + pointer to 1st record** in data file.
- Needs more space.

#### Real-World Example

Think of a **dictionary**: every word is listed → dense index.

### Sparse Index

- Has index entry for **only some** search-key values.
- Number of index entries = **number of blocks** in the data file.

#### Real-World Example

Think of **telephone directory** with A, D, H, M, S headers only.
To find “Rohan”, you go to “R” block under “M–S” section.

## Primary Index Based on Key Attribute

- Data sorted on **primary key**
- Use primary key as search key
- **Sparse index** is created
- Entries = **no. of blocks** in datafile

## Primary Index Based on Non-Key Attribute

- Data sorted on non-key attribute (like Dept_ID)

- Each **unique value** gets an entry → Dense index
- Example:
  Company having many employees in different departments →
  all employees of same dept stored together → clustering index.

## Secondary Index (Non-Clustering Index)

- Data file is **not sorted** → cannot use primary indexing
- Can be built on **any column (key / non-key)**
- No. of entries = **no. of rows in table**
- Always a **dense index**
- Called secondary because one primary index usually already exists.

## Multi-Level Index

- When single index becomes too large
- Divide index into **levels** (Index on index)
- Works like **B-Tree hierarchy**

Analogy:
Book → Chapter index → Sub-topics → Actual pages

## Advantages of Indexing

### 1. Faster Access & Retrieval

Query searches only the index instead of full table.

**Example:**
Looking for word "ELEPHANT" in dictionary
→ go to "E" section → jump directly
→ very fast

But without index
→ you would read page by page.

### 2. Less Disk I/O

Index helps database avoid scanning entire disk blocks.

**Real Example:**
Think of a library with millions of books.
You're searching for "Operating System" books using the library computer.

Without index → librarian searches entire shelf.
With index → computer tells exact rack + row.

Less walking (I/O).

## Limitations / Disadvantages of Indexing

### 1. Extra Space Required

Index table itself takes disk space.

Example:

Like keeping a separate **index book** in library
to quickly find where each book is kept.

Library shelves = data
Index book = index

More books → bigger index book → more storage.

### 2. Slower INSERT operations

Whenever new data is inserted:

1. Actual table changes
2. Index also needs to update
3. Index may need to be sorted again

This extra work slows down inserts.

Example:

Imagine adding a new name “Arjun” to a sorted contact list.
You can't just add at the end —
you must insert it in the correct alphabet position
→ extra time.

### 3. Slower DELETE operations

When a row is deleted:

1. Data is removed
2. Index entry must also be removed
3. Sorting/structure must remain intact

Takes more time.

Example:

Removing a contact from sorted list →
you must rearrange and shift entries up.

## 4. Slower UPDATE operations

If the **indexed column** is updated:

- Old index entry removed
- New index entry created
- Re-sorting needed

Example:

If you change your contact name from
"Vijay" → "Ajay",
you must move the entry from **V** section to **A** section.

This re-arrangement slows updates.

## Table

| Feature | Advantage | Limitation |
|---|---|---|
| **Space** | Fast access | Needs extra storage |
| **INSERT** | Find quickly | Must update index → slow |
| **DELETE** | Removes fast | Must delete index entry → slow |
| **UPDATE** | Search faster | Rebuild index on changes |