CSE 414 - Database
Asignment 4

Şeyda Nur DEMİR
12 10 44 042

1. What are the advantages and disadvantages of B^+-tree for using it for indexing in databases?

- Here is some description from our text book and lecture slides.

Indexing mechanisms used to speed up access to desired data.

search key attribute to set of attributes used to look up records in a file.

index file consists of records (called **index entries**) of the form

search-key	pointer

index files are typically much smaller than the original file

two basic kinds of indices

↳ **ordered indices** search keys are stored in sorted order

↳ **hash indices** search keys are distributed uniformly across buckets using hash function.

Some other terms:

clustering index, primary index, secondary index, nonclustering index, index-sequential file, dence index, sparse index, multilevel index, outer index, inner index

B^+-tree is a rooted tree satisfying the following properties:
all paths from root to leaf are of the same length, each node that is not a root or a leaf has between n/2 and n children, a leaf node has between (n-1)/2 and (n-1) values. Special cases: if the root is not a leaf, it has at least 2 children, if the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n-1) values.

disadvantage of indexed-sequential files

- performance degrades as file grows, since many overflow blocks get created
- periodic reorganization of entire file is required

advantage of B^+-tree index files

- automatically reorganizes itself with small, local, changes in the face of insertions/deletions
- reorganization of entire file is not required to maintain performance

minor disadvantage of B^+-trees

- extra insertion and deletion overhead, space overhead.

advantages of B^+-trees outweigh disadvantages

- B^+-trees are used extensively

comparison 1&2
at page 3. ↪

CSE 414- Database
Assignment 4

Şeyda Nur Demir
12 10 44 042

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

- Here is some description from our text book and lecture slides.

Indexing mechanisms used to speed up access to desired data.

Search key attribute to set of attributes used to look up records in a file

index file consist of records (called **index entries**) of the form.

search key	pointer

Index files are typically much smaller than the original file

two basic kinds of indices

↳ **ordered indices** search keys are stored in sorted order.

↳ **~~ordered~~ hash indices** search keys are distributed uniformly across buckets using hash function

some other terms :

clustering index, primary index, secondary index, nonclustering index,

index-sequential file, dense index, sparse index, multilevel index, outer index, inner index

bucket is a unit of storage containing one or more entries (typically a disk block)

hash function using for obtain the bucket of an entry from its search-key value.

Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B, hash function is used to locate entries for access, insertion as well as deletion, entries with different search-key values may be mapped to the same bucket, thus entire bucket has to be searched sequentially to locate an entry

hash index, bucket store entries with pointers to records.

hash file-organization, buckets store records.

handling bucket overflow, overflow buckets, overflow chaining,

close addressing (closed hashing/open hashing), open addressing (open hashing/closed hashing),

periodic rehashing if number of entries in a hash table becomes (say) 1.5 times size of hash table, create new hash table of size (say) 2 times the size of the previous hash table, rehash all entries to new table.

liner hashing do rehashing in an incremental manner

extendable hashing tailored to disk based hashing, with buckets shared by multiple hash values, doubling of # of entries in hash table, without doubling # of buckets.

advantage of hash over B^+-tree

- Some databases use Btrees, but index retrieval is more efficient from Btree in hash.

Comparison 1&2
at page 3. ↪

CSE 414 - Database
Assignment 4

Şeyda Nur DEMİR
12 10 414 042

(1 and 2) Comparison of B+ tree and Hash

B^+ tree	Hash
• harder to implement	• easier to implement
• complexity is O(logn)	• complexity is O(1)
• retrieval has less efficiency	• retrieval efficiency is very high
• it is sorted, when find it stops	• it needs full table scan
• can be used for column comparisons '=' '>' '>=' '<' "<=" or BETWEEN	• can be used for only equality or inequality checks "=" '<=' ">=" but faster
• can be able to produce ordered results without sort step	• it must sort values to produce ordered values.
• it needs to be updated after every insertion and deletion.	• it does not need to be update after any operation.
• for repeated key values it is more efficient	• for repeated key values, it has collision problems and it is not efficient
• better at range queries.	• better at equility queries
	• with good hash function and enough hash buckets, we can say it is better otherwise, it will tend towards O(n) complexity.

CSE 414 - Database
Assignment 4

Şeyda Nur DEMİR
12 10 441 042

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

- Here is some description from our text book.
  Reasons for not keeping indices on every attribute include:
- Every index requires additional CPU time and disk I/O overhead during inserts and deletions.
- Indices on non-primary keys might have to be changed on updates, although an index on the primary key might not.
  (this is because updates typically do not modify the primary-key attributes.)
- Each extra index requires additional storage space.
- For queries which involve conditions on several search keys, efficiency might not be bad even if only some of the keys have indices on them. Therefore, database performance is improved less by adding indices when many indices already exist.

★ An index offers link to data with a specified value.
Indices make search faster.

★ If too many indices exist in the database,
then the performance improvement is not much.

CSE 414 - Database
Assignment 4

Şeyda Nur DEMİR
121044042

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

- Here is some description from our text book.

