Hệ Cơ Sở Dữ Liệu

Bài tập chapter 2

Nhóm: 3.1

Sinh viên: Nguyễn Đình Khoa

MSSV: 1833051

1.

Characteristics	Primary index	Clustering index	Secondary index
Indexing field	Ordered Key field	Ordered Non-key field	Not ordered field
Number of index entries	Number of file blocks	Number of distinct value	Number of distinct value / Number of file records
Dense or sparse	Sparse	Sparse	Dense
Block anchor	Yes	Yes/No	No
Implementation options	No	No	Yes
Number of block accesses if search for 1 record with “=” on indexing field	Log2(bi) + 1	Log2(bi) + 1	Log2(bi) + 1 + 1
Number of block accesses if search for multiple records with “=” on indexing field			
Number of block accesses if search for multiple records with “<”, “>”, on indexing field			

2. .

	B - Tree	B+ - Tree
	This is a binary tree structure similar to B+ tree. But here each node will have only two branches and each node will have some records. Hence here no need to traverse till leaf node to get the data. B tree is an organizational structure for information storage and retrieval in the form of a tree in which all terminal nodes are at the same distance from the base, and all non-terminal nodes have between n and 2 n sub-trees or pointers (where n is an integer).	This is a balanced tree with intermediary nodes and leaf nodes. Intermediary nodes contain only pointers/address to the leaf nodes. All leaf nodes will have records and all are at same distance from the root. B+ tree is an n-array tree with a variable but often large number of children per node. B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.
	It has more height compared to width.	Most width is more compared to height.
	Number of nodes at any intermediary level 'l' is 2 l. Each of the intermediary nodes will have only 2 sub nodes.	Each intermediary node can have n/2 to n children. Only root node will have 2 children.
	Even a leaf node level will have 2 l nodes. Hence total nodes in the B Tree are $2^{l+1}-1$.	Leaf node stores (n-1)/2 to n-1 values
	It might have fewer nodes compared to B+ tree as each node will have data.	Automatically Adjust the nodes to fit the new record. Similarly it re-organizes the nodes in the case of delete, if required. Hence it does not alter the definition of B+ tree.

	Since each node has record, there might not be required to traverse till leaf node.	Reorganization of the nodes does not affect the performance of the file. This is because, even after the rearrangement all the records are still found in leaf nodes and are all at equidistance. There is no change in distance of records from neither root nor the time to traverse till leaf node.
	If the tree is very big, then we have to traverse through most of the nodes to get the records. Only few records can be fetched at the intermediary nodes or near to the root. Hence this method might be slower.	If there is any rearrangement of nodes while insertion or deletion, then it would be an overhead. It takes little effort, time and space. But this disadvantage can be ignored compared to the speed of traversal
Storage	In a B tree, search keys and data stored in internal or leaf nodes.	In a B+ tree, data stored only in leaf nodes.
Data	The leaf nodes of the three store pointers to records rather than actual records.	The leaf nodes of the tree stores the actual record rather than pointers to records.
Space	These trees waste space	There trees do not waste space.
Function of leaf nodes	In B tree, the leaf node cannot store using linked list.	In B+ tree, leaf node data are ordered in a sequential linked list.
Searching	Here, searching becomes difficult in B- tree as data cannot be found in the leaf node.	Here, searching of any data in a B+ tree is very easy because all data is found in leaf nodes.
Search accessibility	Here in B tree the search is not that easy as compared to a B+ tree.	Here in B+ tree the searching becomes easy.
Redundant key	They do not store redundant search key.	They store redundant search key.

Applications	They are an older version and are not that advantageous as compared to the B+ trees.	Many database system implementers prefer the structural simplicity of a B+ tree.

3. Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 7 bytes long. A file has r = 30,000 EMPLOYEE records of fixed length. Each record has the following fields: Name (30 bytes), Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

a. Calculate the record size R in bytes.

b. Calculate the blocking factor bfr and the number of file blocks b, assuming an unspanned organization.

c. Suppose that the file is ordered by the key field Ssn and we want to construct a primary index on Ssn. How many block accesses via index are needed to retrieve a record with a condition “=“ on Ssn?

d. Suppose that the file is ordered by the nonkey field Department_code and we want to construct a clustering index on Department_code that uses block anchors (every new value of Department_code starts at the beginning of a new block). Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. How many block accesses via index are needed to retrieve a record with a condition “=“ on Department_code? How many block accesses via index are needed to retrieve all the records with a condition “=“ on Department_code?

e. Suppose that the file is not ordered by the key field Ssn and we want to construct a secondary index on Ssn. How many block accesses via index are needed to retrieve a record with a condition “=“ on Ssn?

f. Suppose that the file is not ordered by the nonkey field Department_code and we want to construct a secondary index on Department_code, using option 3, with an extra level of indirection that stores record pointers. Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. How many block accesses via index are needed to retrieve a record with a condition “=“ on Department_code? How many block accesses via index are needed to retrieve all the records with a condition “=“ on Department_code?

g. Calculate the number of levels needed if we make each aforementioned index into a multilevel index.

h. Calculate the total number of blocks required by each multilevel index (including the blocks in the extra level of indirection if any).

i. Suppose that the file is not ordered by the key field Ssn and we want to construct a B+-tree access structure (index) on Ssn. Calculate:

(i.1) the orders p and pleaf of the B+-tree;

(i.2) the number of leaf-level blocks needed if blocks are approximately 69% full (rounded up for convenience);

(i.3) the number of levels needed if internal nodes are also 69% full (rounded up for convenience);

(i.4) the total number of blocks required by the B+-tree;

(i.5) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the B+-tree.

j. Repeat (i), but for a B-tree rather than for a B+-tree. Compare your results for the B-tree and for the B+-tree.

