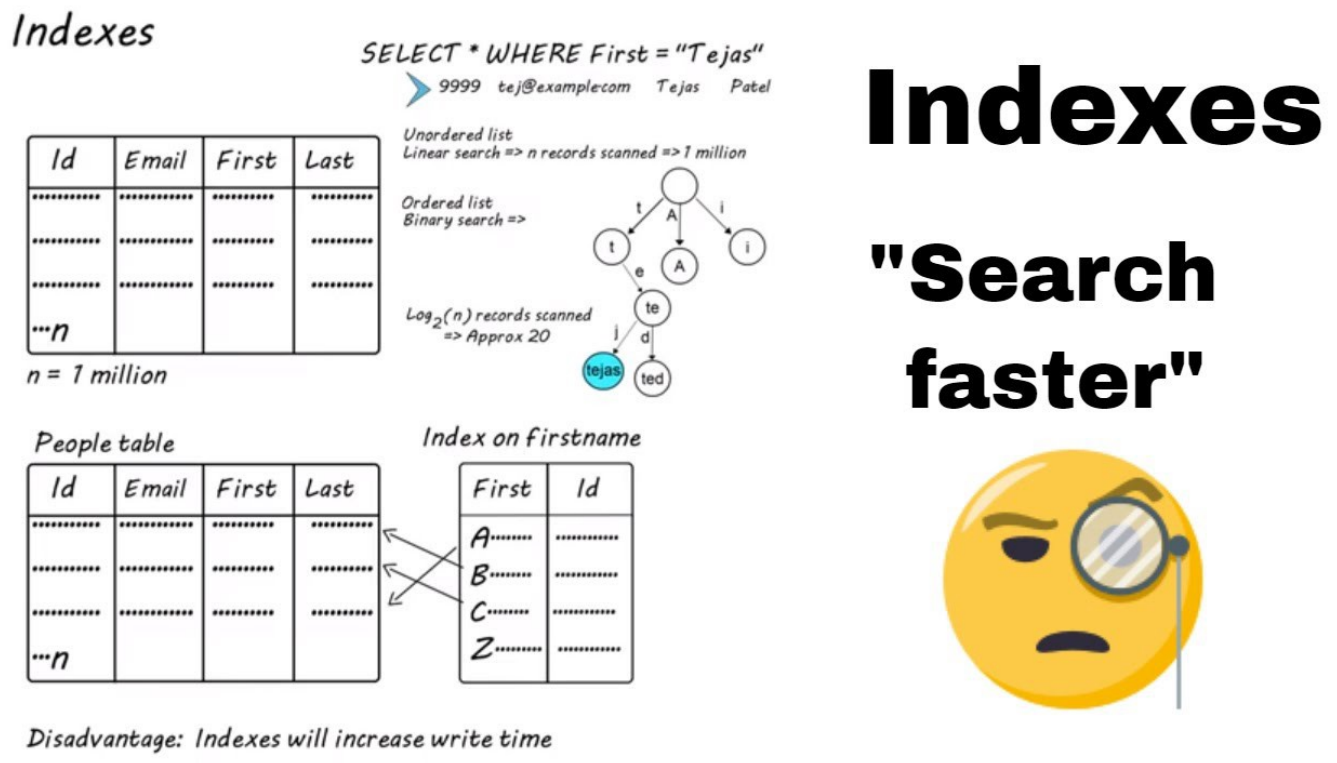
How database indexing works


Indexes
SELECT * WHERE First = "Tejas"
9999 tej@examplecom Tejas Patel
Id Email First Last
...n
n = 1 million
Unordered list
Linear search => n records scanned => 1 million
Ordered list
Binary search =>
Log2(n) records scanned => Approx 20
t A i te tejas ted
People table
Index on firstname
First Id
A B C Z
Disadvantage: Indexes will increase write time
Indexes
"Search faster"

What is database indexing?

Indexing is the process of associating a key with the location of a corresponding data record.

Creating an index on a field in a table creates another data structure that holds the field value, and a pointer to the record it relates to. This index structure is then sorted, allowing Binary Searches to be performed on it.

The downside to indexing is that these indices require additional space on the disk since the indices are stored together in a table using the MyISAM engine, this file can quickly reach the size limits of the underlying file system if many fields within the same table are indexed.

