Name: Vaibhav Rupapara

Pid : 5155522

COP 5725

Principles of Database Management Systems (SPC)

Solution 8.4

For Alternative (2), the notation *A*, (*B*,*C*) is used where *A* is the search key for the entry and (*B*,*C*) is the *rid* for data entry, with *B* being the page number of the entry and *C* being the location on page *B* of the entry.

For Alternative (3), the notation is the same, but with the possibility of additional *rid*'s listed.

1) An unclustered index on *age* using Alternative (1).

Answer. Contradiction. **Cannot build** unclustered index using Alternative (1) **since** methodis **inherently clustered.**

2) An unclustered index on *age* using Alternative (2).

Answer <11,(1,1)>,<12,(1,2)>,<18,(1,3)>,<19,(2,2)> . The order of entries is not significant.

3) An unclustered index on *age* using Alternative (3).

Answer <11,(1,1)>,<12,(1,2)>,<18,(1,3)>,<19,(2,1),(2,2)>, The order of entries is not significant.

4) A clustered index on *age* using Alternative (1)

Answer 11, 19. The order of entries is significant since the order of the entries is the same as the order of data record.

5) A clustered index on *age* using Alternative (2).

Answer <11,(1,1)>,<19,(2,1)>. The order of entries is significant since the order of the entries is the same as the order of data record.

6) A clustered index on *age* using Alternative (3).

Answer <11,(1,1)>,<19,(2,1),(2,2)> . The order of entries is significant since the order of the entries is the same as the order of data record.

7) An unclustered index on *gpa* using Alternative (1)

Answer Contradiction. **Cannot build** unclustered index using Alternative (1) since methodis **inherently clustered.**

8) An unclustered index on *gpa* using Alternative (2).

Answer <1.8,(1,1)>,<2.0,(1,2)>,<3.2,(2,1)>,<3.4,(1,3) >,<3.8,(2,2)>. The order of entries is not significant.

9) An unclustered index on *gpa* using Alternative (3)

Answer <1.8,(1,1)>,<2.0,(1,2)>,<3.2,(2,1)>,<3.4,(1,3) >,<3.8,(2,2)>. The order of entries is not significant.

10) A clustered index on *gpa* using Alternative (1).

Answer Alternative (1) **cannot be used** to build a clustered index on *gpa* because the records in the file are not sorted in order of *gpa*. Only if the entries in (*1,3*) and (*2,1*) were switched would this possible, but then the data would no longer be sorted on *age* as previously defined.

11) A clustered index on *gpa* using Alternative (2)

Answer Alternative (2**) cannot be used to build** a clustered index on *gpa* because the records in the file are not sorted in order of *gpa*. Only if the entries in (*1,3*) and (*2,1*) were switched would this possible, but then the data would no longer be sorted on *age* as previously defined.

12) A clustered index on *gpa* using Alternative (3).

Answer Alternative (3) **cannot be used to build** a clustered index on *gpa* because the records in the file are not sorted in order of *gpa*. Only if the entries in (*1,3*) and (*2,1*) were switched would this possible, but then the data would no longer be sorted on *age* previously defined.

Question 8.5

Explain the difference between Hash indexes and B+-tree indexes. In particular, discuss how equality and range searches work, using an example.

Solution

Hashing function can be used to build a Hash index, that quickly maps an search key value to a specific location in an array-like list of elements called buckets. The buckets are often constructed such that there are more bucket locations than there are possible search key values, and the hashing function is chosen so that it is not often that two search key values hash to the same bucket. A B+-tree index is constructed by sorting the data on the search key and maintaining a hierarchical search data structure that directs searches to the correct page of data entries. Insertions and deletions in a hash based index are relatively simple.

If two search values hash to the same bucket, called a collision, a linked list is formed connecting multiple records in a single bucket. In the case that too many of these collisions occur, the number of buckets is increased. Alternatively, maintaining a B+-tree's hierarchical search data structure is considered more costly since it must be updated whenever there are insertions and deletions in the data set. In general, most insertions and deletions will not modify the data structure severely, but every once in a while large portions of the tree may need to be rewritten when they become over-filled or under-filled with data entries.

Hash indexes are especially good at equality searches because they allow a record look up very quickly with an average cost of 1.2 I/Os. B+-tree indexes, on the other hand, have a cost of 3-4 I/Os per individual record lookup. Assume we have the employee relation with primary key *eid* and 10,000 records total. Looking up all the records individually would cost 12,000 I/Os for Hash indexes, but 30,000-40,000 I/Os for B+tree indexes.

