LET'S START WITH DBMS :)

Indexing and its types

Why do we need indexing?

Imagine you have a 1,000-page textbook and you want to serach a topic "xyz".

- When indexing is not present-> You need to manually search for each term in the book, which could be frustrating and time-consuming.

- When indexing is present->The index allows you to quickly locate each topic. You can look up "xyz," find that it's covered on pages 448-490.This makes it more efficient and less time-consuming.

Indexing is a critical technique in database management that significantly improves the performance of query operations by minimizing the no of disk access. Index table is always sorted

LET'S START WITH DBMS :)

Indexing and its types

Why do we need indexing?

- In the same way how we have used indexing in book , it helps us to find specific records or data entries in a large table or dataset.

Search key	Data acccess
It is used to find the record	It contains the address where the data item is stored in memory for the provided search key

Index table

It helps in finding what a user is looking for (P.K OR C.K)

It tell where the record is stored in the memory (set of pointer holding address of block)

LET'S START WITH DBMS :)

Indexing and its types

How it helps?

- **Improved Query Performance:** Indexes allow the database management system (DBMS) to locate and retrieve data much more quickly than it could by scanning the entire table.

- **Indexes can be used to optimize queries that sort data (ORDER BY) or group data (GROUP BY) :** When an index is available on the columns being sorted or grouped, the DBMS can retrieve the data in the desired order directly from the index, eliminating the need for additional sorting operations.

- We have certain indexing methods like Dense(index entry for all the records) and Sparse(index entries for only a subset of records) index.

LET'S START WITH DBMS :)

Indexing and its types

Sparse Index: A sparse index is a type of database indexing technique where the index does not include an entry for every single record in the database. Instead, it contains entries for only some of the records, typically one entry per block (or page) of data in the storage. This makes sparse indexes smaller and faster to search through compared to dense indexes, which have an entry for every record.

It is used when we have an ordered data set.

EX: If a table has 1,000 rows divided into 100 blocks, and you create a sparse index, the index might only have 100 entries, with each entry pointing to the first record in each block.

Index table

Search key	data ref
1	B1
4	B2

Sparse Index
no of records in index file=no of blocks

1 Ram 2 Shyam 3	B1
4 5 6	B2

disk

RollNo	Name
1	Ram
2	Shyam
3	Raghu
4	Riti
5	Raj
6	Rahul

How Sparse Indexing Works:

- **Primary Index:** Sparse indexes are often used with primary indexes, where the table is sorted on the indexed column. The sparse index only includes an entry for the first record in each block or page.

- **Searching:** When searching for a specific value, the database uses the sparse index to quickly locate the block where the record might reside. It then searches within the block to find the exact record.

Disadvantages of Sparse Indexes

- **Additional I/O Operations:** After using the index to find the correct block, an additional search within the block is required to find the specific record.

- **Less Efficient for Random Access:** If queries often need to retrieve random, non-sequential records, a dense index might be more efficient.

Use Cases:

- **Primary Indexing on Large Tables:** Sparse indexes are ideal for primary indexes on large tables where records are sequentially stored.
- **Range Queries:** Sparse indexes can be effective for range queries where the query needs to retrieve a range of records rather than a single record.

Index table

Search key	data ref
1	B1
4	B2

Sparse Index
no of records in index
file=no of blocks

1 2 3	B1
4 5 6	B2

disk

RollNo	Name
1	Ram
2	Shyam
3	Raghu
4	Riti
5	Raj
6	Rahul

LET'S START WITH DBMS :)

Indexing and its types

Dense Index: A dense index is a type of indexing technique used in databases where there is an index entry for every single record in the data file. This means that the index contains all the search keys and corresponding pointers (or addresses) to the actual data records, making it highly efficient for direct lookups.

It is used when we have an un-ordered data set.

EX : Consider a table with 1,000 rows, and you create a dense index on a non-primary key column. The dense index will have no ofunique non key records, each pointing to a specific row in the table.

These are useful in scenarios where random access to individual records is quiet frequent .

Search key	data ref
18	B1
19	B1
20	B2
21	B2

Index table

18 Ram 18. Shyam 19	B1
20 20 21	B2

Dense Index
no of records in index file=no of blocks unique non key records

Age	Name
18	Ram
18	Shyam
19	Raghu
20	Riti
20	Raj
21	Rahul

How Dense Indexing Works:

In a dense index, each index entry includes:

- **A search key** (the value of the indexed column).
- **A pointer** (or address) to the actual record in the data file.

When a query searches for a specific value, the database uses the dense index to quickly locate the corresponding record.

Disadvantages of Dense Indexes

- **Storage Intensive:** Requires significant storage space because it maintains an entry for every record in the table.
- **Maintenance Overhead:** Inserting, updating, or deleting records requires updating the dense index, which can be costly in terms of performance, especially for large tables.