In general, it is not possible to have two primary indices on the same relation for different keys because the tuples in a relation would have to be stored in different order to have the same values stored together. We could accomplish this by storing the relation twice and duplicating all values, but for a centralized system, this is not efficient.

* An index offers link to data with a specified value.
Indices makes search faster.
A clustering index has the same sort order as that of the relation.

* Generally, it is impossible.
Obviously, it is not an efficient approach for a centralized system.

CSE 414-Database
Assignment 4

Şeyda Nur DEMİR
12 10 44 042

5. Suppose you have a relation r with n_r tuples on which a secondary B^+-tree is to be constricted.

a. Give a formula for the cost of building the B^+-tree index by inserting one record at a time. Assume each block will hold an average of f entries and that all levels of the tree above the leaf are in memory.

b. Assuming a random disk access takes 10 miliseconds, what is the cost of index construction on a relation with 10 million records?

- The cost to locate the page number of the required leaf page for an insertion is negligible since the non-leaf nodes are in memory. On the leaf level it takes one random disk access to read and one random disk access to update it along with the cost to write one page. Insertions which lead to splitting of leaf nodes require an additional page write. Hence to build a B^+-tree with (n_r) entries it takes a maxmum of $(2 * n_r)$ random disk accesses and $(n_r + 2 * (n_r/f))$ page writes. The second part of the cost comes from the fact that in the worst case each leaf is half filled, so the number of splits that occur is twice (n_r/f). This formula ignores the cost of writing non-leaf nodes, since we assume they are in memory, but in reality they would also be written eventually. This cost is closely appraximated by $(2 * (n_r/f)/f)$, which is the number of internal nodes just above the leaf, we can add further terms to account for higher levels of nodes, but these are much smaller than the number of leaves and can be ignored.

* Entries : n_r
* Number of splits : $(2 * (n_r/f))$
* Random Disk Access : $(2 * n_r)$
* Page Writes : $(n_r + (2 * (n_r/f)))$

- Substituting the values in the above formula and neglecting the cost for page writes, it takes 10M * 20ms ≅ 56 hours, since each insertion costs 20ms.

* Random Disk Access : $(2 * n_r)$
2 * 10M * 10ms = 200ks. ≅ 55,5 h ≅ 56hours.

CSE 414 - Database
Assignment 4

Şeyda Nur DemiR
12 10 44 042

6. Suppose there is a relation $r(A,B,C)$ with a B^+-tree index with search key (A,B).

a. What is the worst-case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?

b. What is the worst-case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above?

- For the wost-case, it traverse whole tree of height h
  hence the cost of the sigle record is $(1 * h)$
  and the cost of the n_1 records is $(n_1 * h)$

  For this case too, we find the some number of records.
  Because the matching tuples between the two conditions are same for n_1 and n_2.
  then the cost of the n_2 records is $(n_1 * h)$ too.

CSE 414- Database
Assignment 4

Şeyda Nur DEMIR
12 10 44 042

7. Given the relations listed below, (...)
Let us define a view branch_cust as follows: (...)

a. Suppose that the view is materialized, that is, the view is computed and stored. Write triggers to maintain the view; that is, to keep it up-to-date on insertions to depositor or account. It is not necessary to handle deletions or updates. Note that, for simplicity, we have not required the elimination of duplicates.

b. Write an SQL trigger to carry out the following action: on delete of an account, for each customer-owner of the account, check if the owner has any remaining accounts, and if she does not, delete her from the depositor relation.

-
```
create trigger trigger1
after insert on depositor
referencing new row as inserted
for each row
insert into branch_cust
        select branch_name, inserted.customer_name
        from account
        where inserted.account_number = account.account_number
```

```
create trigger trigger2
after insert on account
referencing new row as inserted
for each statement
insert into branch_cust
        select inserted.branch_name, customer_name
        from depositor
        where depositor.account_number = inserted.account_number
```

```
create trigger trigger3
after delete on account
referencing old row as drow
for each row
delete from depositor
        where depositor.customer_name not in
        (select customer_name from depositor where account_number <> drow.account_number)
        end
```

CSE 414 – Database
Assignment 4

Şeyda Nur DEMIR
12 10 44 042

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

- Characteristics of NoSQL:

✓ NoSQL systems store and retrieve the data from many formats: key-value stores, graph databases, column-family (Big table) stores, document stores, and even rows in tables

✓ Allows us to extract our data using simple interfaces without joins.

✓ Allows us to drag-and-drop your data into a folder and then query it without creating an ER model.

✓ Allows us to store our database on multiple processors and maintain high-speed performance.

✓ Most (but not all) ones of them leverage low-cost commodity processors that have separate RAM and disk.

✓ When we add more processors, we get consistent increase in performance.

✓ Offers us options to a single way of storing, retrieving, and manipulating data.

Differences between NoSQL and SQL:

✓ SQL → relational NoSQL → not relational
✓ SQL → vertically scalable NoSQL → horizontally scalable
✓ SQL → table based NoSQL → key-value, graph, column-family, document.
✓ SQL → better for multi-row transactions NoSQL → better for unstructured data
✓ SQL → structured query language NoSQL → unstructured data
✓ SQL → predefined schema NoSQL → dynamic schema