assignment 4 , ahmed semih özmekik
171044039

1. what are the adventages and disadvantages of B+tree for using for indexing in databases?

B-tree is a self-balancing search tree and, B+tree is a B-tree with an additional level at the bottom with linked leaves and each node containing only keys. (not key-value pairs)
use of this data structure we known of, we are getting some pros and cons; here are some adventages.

- Because all data are saved in the leaf node and are ordered sequentially in a linked-list, searching is very easy.
- Any insert, delete or update has no effect on the performance because it is a balancing tree strutre.
- Because all of the data is stored in the leaf nodes, and there is more branching of interior nodes, the tree's height is reduced. Disk I\O is reduced as a result. Hence, it functions effectively on secondary storage devices.
- retrieved can be obtained in range or partial, made faster by traversing across the tree strutre.

Although it is used common, due to having more pros than cons, here are some of it's disadvantages:

- additional overhead for insertion and deletion, and space overhead.
- for static tables, it is less efficient.

2. what are the advantages and disadvantages of hashing for using it for indexing in databases?

advantages:

- point lookups work fastest with hash indexes. (if we are to implement DBMS, and our need is primarly focuses on lookups, we should pick these, beacuse it's O(1), and is best.)
- for equality testing, they are faster than B+ trees.

disadvantages:

- they are not ideal for range searches. Since the keys are not stored in any order.
- Because the lookup was based on hash computation of the key, and hence requires the complete precise key, they are not suited for prefix matching of the keys.

3. Indices speed query processing, but it is usually bad idea to create indices on every attribute, and every combination of attributes, that are potential search keys. explain why.

Here are many reasons for not keeping indices on every attribute.

- first thing that comes to mind, extra indexes necessitates more storage space and, cause space overhead.
- More indexes necessitates more CPU time, as well as disk I/O. that will cause computation overhead.
- non-primary key indexes may need to be hung on updates, but primary key indexes may not.

4- Is it possible in general to have two clustering indices on the same relation for different search keys? explain your answer.

no, it is not possible. Two clustering indices for distinct keys on the same relation are not conceivable since the tuples in the relation would have to be stored in a distinct order to get the same values saved together. We could do this by replicating every values and saving the relation twice.

5-a Because the non-leaf nodes are in memory, the cost of locating the page number of the required leaf page for an insertion is low. On the leaf level, one random disk access is required to read and one random disk access to update it. In addition to the cost of writing one page. Insertions that result in the splitting of leaf nodes necessitate a second page write.

As a result, constructing a tree with x elements requires no more than 2x random disk reads and x+2(x/f) page writes. The second component of the cost derives from the fact that, in worst-case scenario each leaf is half-filled, resulting in twice as many splits than x/f. Because we assume non-leaf nodes are in memory, the above formula overlooks the cost of writing them, though they would be written eventually. The cost is quite close to 2(x/f)/f, which is the number of internal nodes immediately above the leaf; we can add more terms to account for higher layers of nodes.

5-b. Because each insertion costs 20ms, substituting the numbers in the preceding calculation and ignoring the cost of page writes takes around 10,000,000 X 20ms.

6.a According to the definition, the (composite key (A, B)) is ordered key A, while data with the same A value is ordered by B. As an example,

$$(a_i, b_i) < (a_j, b_j) \iff (a_i, b_i) \vee (a_i = a_j \wedge b_i < b_j)$$

given range for search, satisfying the records is 101 leafs matching tuples. In the worst case scenario, where the leaf is one, matched tuples are kept in n_1 leaves. As a result, the worst case cost is $h + n_1$. (where h is the height and n_1 is the leaves)

6.b. In the worst case situation, the cost $h + n_1$ will be the same as in the best case scenario. In this case, n_2 or the number of matching tuples has no influence on the final cost. As a result, the worst-case cost is $h + n_1$.

7-a We must establish a database trigger on an insert into depositor and account, before putting into materialized view branch cust. For rule execution, we assume that the database system employs instantaneous updating. Assume that a represent version of a relation is indicated by the relation name alone, and that set of newly added tuples is represented by qualifying the relation name with prefix.

```
DEFINE TRIGGER insert_to_branch_cust_depositor
AFTER INSERT ON DEPOSITOR
table as inserted for each
insert into branch_cust
    select branch_name, customer name
    from inserted, account
    where inserted.account_number = account.account_number

DEFINE TRIGGER insert_to_branch-cust_account
AFTER INSERT ON ACCOUNT
table as inserted for each
insert into branch_cust
    select branch_name, customer_name
    from depositor, inserted
    where depositor.account_number = inserted.account_number;
```

7.b CREATE TRIGGER Check_loan_delete 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

8. Give characteristics of NoSQL. What is the difference between SQL and NoSQL?

NoSQL databases are not tabular and store data in ways that relational tables do not. NoSQL databases are classified into several categories based on their data model. Document, key-value wide-column, and graph are the most common. They offer flexible schemas and can quickly grow with large volumes of data and significant use demands.

SQL	NoSQL
Relational	not related
Specified schema and employ query language	Dynamic schemas
Scale vertically	Scale horizontally
Table-based	not table based