For range queries, hash indexes perform terribly since they could conceivably read as many pages as there are records since the data is not sorted in any clear grouping or set. On the other hand, B+-tree indexes have a cost of 3-4 I/Os plus the number of qualifying pages or tuples, for clustered or unclustered B+-trees respectively. Assume we have the employees example again with 10,000 records and 10 records per page. Also assume that there is an index on *sal* and query of *age* ¿ 20,000, such that there are 5,000 qualifying tuples. The hash index could cost as much as 100,000 I/Os since every page could be read for every record. It is not clear with a hash index how we even go about searching for every possible number greater than 20,000 since decimals could be used. An unclustered B+-tree index would have a cost of 5,004 I/Os, while a clustered B+-tree index would have a cost of 504 I/Os. It helps to have the index clustered whenever possible.

Question 8.9

What main conclusions can you draw from the discussion of the five basic file organizations discussed in Section 8.4? Which of the five organizations would you choose for a file where the most frequent operations are as follows?

Solution

The main conclusion about the five file organizations is that all five have their own advantages and disadvantages. No one file organization is uniformly superior in all situations. The choice of appropriate structures for a given data set can have a significant impact upon performance. An unordered file is best if only full file scans are desired. A hash indexed file is best if the most common operation is an equality selection. A sorted file is best if range selections are desired and the data is static; a clustered B+ tree is best if range selections are important and the data is dynamic. An unclustered B+ tree index is useful for selections over small ranges, especially if we need to cluster on another search key to support some common query.

1) Search for records based on a range of field values.

Answer Using these fields as the search key,

Sorted file organization or a clustered B+ tree depending on whether the data is static or not.

2) Perform inserts and scans, where the order of records does not matter

Answer

Heap file could be useful.

3) Search for a record based on a particular field value.

Answer Using this particular field as the search key,

A hash indexed file would be the best.

Solution 8.11

1) We should create an unclustered hash index on <ename,age,sal> fields of Emp

(b) since then we could do an index only scan. If our system does not include index only plans then we shouldn't create an index for this query (e). Since this query requires us to access all the Emp records, an index won't help us any, and so should we access the records using a files can.

2) We should create a clustered dense B+ tree index (c) on<floor,budget> fields of Dept, since the records would be ordered on these fields then. So when executing this query, the first record with *floor* = 10 must be retrieved, and then the other records with *floor* = 10 can be read in order of budget. Note that this plan, which is the best for this query, is not an index-only plan (must look up dids).

Question 9.5 Consider a disk with a sector size of 512 bytes, 2000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec.

Solution

1) What is the capacity of a track in bytes? What is the capacity of each surface? What is the capacity of the disk?

Answer

Capacity of track in bytes would be:

Formula used: bytes/sector= 512

Sectors/track=50

Bytes/track = bytes/sector * sectors/track = 512 * 50 = 25000

Bytes/track =25000

Bytes/surface= bytes/track * tracks/surface = 25000*2000= **50000 K**

Bytes/disk= bytes/surface * surface/disk= 50000K *5*2 = **500000 K**

2) How many cylinders does the disk have?

Answer The number of cylinders = the number of tracks on each surface =2000

The disk would have **2000 cylinders**

3) Give examples of valid block sizes. Is 256 bytes a valid block size? 2048? 51200?

Answer The block size should be a **multiple of the sector size**. We can see that **256 is not a valid** block size .

The **valid block size is 2048.**

51200 is not a valid block size in this case because block size cannot exceed the size of a track, which is 25600 bytes.

4) If the disk platters rotate at 5400 rpm (revolutions per minute), what is the maximum rotational delay?

Answer If the disk platters rotate at 5400rpm, the time required for one complete rotation, which is the maximum rotational delay, is (1/5400)× 60 **= 0.011*seconds***

The maximum rotational delay is **0.011 seconds**

The average rotational delay is half of the rotation time**, 0.006 seconds**.

5) If one track of data can be transferred per revolution, what is the transfer rate?

Answer The capacity of a track is 25K bytes. Since one track of data can be transferredper revolution, the data transfer rate is 25k/0.011 = **2,272 Kbytes/ second**

The transfer rate is **2,272 Kbytes/ second**

Question 9.6 Consider again the disk specifications from Exercise 9.5, and suppose that a block size of 1024 bytes is chosen. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

Solution

1) How many records fit onto a block?

Answer

024/100 = 10.

Total **10 records** are fit onto a block.

2) How many blocks are required to store the entire file? If the file is arranged sequentially on the disk, how many surfaces are needed?

Answer There are 100,000 records all together, and each block holds 10 records.

So **10,000 blocks** are required to store the entire file.

One track has 25 blocks, one cylinder has 250 blocks. we need 10,000 blocks to store this file. So we will use more than one cylinders, that is, need **10 surfaces** to store this file.

3) How many records of 100 bytes each can be stored using this disk?

Answer The capacity of the disk is 500,000K, which has 500,000 blocks. Each block has10 records. Therefore, the disk can store no more than **5,000,000 records**.

So, **5,000,000 records** of 100 bytes each can be stored using this disk

4) If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what page is stored on block 1 of track 1 on the next disk surface? How would your answer change if the disk were capable of reading and writing from all heads in parallel?

