Gökhan HAS
161044067

CSE 414
ASSIGNMENT 04

(1) First, I can actually understand what we use B+ trees for by looking at the definition. The B+ tree is a tree structure used for indexing purposes, which is frequently preferred to access the data in a fast and efficient way when adding new data to the data in the sorted state, when decrementing from this data or when we just want to access the data. Indexing is done by selecting a value (called a key) in each data that is unique only to that data.

If I come to the advantages of using databases, the first thing I remember is to access the data quickly. Speed is important in databases. The user should not be too angry. It is then possible to keep the records on an equal number of discs. Being able to be used for key indexing is a big plus for database management systems. By the way, it is fast because of the leaf nodes used in B+ trees. It also allows sequential or non-sequential access. Reorganization of entire file is not required to maintain performance and used extensively.

The first drawback is of course that is relatively difficult to delete as it is kept in knot form. Of course, this depends on which leaves are to be deleted. It can be complicated if deleted from the inside. Leaf nodes cannot be connected. And since these nodes are of different size compared to other nodes, they are more difficult to store. Especially extra insertion and deletion overhead, space overhead.

Gökhan HAS
161220067

② When we search for a value in the database, all the records are looked at one by one in order to find the relevant values. But if the number of records is in the millions, then looking at the whole table for the sought value will delay the result is too much. We create indexes to prevent the entire table from being scanned in queries.

I can say that it has a positive effect on performace, which I can count as an advantage and uses less storage then its counterparts. Special functions (Hash) are used especially for adding, deleting and accessing. The disadvantages include the inability to query with a partial key, the inability to use it to avoid sorting, the significant computation time to search at each node, and the inability to use indexed data while the database is operating.

③ Simply put, it will be necessary to increase the storage for every feature we hold. Increasing storage also means paging more. Let's say we are indexing these extra directories. This indexing process will use the resources of the computer or device. Excessive use of these resources can both reduce performace and cause an irreversible event. And most importatly, I think, the values here may need to be updated as they are not the actual values. This update process also means extra cost. Instead of using all of them, this process can be done for some features (provided that they are not too many) and performace can be improved.

Gökhan HAS
161044017

④ I believe that this approach is not possible. Normally an index is bound by a property. Connecting it with other extra features will reduce its performance. If we want to store the same values, they will need to be used in a diffeent order. One of our purposes for using index is to speed up the search anyway. However, if the mentioned event is done, the relationships will be stored extra at least twice. Extra cost will be created. Because of this situation, this approach is not possible.

⑤ a) Non-leaf nodes are not included when calculating the cost. There is a cost to find the requied leaf node. There is a disk access cost. There is also the processing cost of writing the necessary update after accessing it. Adding up these costs gives the overall result.

In the worst-case, the leaf nodes are completely half filled and the split count is calculated as $2 \times \frac{nr}{f}$.

We can think all scenarios, So, the total write cost ise $\max(2 \times nr, nr + 2 \times (\frac{nr}{f}))$

$2 \times nr$: random disk access

$nr + 2 \times (\frac{nr}{f})$: page writes

b) The cost of writing the page is assumed to be insignificant. So, random disk access costs more. Therefore, if we write the values given in the above formula $(2 \times nr)$; random disk access

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

$$= 200.000 \text{ sec}$$

Gökhan HAS
1610404067

(6) a) If (A, B) is ordered by A, and the data of the same value of A is ordered by B. Then, for each record retrivial, all must traverse the whole tree of height h. And then the cost of a one record is 1*h. Based on this, the result for n_1 record is $n_1 * h$. R(A, B, C) and the search keys are R(A, B)

b) In this question, the matching tuples between the two conditions are same for n_1 and n_2 records. So, the same number of records.

$10 < A < 50$
$5 < B < 10$
} In this case, the same worst-case time complexity $n_1 * h$ is obtained for finding records.

(7) a)
```
DEFINE TRIGGER insert_branch_customer_depositor
    AFTER INSERT ON depositor
    inserted FOR EACH STATEMENT
    INSERT INTO branch_cust SELECT brach_name, custom_name
    FROM inserted, account WHERE inserted.account-number = account.account_
                                                           number
```

```
DEFINE TRIGGER insert_brach_customer_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
```

* Deletes operations are not necessary in this questio.

b)
```
CREATE TRIGGER check-delete_trigger
    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)
```

Göktan HAS
161044067

→ NoSQL:

(8) • It works like hardware independent. That means it will run on many processors.

• There is only one way to store and retrieve data.

• If you add more processors, you get a consistent increase in performance.

• When you use NOSQL, the cost of hardware will be cheaper.

• It's free of joins. You can use your data using simple interfaces without joins.

• These systems provide ease of importing data from many formats.

→ The difference between SQL and NoSQL:

SQL	NOSQL
• relational	non-relational
• predefined schema	dynamic schema
• vertically scalable	horizontally scalable
• better for multi-row transactions	better for unstructured data

• SQL databases are meaning that by increasing the RAM, CPU or SSD. But NOSQL databases are meaning more traffic can be managed by distributing the NoSQL database or adding more servers.

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

(6)