Why do we need indexing?

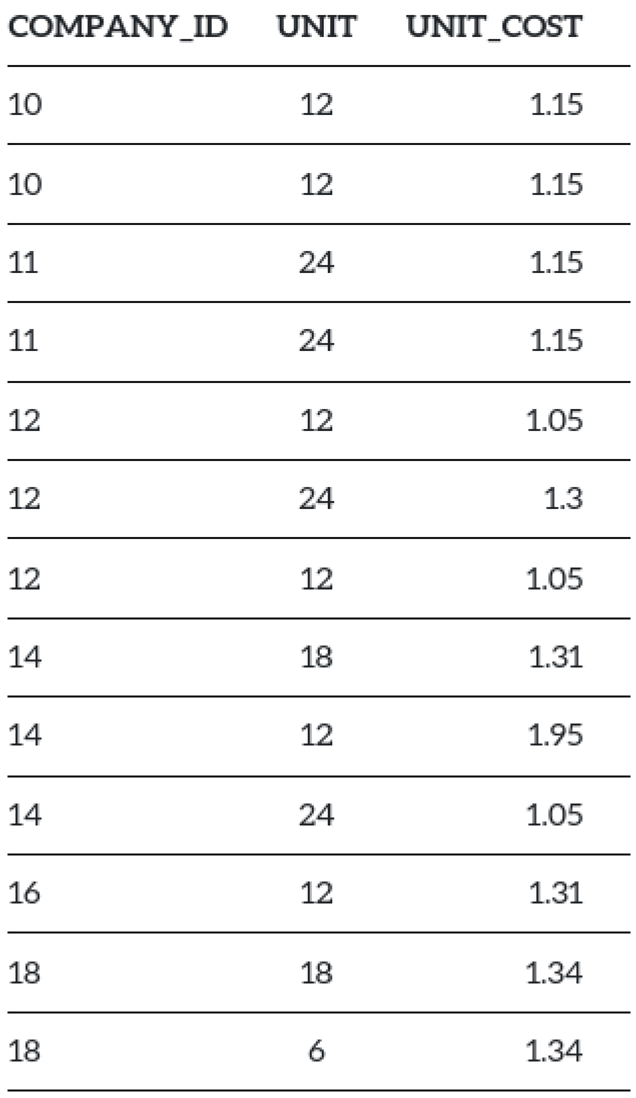
Suppose we have a table called `costing` with three columns company_id, unit, and unit_cost which is completely unordered.

COMPANY_ID	UNIT	UNIT_COST
10	12	1.15
12	12	1.05
14	18	1.31
18	18	1.34
11	24	1.15
16	12	1.31
10	12	1.15
12	24	1.3
18	6	1.34
18	12	1.35
14	12	1.95
21	18	1.36

If we run the query below:

```
SELECT
 company_id,
 units,
 unit_cost
FROM
 costing
WHERE
 company_id = 18
```

The database would have to go through all the records one by one, in the order, they appear to search for the matching datasets. It will become more and more time-consuming once the table size increases.



COMPANY_ID	UNIT	UNIT_COST
10	12	1.15
10	12	1.15
11	24	1.15
11	24	1.15
12	12	1.05
12	24	1.3
12	12	1.05
14	18	1.31
14	12	1.95
14	24	1.05
16	12	1.31
18	18	1.34
18	6	1.34

So here, ***indexing comes into the picture***.

How indexing actually works?