Bài làm

a. Record size R = 30 + 9 + 9 + 40 + 10 + 8 + 1 + 4 + 4 +1 = 116 bytes

b. Blocking factor Bfr = B/R = 512/116 = 4 records/block

Number of file blocks = r/bfr = 30000/4 = 7500 blocks

c. Field size Ssn=9 bytes

Block pointer size P=6 bytes.

Index entry size Ri = VSSN+ P = 9+6 = 15 bytes

Index blocking factor bfri = B/Ri= 512/15= 34 entries/block

Number of index entries ri = number of blocks b = 7500 entries

Number of index blocks bi= ri / bfri = 7500/34 = 221 blocks

Binary search on the index needs log2bi = log2 221 = 8 block accesses

Total search 1 record cost via the index is: 8 + 1 = 9 block accesses

d. 1000 distinct values & evenly distributed

Field size Department_code = 9 bytes

Block pointer size P = 6 bytes

Index entry size Ri = 9 + 6 = 15 bytes

Index blocking factor bfri = B/Ri = 512/15 = 34 entries/block

Number of index entries ri = Number of distinct values = 1000 entries

Number of index blocks bi = ri/(B/Ri) = 1000/34 = 30 blocks

Binary search on the index needs log2bi = log2 30= 5 block accesses

Total search 1 record cost via the index is: 5 + 1 = 6 block accesses

Total search all record cost via the index is: 5 + 1 = 6 block accesses

e. Field size Ssn=9 bytes

Record pointer size PR=7 bytes

Index entry size Ri = Vssn + PR = 9 + 7 = 16 bytes

Index blocking factory bfri = B/Ri = 512/16 = 32 entries/block

Number of index entries ri = Number of file records r = 30000 entries

Number of index blocks bi = ri/bfri = 30000/32 = 938 blocks

Binary search on the index needs log2bi = log2 938 = 10 block accesses

Total search 1 record cost via the index is: 10 + 1 = 11 block accesses

f. 1000 distinct values & evenly distributed

Field size Department_code = 9 bytes

Block pointer size Pr = 7 bytes

Index entry size Ri = 9 + 7 = 16 bytes

Index blocking factor bfri = B/Ri = 512/16 = 32 entries/block

Number of index entries ri = Number of distinct values = 1000 entries

Number of index blocks bi = ri/(B/Ri) = 1000/32 = 32 blocks

Index blocking factor at the indirection level bfrii = B/PR = 512/7 = 73 pointers/block

Number of index entries rii per distinct value at the indirection level = number of record pointers per distinct value of Department_code = 30,000/1000 = 30 entries

Number of index blocks per distinct value at the indirection level bii = rii/bfrii = 30/73 = 1 block

Binary search on the index needs log2bi = log2 32 = 5 block accesses

The total search 1 record cost via the index is: 5 + 1 + 1 = 7 block accesses

Number of block accesses to the data file: 30,000/1000 = 30 block accesses

The total search all record cost via the index is: 5 + 1 + 30 = 36 block accesses

g.

+ Primary Index

rii = bi = 221 entries

bii = rii/bfri = 221/34 = 7 blocks

riii = bii = 7 entries

biii = riii/bfri = 7/34 = 1 block

=> 3 levels

- Clustered Index

rii = bi = 30 entries

bii = rii/bfri = 30/34 = 1 block

=> 2 levels

- Secondary Index on ssn

rii = bi = 938 entries

bii = rii/bfri = 938/32 = 30 blocks

riii = bii = 30 entries

biii = riii/bfri = 30/32 = 1 block

=> 3 levels

- Secondary Index on Department_code

rii = bi = 32

bii = rii/bfri = 32/32 = 1 block

=> 2 level

h.

- Primary Index

Number of blocks = bi + bii + biii = 221 + 7 + 1 = 229 blocks

- Clustered Index

Number of blocks = bi+ bii = 30 + 1 = 31 blocks

- Secondary Index on ssn

Number of blocks = bi + bii + biii = 938 + 30 + 1 = 969 blocks

- Secondary Index on Department_code

Number of blocks in indirectional level = distinct values x number of index block per distinct value = 1000 x 1 = 1000 blocks

Number of blocks = bi + bii + Number of blocks in indirectional level

= 32 + 1 + 1000 = 1033 blocks

i.

1.

+ Internal tree node:

  (p x P) + ((p-1) x Vssn) < B

  ⇔ (p x 6) + ((p-1) x 9) < 512

  ⇔ p = 34

+ Leaf node:

  (pleaf x (Vssn + PR)) + P < B

  ⇔ (pleaf x (9 + 7)) + 6 < 512

  ⇔ pleaf = 31

2. Leaf nodes are 69% full, average number of values in a leaf node is:

0.69 x pleaf = 0.69 x 31 = 21.39 = 22 (round up for convenience)

Number of leaf blocks = r / average number of values in a leaf node

= 30000 / 22 = 1364 blocks

3. Internal nodes are 69% full, average number of values in a internal node:

0.69 x p = 0.69 x 34 = 23.46 = 24 (round up)

Number of blocks 2nd level = Number of leaf blocks /24 = 1364/24 = 57 blocks

Number of blocks 3rd level = Number of blocks 2nd level /24 = 57/24 = 2 blocks

Number of blocks 4rd level = Number of blocks 3rd level /24 = 2/24 = 1 block

=> 4 is the highest level

=> Need 4 levels

4. Total number of blocks needed = 1364 + 57 + 2 + 1 = 1424 blocks

5. Number of block accesses = levels + 1 = 4 + 1 = 5 block accesses