📘 Indexing in DBMS

🔹 What is Indexing?

Indexing is a technique used in a database to improve the speed of **data retrieval**. An index is a **small, fast-access data structure** (like a pointer) that maps key values to the location of the corresponding records in the database.

🔹 Why is Indexing Needed?

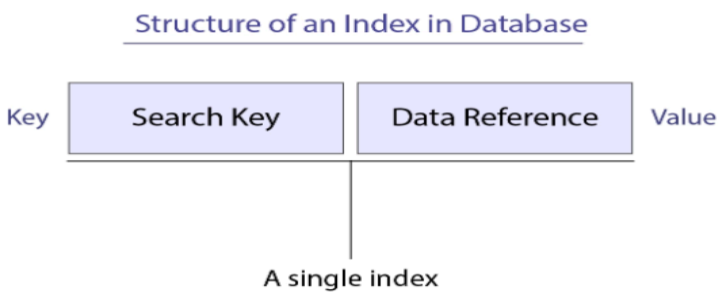
Without indexes:

- The database must **scan the entire table** to find matching rows (called a **full table scan**).
- This is slow, especially for large tables.

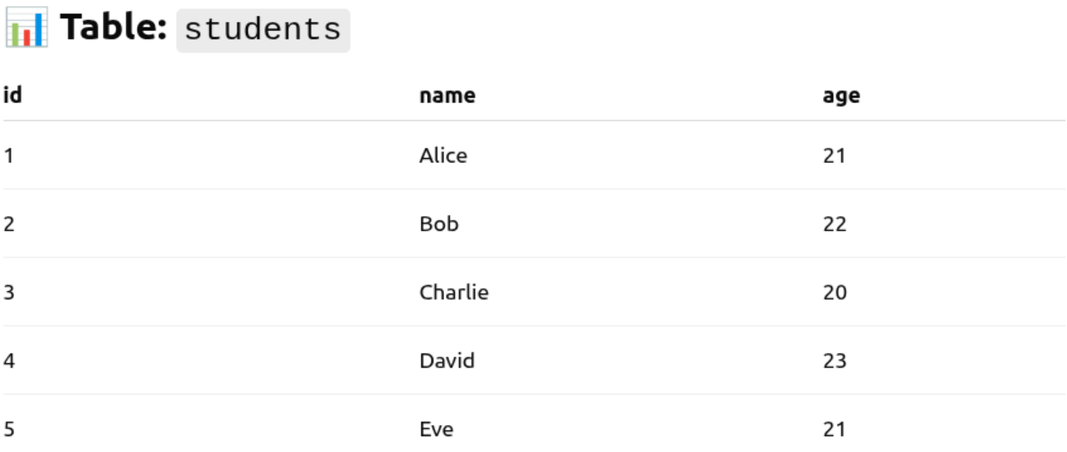
With indexing:

- The database **jumps directly** to the required data using the index — **like finding a topic in a book using the index page**.

🔹 How Indexing Works



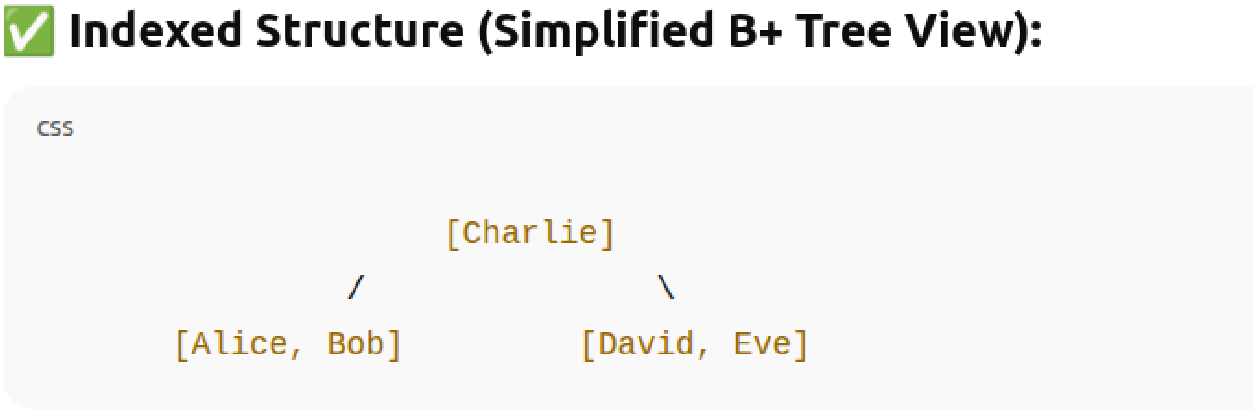
- Most databases use **B-Tree** or **B+ Tree** structures to maintain indexes.
- When a query uses a column with an index (e.g., `WHERE name = 'John'`), the database:
    1. **Looks up the value** in the index.
    2. Gets the **address of the data row**.
    3. Directly retrieves it, skipping unnecessary rows.