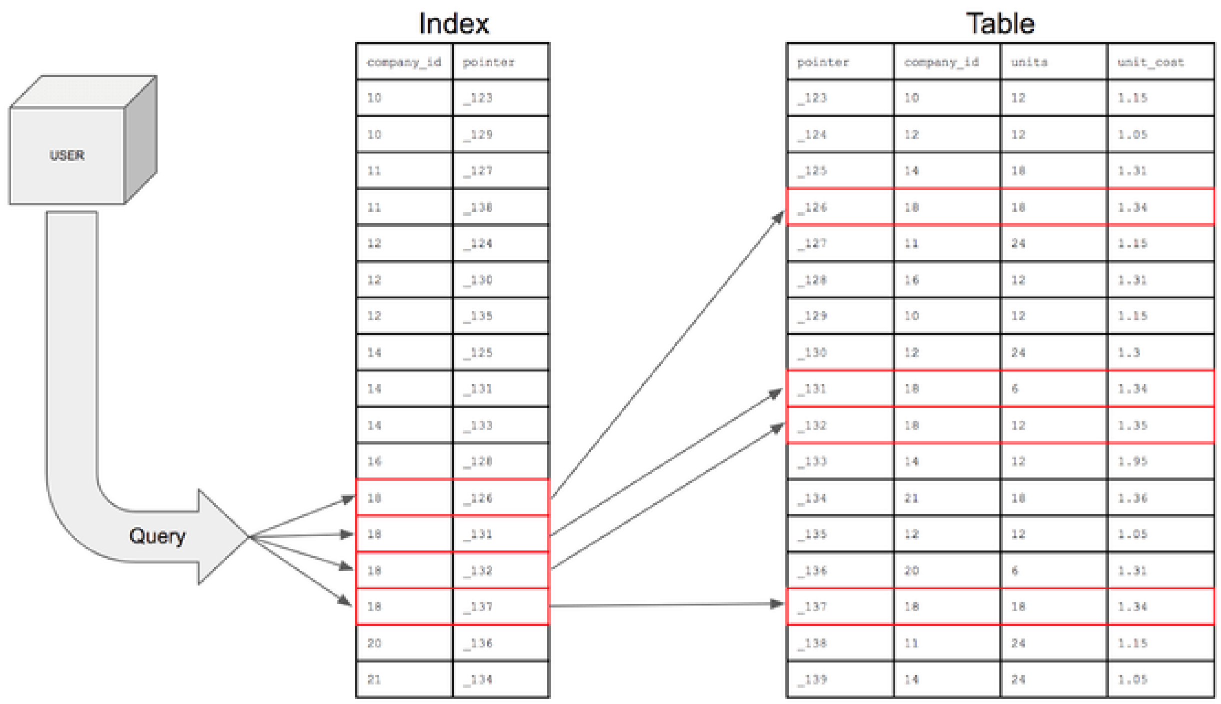
An index is a data structure, (most commonly its B-tree `Its balanced tree, not binary tree` *) that stores the value for a specific column in a table.*

The database table does not reorder itself when we put an index on a column to optimize the query performance.

So the index would look like this -

COMPANY_ID	POINTER
10	_123
10	_129
11	_127
11	_138
12	_124
12	_130
12	_135
14	_125
14	_131
14	_133
16	_128
18	_126
18	_131
18	_132

Visual Representation of the Query execution



Index

company_id	pointer
10	_123
10	_129
11	_127
11	_138
12	_124
12	_130
12	_135
14	_125
14	_131
14	_133
16	_128
18	_126
18	_131
18	_132
18	_137
20	_136
21	_134

