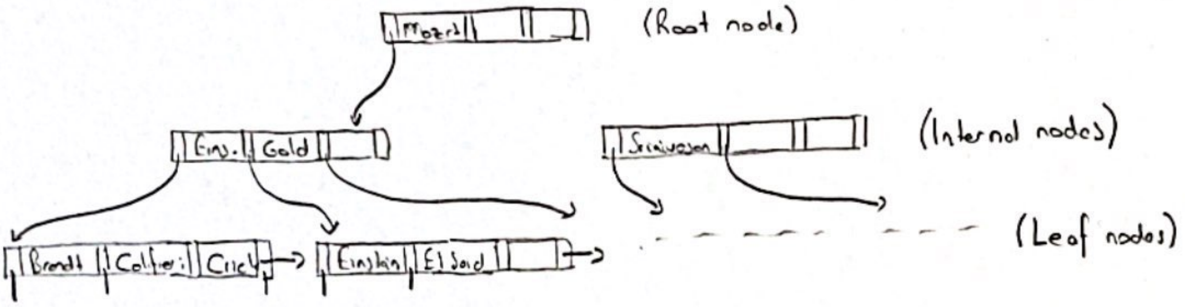
CSE414 - Database
HW#4

Fatih Selim YAKAR
161044054

1) B+ tree is an indexing system that is not in B tree or has been improved by removing bad features. The primary advantage of B+ is that the field holding the pointer to the records is kept only in leaf nodos. This allows the B+ tree to keep the same data in less space, thus providing a much faster system when searching. Below we can see this system for the B+ tree.



Leaf nodes are interconnected, which is to provide sequential access to records. This structure also provides equivalent distant access from root node to leaf node. Each internal node has n/2 to n children. Also a leaf node has (n-1)/2 to n-1 values. Thus searching becomes very easy as all records are stored only in the leaf node and are listed in the ordered linked list. It provides a flexible structure in case the number of registrations increases or decreases. Further branching of internal nodes shortens the tree's height. Therefore it works fine on disk Updates do not affect performance as it is self-balancing tree structure Since the leaf nodes are connected to each other, it is very easy to get records certain range. Insertions and deletion can be handled efficient in leaf. The index can be restructed in short-time However, this structure is less useful for static tables as it is designed for dynamic tables in general. While insertions and deletions are easy at leaf nodes, those operations are difficult because the nodes are not pointed to each other in the interior of the tree. Also, this system brings space overhead

2) The sections containing the entries in hashing are called buckets We can get this entry when the hash function is run using a search key. This hash function is used to adding, deleting and accessing for entry. As a result of the hash function, different keys can be assigned to the same bucket, in this case, a sequential search made in the bucket. In a hash index, buckets store entries with pointers to records and in a hash buckets store records.

2 (continues.)) The hash indexing method is very advantageous for looking spesific point. The result of this search can be obtained directly by a certain search-key to the hash function. But hash indexes are not suitable for range searches in databases, as they do not store keys in any other. Since there is no need for any rearrangement in the hash indexing system, it is advantageous over systems such as B+ tree. However, since the indexes overlapping with the filling of the hash map providing the hash index will increase, the time to find the entry will increase unpredictably accordingly. In this case hash map should be resized. This will add additional costs both in terms of filling the disk and in terms of operation.

3) Since the indexing process creates a link between the data and the determined value, it makes searching indexes much easier and faster. But creating indices for each attribute does not make much sense. If there are really too many and unnecessary indices, the performance improvement is reduced here. It's best to build only for the main attributes that are really needed. Because these added. indexes have additional overheads. These can be summarized. as additional space required in memory, disk and processor time required to add or delete records. For queries with conditions for many search keys, indexes should be used on a few of the keys to ensure efficiency. As for updates, if the index is on a non-primary key, it also needs to be changed on every update. This again adds an overhead.

4) As stated above, indexes provide a link between data and determined value. The feature of the clustering indices mentioned in the question is that it has the same sort order as the relation. It is unlikely that there are 2 clustering indexes on the same relation for the different search keys mentioned in the question. To store the same values together, the records in the relationship must be stored in different order In order to achieve this, it will be necessary to duplicate the relationship twice, that is, to duplicate all values. This situation will not be very logical and efficient in terms of the additional load it will add to the current system.

5)

a) The cost of creating a B+ tree index by adding one record at a time will be as follows under the specified conditions.