📊 **Table:** `students`

id	name	age
1	Alice	21
2	Bob	22
3	Charlie	20
4	David	23
5	Eve	21

CREATE INDEX idx_name ON students(name);



✅ Indexed Structure (Simplified B+ Tree View):

```css
                [Charlie]
             /              \
      [Alice, Bob]        [David, Eve]
```

Query:

SELECT * FROM students WHERE name = 'David';

🔹 Advantages of Indexing

✅ **Faster data retrieval** for SELECT queries

✅ **Improves performance** in joins, searches, sorting, and filtering

✅ **Reduces I/O** by avoiding full table scans

✅ Useful for **enforcing uniqueness** (with unique indexes)

🔹 Disadvantages of Indexing

❌ **Slower write operations** (INSERT, UPDATE, DELETE) due to index updates

❌ **Extra storage** is required to maintain indexes

❌ **Too many indexes** can degrade performance instead of helping

❌ Index maintenance overhead during frequent data changes

🔹 Types of Indexes (in brief)

Type	Description
Primary Index	Created automatically on the primary key
Unique Index	Ensures all values in the column(s) are unique
Composite Index	Index on multiple columns; useful for combined filtering
Clustered Index	Reorders table data to match the index; only one allowed per table
Non-clustered	Stores index separately from table; can have many
Full-text Index	For efficient text search on large text fields
Bitmap Index	Efficient for columns with few distinct values (e.g., gender, status)
Hash Index	Uses hash table internally (good for equality checks, not range queries)

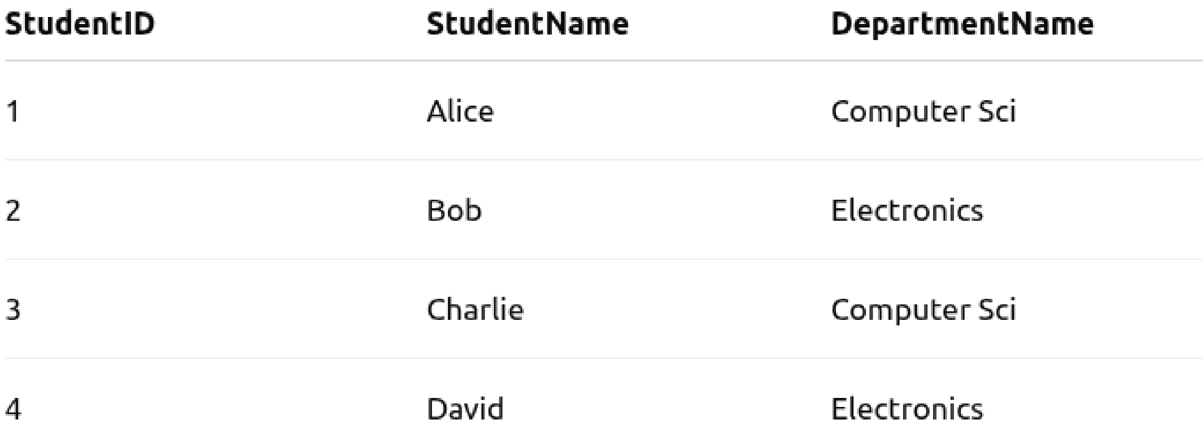
🔹 Example

```
-- Create table
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  department VARCHAR(50),
  salary DECIMAL(10,2)
);

-- Create index on department
CREATE INDEX idx_department ON employees(department);

-- Now this query is faster:
SELECT * FROM employees WHERE department = 'Sales';
```

Anomalies



StudentID	StudentName	DepartmentName
1	Alice	Computer Sci
2	Bob	Electronics
3	Charlie	Computer Sci
4	David	Electronics

✅ Summary

- **Indexing boosts query performance**, especially on large datasets.
- **Use wisely** — too many or wrong indexes can hurt performance.
- Best used on:
  - Frequently searched columns

- Join/filter/sort conditions
- Columns in `WHERE`, `JOIN`, `ORDER BY`