Answer There are 25K bytes, or we can say, 25 blocks in each track. It is block 26 on block 1 of track 1 on the next disk surface.

If the disk were capable of reading/writing from all heads in parallel, we can put the first 10 pages on the block 1 of track 1 of all 10 surfaces. Therefore, it is block 2 on block 1 of track 1 on the next disk surface.

5) What time is required to read a file containing 100,000 records of 100 bytes each sequentially? Again, how would your answer change if the disk were capable of reading/writing from all heads in parallel (and the data was arranged optimally)?

Answer A file containing 100,000 records of 100 bytes needs 40 cylinders or 400 tracks inthis disk. The transfer time of one track of data is 0.011 seconds. Then it takes 400 × 0.011 = 4.4*seconds* to transfer 400 tracks.

This access seeks the track 40 times. The seek time is 40 × 0.01 = 0.4*seconds*.

Therefore, total access time is 4.4 + 0.4 = ***4.8seconds***.

If the disk were capable of reading/writing from all heads in parallel, the disk can read 10 tracks at a time. The transfer time is 10 times less, which is 0.44 seconds.

Thus total access time is 0.44 + 0.4 = ***0.84seconds***

6) What is the time required to read a file containing 100,000 records of 100 byteseach in a random order? To read a record, the block containing the record has to be fetched from disk. Assume that each block request incurs the average seek time and rotational delay.

Answer

For any block of data,

averageaccesstime = *seektime*+*rotationaldelay*+*transfertime*.

seektime = 10*msec*

rotationaldelay = 6*msec*

transfertime = 1K/(2,250*K/sec*) = 0.44*msec*

The average access time for a block of data would be 16.44 msec. For a file containing 100,000 records of 100 bytes,

The **total access time is 164.4 seconds**.

Question 9.12 What is *sequential flooding* of the buffer pool?

Solution

Sequential flooding occurs in the buffer pool when the user requires sequential scans of the data. When Sequential flooding occurs, LRU (Least Recently Used) proves to be one of the worst possible strategy. If we suppose that a buffer pool has 8 frames and files that are to be scanned are less than 8 or equal to 8, and we assume that there are no other clashing or competing requests for the pages. So the pages in the subsequent scans will find desiderating pages in the buffer pool. Whereas if we suppose that the file to be scanned 9 pages, which is more than 8 in the buffer pool, then if we use LRU (Least Recently Used) every scan of the file will result in reading every page of the file, this situation is called Sequential Flooding.

Question 9.17 Describe two possible page formats. What are the trade-offs between them?

Solution

The two possible page formats are as following:

1) Fixed length records: In Fixed length records, the records are stored in the first N slots where N is the number of records on the page. When we delete a record, the last record on the page is moved to the vacated slot which allows us to check the ith record. Through this all the empty sets are seen at the end of the page. The other way to monitor deletions is by using an array of bits which states one bit per slot thorugh that we can keep track of free slot information. Records are located scanning of bit arrays which find the slots whose bits are on, whereas if the record is deleted its bit is turned off.
2) Variable length records: In variable length records a directory of slots is maintained for each page with a pair per slot. The record offset acts as a 'pointer' to the record and set to -1 if a record is to be deleted.

Trade-off between them:

The consecutive slots organization is mostly used for fixed length record formats. It handles the deletion by using bitmaps or linked lists.

The slot directory organization maintains a directory of slots for each page, with a *record offset*, *record length* pair per slot.

The slot directory is an indirect way to get the offset of an entry. Because of this indirection, deletion is easy. It is accomplished by setting the length field to 0. And records can easily be moved around on the page without changing their external identifier.

Question 9.19 Consider the two internal organizations for heap files (using lists of pages and a directory of pages) discussed in the text.

Solution

1) Describe them briefly and explain the trade-offs. Which organization would you choose if records are variable in length?

Answer Linked list of pages: Linked list of pages keeps the track of the pages that have free space. It keeps the track of the pages with free space and a doubly linked list of full pages.

In a linked list, a new page is obtained by making a request to the disk space manager. If a page has to be deleted from the heap file, it is removed from the list and the disk space manager.

Directory of pages: The directory is the collection of pages and through the directory of pages, the Database management system remembers where the directory page of each heap file is located.

Each directory entry can identify a page in the heap file, and as the heap file is edited or changed, the number of entries in the directory also change automatically. If the heap file increases or decreases, the number of pages in the directory increase or decrease automatically.

Trade-off between them: In linked list of pages all pages in the file are on the free list if records are of variable length and that is so because every page has at least a few free bytes, whereas in Directory of pages free space is managed by maintaining a bit per entry which indicates whether the corresponding page any free space or a count per entry.

2) Can you suggest a single page format to implement both internal file organizations?

Answer A single page format to implement both internal file organizations would be a page format with page pointers such as the previous and next page pointers.