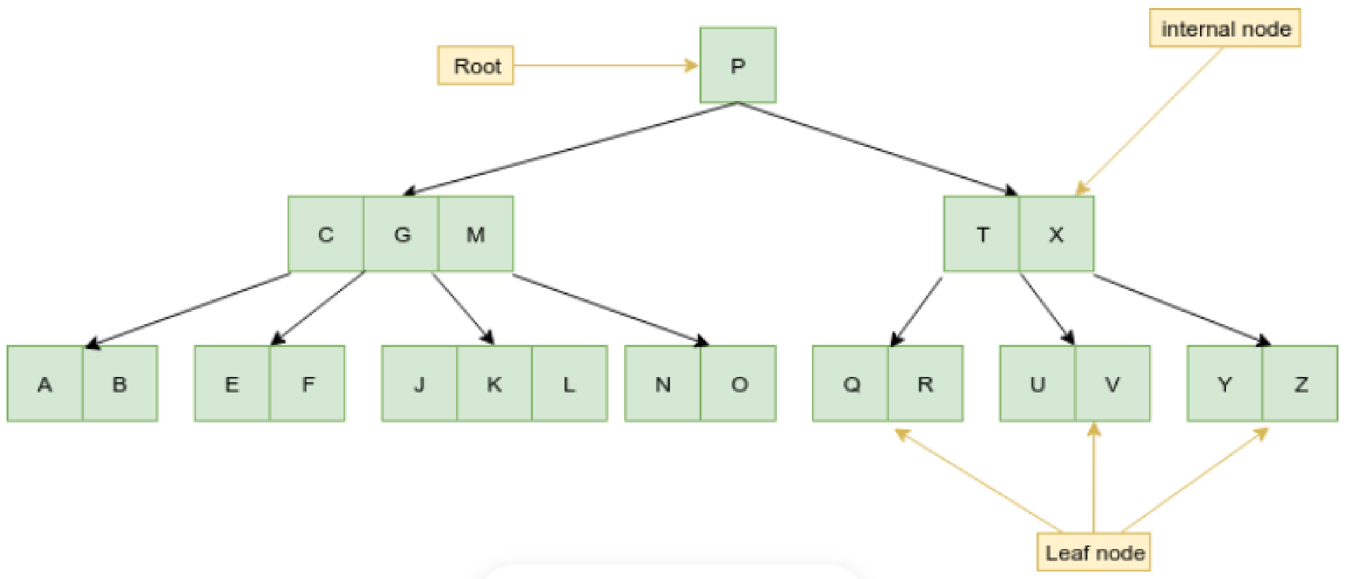
Table

pointer	company_id	units	unit_cost
_123	10	12	1.15
_124	12	12	1.05
_125	14	18	1.31
_126	18	18	1.34
_127	11	24	1.15
_128	16	12	1.31
_129	10	12	1.15
_130	12	24	1.3
_131	18	6	1.34
_132	18	12	1.35
_133	14	12	1.95
_134	21	18	1.36
_135	12	12	1.05
_136	20	6	1.31
_137	18	18	1.34
_138	11	24	1.15
_139	14	24	1.05

So, indexing just cuts down the time complexity from o(n) to o(log n).

More from the software perspective how the BTree works:

B-trees are often the data structure of choice to implement database indexes. A B-tree is a generalization of a binary search tree (BST). A BST is a sorted data structure composed of "nodes" (which hold data) and left and right "pointers" (which direct to other nodes within the BST). A BST has exactly two "children" nodes per node, while a B-tree can have more than two child nodes per node.



But Why only B-Tree?

The main idea of using B-Trees is to reduce the number of disk accesses. Most of the tree operations (search, insert, delete, max, min, ..etc) require O(h) disk accesses where h is the height of the tree. B-tree is a fat tree. The height of B-Trees is kept low by putting the maximum possible keys in a B-Tree node. Generally, the B-Tree node size is kept equal to the disk block size. Since the height of the B-tree is low so total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees like AVL Tree, Red-Black Tree, ..etc.

Properties of B-Tree:

- All leaves are at the same level.
- A B-Tree is defined by the term minimum degree ‘t’. The value of t depends upon disk block size.
- Every node except the root must contain at least t-1 keys. The root may contain a minimum of 1 key.
- All nodes (including root) may contain at most 2t-1 keys.
- The number of children of a node is equal to the number of keys in it plus 1.
- All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2.

- B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink downward.
- Like other balanced Binary Search Trees, the time complexity to search, insert and delete is O(Logn).

Advantages:

- Speed up SELECT query
- Helps to make a row unique or without duplicates(primary,unique)
- If index is set to fill-text index, then we can search against large string values. for example to find a word from a sentence etc.

Disadvantages

- Indexes take additional disk space.
- indexes slow down INSERT,UPDATE and DELETE, but will speed up UPDATE if the WHERE condition has an indexed field. INSERT, UPDATE and DELETE becomes slower because on each operation the indexes must also be updated.

What is the difference between a key and an index in DBMS? Can’t you use the primary key to find any piece of data?

The analogy:

Each library(database) has books (tables). Each book(table) has pages(rows). Each page(row) has a unique page number to identify it (key-value), so you could say that the *key* to finding a certain page is through its number. But if you want to find a certain page, you don’t know exactly where to look for it, right? You’ll have to turn page by page until you reach the page number you were looking for. Wouldn’t it be easier if you had several bookmarks that tell you on which page they were placed? If you look for page 1267 and you have a bookmark on page 1266 then you go straight to that bookmark and now you only have one page to turn to get to your page. The index is like a set of bookmarks for a book. It helps you find the desired rows much faster.