Search key	data ref
18	B1
19	B1
20	B2
21	B2

Index table

18 Ram 18. Shyam 19	B1
20 20 21	B2

Dense Index
no of records in index file=no of blocks unique non key records

Age	Name
18	Ram
18	Shyam
19	Raghu
20	Riti
20	Raj
21	Rahul

Use Cases:

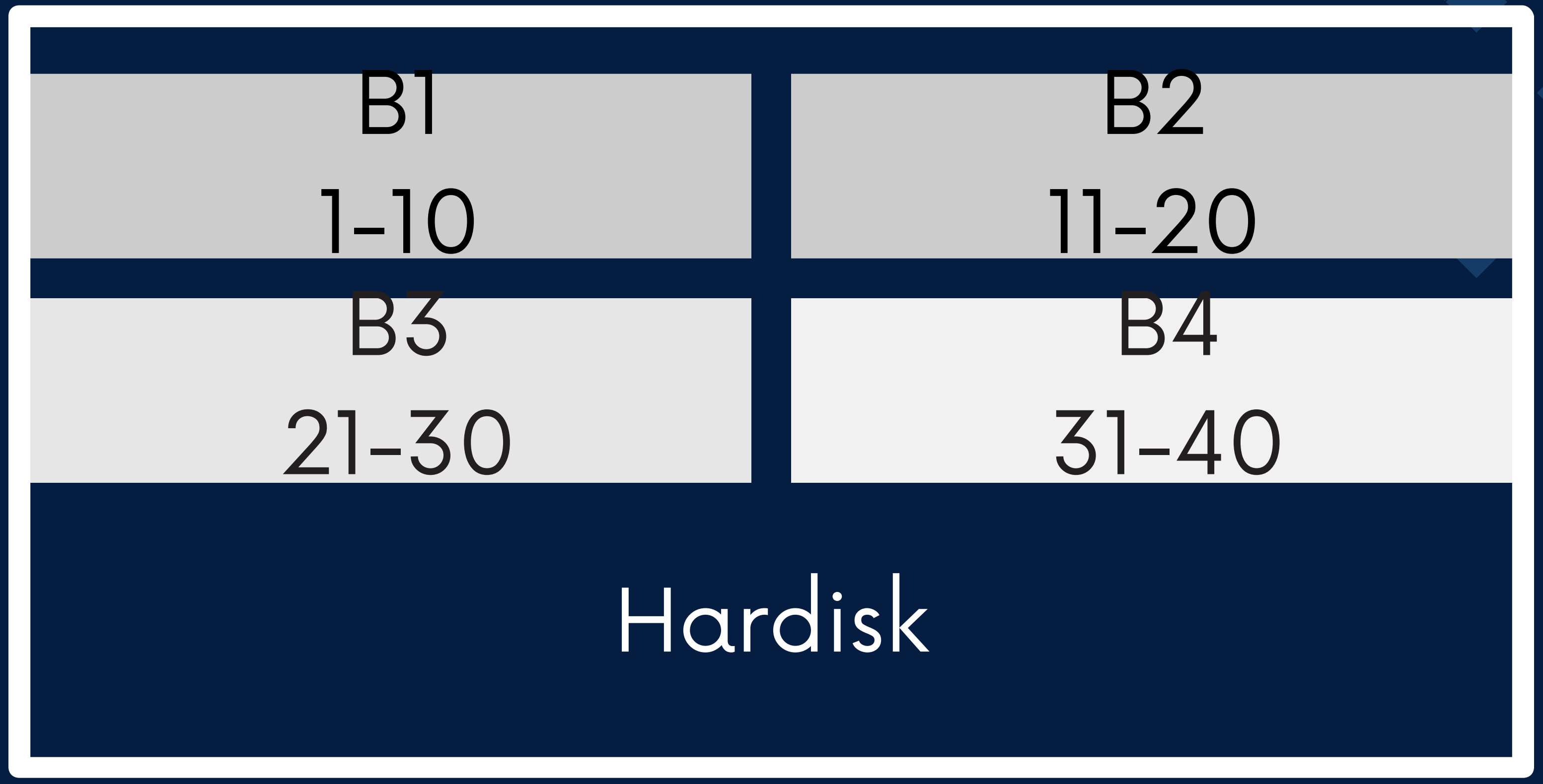
- **Primary Indexing:** Dense indexes are often used for primary indexing when the table is small to medium-sized, and quick access to individual records is required.
- **Exact Match Queries:** Ideal for situations where queries frequently request individual records based on an exact key match.

LET'S START WITH DBMS :)

Indexing and its types

Index table

Search key	data ref
1	B1
11	B2
21	B3



When a query is run, the database engine checks the index to find the pointers to the rows that contain the desired data. It then retrieves these rows directly, without scanning the entire table.

LET'S START WITH DBMS :)

Indexing and its types

How indexing works?

It takes a search key as input and then returns a collection of all the matching records. Now in index there are 2 columns, the first one stores a duplicate of the key attribute/ non-key attribute(serach key) from table while the second one stores pointer which hold the disk block address of the corresponding key-value.