. In the worst-case scenario, the leaf nodes are completely half filled

And the split count is calculated as: $2 \times \left(\frac{n_r}{f}\right)$

The insertion cost is the sum of the cost of finding the leaf node's page number (this cost is insignificant for insertion, since non-leaf nodes are already in memory) + the cost of reading disk access after reaching the leaf node + the cost of updating (random disk access) + the cost of writing to page. But in this case if a node split occurs due to insertion, an extra page write cost will be added.

The resul is the total write cost: $\max\left(\underbrace{2\times n_r}_{\text{random disk access}},\ \underbrace{n_r + 2\times\left(\frac{n_r}{f}\right)}_{\text{page writes}}\right)$

b) In this section the cost of writing the page is assumed to be insignificant. Therefore random disk access costs more.

For the result, if we write the values given in the above formula

$(2 \cdot n_r)$ random disk access

$= 2\times 10.000.000 \cdot 10 \cdot \frac{1}{1000}$ sec

$= \underline{200.000 \text{ sec}}$

6)

a) In the relation $R(A,B,C)$, the search keys are $R(A,B)$. To find records between $10 < A < 50$ using the index for n_1 records, the worst case time complexity can be found as:

For each record retrieval, it must traverse the entire tree of height h. So the cost of a single record is $1 * h$. Based on this situation the result for n_1 record is $n_1 * h$

b) the matching tuples between the two cases are the same for the n1 and n2 records. In this case, both cases $10 < A < 50$ and $5 < B < 10$ have the same number of records.

In this case the same worst case time complexity $n_1 * h$ is obtained for finding records in n_2

7)

a)
```
DEFINE TRIGGER insert_branch_cust_depositor
        AFTER INSERT ON depositor
        REFERENCING NEW TABLE AS inserted FOR EACH STATEMENT
        INSERT INTO branch cust SELECT branch_name, customer_name FROM inserted, account
        WHERE inserted.account_number = account.account_number.
```

```
DEFINE TRIGGER insert_branch_cust_account.

        AFTER INSERT ON account
        REFERENCING NEW TABLE AS inserted FOR EACH STATEMENT
        INSERT INTO branch cust SELECT branch_name, customer_name FROM depositor, inserted
        WHERE depositor.account_number = inserted.account_number
```

Similar triggers could be written in delete case, but since the question states that delete does not need to be handled, no additional effort was made for them.

b)
```
CREATE TRIGGER check-delete_account AFTER DELETE ON account
       REFERENCING OLD ROW AS old_row.
       FOR EACH ROW
       DELETE FROM depositor WHERE depositor.customer_name NOT IN
       (SELECT customer_name FROM depositor WHERE account_number <> old_row.account_number)
       END.
```

- SQL databases are called Relational Databases but NoSQL databases called as non-relational or distributed database

- In general, SQL databases can be scaled vertically, meaning that by increasing the RAM CPU or SSD physical hardware, the load on the single server can be increased. But NoSQL databases can scale horizontally, meaning more traffic can be managed by distributing the NoSQL database or adding more servers

- NoSQL database has dynamic schema for unstructured data. Data can be stored by organizing as document-oriented, column-oriented, graph-based or KeyValue storage. By contrast, SQL databases use a data-driven structured query language.. SQL is one of the most versatile and widely used options available, making it safe choice., especially for large complex queries. But on the other hand, it can be restrictive. SQL requires you to use predefined schemas to. determine the structure of your data before working with data. Also, all your data must have the same structure, which means a change in structure will be difficult for entire system

- While SQL databases contain ACID properties (Atomicity, Consistency, Isolation and Durability), the NoSQL database contains the Brewers CAP theorem (Consistency, Availability and Quotient tolerance)

- NoSQL databases are key-values, document-based, graph databases or large column stores, but SQL databases are table-based.

- SQL databases are not suited for hierarchied data storage but NoSQL databases are best suited for hierarchied data storage.

- Some examples of SQL databases include PostreSQL, MySQL Oracle, and Microsoft-SQL Server. Examples of NoSQL databases include Redis, RavenDB Cassandra, MongoDB, BigTable, HBase, Neo4j and CouchDB.

About NoSQL

The datastructures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL. A NoSQL database includes ease of design, simpler horizontal scaling to machine clusters, and more precise. control availability. The datastructures used by NoSQL databases are also. sometimes seen as more flexible than. relational database tables. But the suitability of a particular. NoSQL database. depends on the problem it has to solve.