B1
1-10

B2
11-20

LET'S START WITH DBMS :)

Indexing and its types

Indexing

Clustered Index

Primary index

When key= PK
order= sorted
Follows sparse indexing

Clustering Index

When key= non-key
order= sorted
Follows dense indexing

Multilevel Indexing

When there is one more level of indexing in the index table we use inner and outer index.
Mostly used for large files

Non clustered Index

Secondary index

When key= non-key/key
order= unsorted
Follows dense indexing

LET'S START WITH DBMS :)

Indexing and its types

Key data: Unique identifiers for each record. Often used to distinguish one record from another.
Non-key data: All other data in the record besides the key.

- **Single-Level Indexing** is straightforward and works well for smaller databases, offering a direct approach to speeding up query performance.
- **Multi-Level Indexing** is more complex but necessary for larger databases, as it effectively manages large indexes by creating additional layers of indexing, thereby improving data retrieval times.

LET'S START WITH DBMS :)

Primary Index

It enhances the efficiency of retrieving records by using their primary key values. It establishes an index structure that associates primary key values with disk block addresses. This index is composed of a sorted list of primary key values paired with their corresponding disk block pointers, thereby accelerating data retrieval by reducing search time. It is especially beneficial for queries based on primary keys. It is done on sorted data

Search key	data ref
1	B1
4	B2

It creates an index structure that maps primary key values to disk block addresses.
Index table

1 2 3	B1
4 5 6	B2

Sparse Index
no of reocrds in index file=no of blocks

RollNo	Name
1	Ram
2	Shyam
3	Raghu
4	Riti
5	Raj
6	Rahul

LET'S START WITH DBMS :)

Primary Index

Example : In a Users table with a UserID column as the primary key, a primary index on UserID allows for quick retrieval of user records when you know the UserID

The primary index ensures that the database can immediately locate the row associated with a given UserID, making operations like SELECT, UPDATE, and DELETE highly efficient.

LET'S START WITH DBMS :)

Cluster Index

The cluster index is generally on a non- key attribute. The data in the table is typically ordered according to the order defined by the clustered index. Mostly each index entry points directly to a row. It is mostly used where we are using GROUP BY. It physically order records in a table based on the indexed columns.

Search key	data ref
18	B1
19	B1
20	B2
21	B2

Index table

18 **18** **19**	**B1**
20 20 21	B2

Dense Index
no of reocrds in index file=no of blocks unique non key records

Age	Name
18	Ram
18	Shyam
19	Raghu
20	Riti
20	Raj
21	Rahul

LET'S START WITH DBMS :)

<u>Cluster Index</u>

Example : In an Employees table, if you often need to produce lists of employees ordered by LastName, creating a clustered index on LastName can make these queries faster.

The clustered index keeps the rows in LastName order, reducing the need for the database to perform additional sorting when executing queries.

LET'S START WITH DBMS :)

Multilevel Index

A multilevel index is an advanced indexing technique used in database management systems to manage large indexes more efficiently. When a single-level index becomes too large to fit into memory, multilevel indexing helps by breaking down the index into multiple levels, reducing the number of disk I/O operations needed to search through the index

- First Level (Primary Index): The first level is the original index, where each entry points to a block of data or a page in the table.
- Second Level (Secondary Index): If the first level index is too large to fit in memory, a second-level index is created. This index points to blocks of the first-level index, effectively indexing the index itself.

LET'S START WITH DBMS :)

Multilevel Index

Eid	Pointer
E101	
E201	
E301	

Outer index

Eid	Pointer
E101	B1
E151	B2

Eid	Pointer
E201	B3
E251	B4

Eid	Pointer
E301	B5
E351	B6

Inner index

Eid	Data
E101 E102 . .	
E151 E152 . .	
E201 E202 . .	

LET'S START WITH DBMS :)

Secondary Index

Secondary indexing speeds up searches for non-key columns in a database. Unlike primary indexing, which uses the primary key, secondary indexing focuses on other columns. It creates an index that maps column values to the locations where the records are stored, making it faster to find specific records without scanning the entire table.

The data is mostly unsorted and it can be performed on both key and non-key attributes.

LET'S START WITH DBMS :)

Secondary Index

1.when serach key is key attribute and data is unsorted

Dense Index
no of reocrds in index
file=no of blocks unique
non key records

Search key	data ref
18	B1
20	B1
22	B2
24	B2
. . .	. . . .

Index table

Age **18** 20 45 67 68	**B1**
22 24 30	B2

LET'S START WITH DBMS :)

Secondary Index

2. when serach key is non- key attribute and data is unsorted

Dense Index
no of reocrds in index file=no of blocks unique non key records

Search key	data ref
18	B1
20	B1
22	B1->B2
24	B2
. . .	. . . .

Index table

Age **18** **18** 20 67 22	**B1**
22 24 30	B2

LET'S START WITH DBMS :)

Indexing and its types

- **How to create indexes**

Non-Clustered index:

(On one column)

```
CREATE INDEX index_name
ON table_name (column_name);
```

(On multiple column)

```
CREATE INDEX index_name
ON table_name (column_name1, column_name2,.....)
```

Clustered index:

Automatically created with the primary key. MySQL does not support explicit creation of additional clustered indexes.

- **Remove an index**

```
DROP INDEX  index_name ON table_name
```