CS-245 Database System Principles – Winter 2004
Assignment 2

Solutions

A relational database system holds three relations: *C* (companies), *P* (products) and *M* (models) with the following characteristics:

Relation *C* (company):

- Tuples are stored as fixed length, fixed format records, length 250 bytes.
- There are 15,000 *C* tuples.
- Tuples contain key attribute *C.N* (company number), length 20 bytes; other fields and record header make up rest.

Relation *P* (product):

- Tuples are stored as fixed length, fixed format records, length 200 bytes.
- There are 45,000 *P* tuples.
- Tuples contain attribute *P.N* (the company number who makes the product), length 20 bytes; other fields and record header make up rest.
- Tuples also contain attribute *P.I* (product identifier), length 25 bytes.

Relation *M* (model):

- Tuples are stored as fixed length, fixed format records, length 100 bytes.
- There are 135,000 *M* tuples.
- Tuples contain attribute *M.I* (the identifier of the product involved), length 25 bytes, and an attribute *M.P* (the price of the product), length 10 bytes; other fields and record header make up rest.

While the number of products associated with each company varies, for evaluation purposes we may assume that each company has 3 products, and each product has 3 model records associated with it. Thus, you can assume that there are 9 model records for each company record.

The records are to be stored in a collection of 4 kilobyte (4096 bytes) disk blocks that have been reserved to exclusively hold *C*, *P*, or *M* records, or combinations of those records, and indexes over the records. (That is, there are no other types of records in the blocks we are discussing in this problem.) Each block uses 50 bytes for its header; records are not spanned.

Two disk organization strategies are being considered:

1. Sequential
   - All the company (*C*) records are placed sequentially (ordered by company number) in one subset of blocks.

- Product (*P*) records are separate in another set of blocks. Products are ordered by company number.
- Finally, model (*M*) records are in a third set of blocks, ordered by product identifier.

2. Clustered
- For each company (*C*) record, the 3 products for that company (*C.N = P.N*) reside in the same block.
- Similarly, the 9 model records for each company reside in the same block as their company record .
- The company records are sequenced by company number.

Problem 1. (20 points)

For each storage organization, compute the total number of disk blocks required to hold all three relations.

Sequential:
- *Each C tuple is 250 bytes. Each block holds ë4046 bytes/250 bytesû = 16 tuples (where ë û indicates round down). There are 15,000 tuples, so é15000 tuples/16 tuples per blockù = 938 blocks required (where é ù indicates round up).*
- *Each P tuple is 200 bytes. Each block holds ë4046 bytes/200 bytesû = 20 tuples. There are 45,000 tuples, so é45000 tuples/20 tuples per blockù = 2250 blocks required.*
- *Each M tuple is 100 bytes. Each block holds ë4046/100 bytesû = 40 tuples. There are 135,000 tuples, so é135000 tuples/40 tuples per blockù = 3375 blocks required.*
- *The total is 938 + 2250 + 3375 =* ***6563 blocks.***

Clustered:
- *A company record, plus its associated product records, plus their associated model records, is 250 + 3x200 + 9x100 = 1750 bytes. Each block holds ë4046/1750û = 2 companies and associated product and model tuples. There are 15,000 company tuples, so é15,000 / 2ù =* ***7500 blocks.***

Problem 2. (10 points)

Imagine that we are told there are two main query types:
- *Table scan*: scan all of the company (*C*) records in order by company id.
- *Join*: For a given company number *C.N*, get the company record followed by all its model records. That is, get all model (*M*) records with *M.I = P.I* for each *P* record with *P.N = C.N*.

Which storage organization would you prefer for each query type? Why?

For table scan, the sequential organization is best. This is because all of the company tuples can be read in order without reading any product or model tuples. This reduces the number of I/Os.

For the join, the clustered organization is best. This is because a single block contains the answer to the join. Under the sequential organization, at least 3 and as many as 13 blocks may have to be read.

Problem 3. (35 points)

Let us now examine some indexing options. Imagine that you want to build a primary index on *C.N*, and a secondary index on *M.P*. Each index will associate with each key a 10 byte pointer to data (an 8 byte block id and a 2 byte offset). For each storage organization (sequential or clustered) and index type (primary and secondary), compute the number of 4096 byte blocks needed for the sparse index, and the number of 4096 byte blocks needed for a dense index. (Recall that 50 bytes of each block are used as a header.) If a particular type of index does not make sense, state so and explain why. Treat duplicates in the secondary index in a straightforward way; that is, for each *M* tuple there should be an index entry, even if there are multiple entries with the same key. Construct each index to use the minimum space possible without spanning index entries.

Sequential organization, sparse primary index: This index requires one index entry for every C block. Each index entry is 20 bytes for the key plus 10 bytes for the pointer, or 30 bytes. Each 4096 byte block can hold ë4046/30û = 134 entries. There are 938 blocks in the C table, so é938/134ù = ***7 index blocks are required.***

Sequential organization, dense primary index: This index requires one index entry for every C tuple. Each 4096 byte block can still hold 134 entries, but there are now 15,000 entries needed, so é15,000/134ù = ***112 blocks are required****.*

Sequential organization, sparse secondary index: You cannot build a sparse secondary index. Since the M tuples are not ordered by M.P, a sparse index will not allow you to find any entries that are not in the index.

Sequential organization, dense secondary index: This index requires one index entry for every M tuple. Each index is 10 bytes (for the key) + 10 bytes (for the pointer) = 20 bytes. So each 4096 byte block can hold ë4046/20û = 202 entries, and there are 135,000 M tuples, so é135,000/202ù = ***669 blocks are required.***

Clustered organization, sparse primary index: This index requires one index entry for every data block. Each 4096 byte index block can still hold 134 entries, but now there are 7500 entries needed, so é7500/134ù = ***56 blocks are required.***

Clustered organization, dense primary index: This index is the same size as the dense primary index for the sequential organization, so ***112 blocks are required.***

Clustered organization, sparse secondary index. You cannot build a sparse secondary index. Since the M tuples are not ordered by M.P, a sparse index will not allow you to find any entries that are not in the index.

Clustered organization, dense secondary index. This index requires one index entry for every M tuple. The size is the same for the sequential organization, or ***669 blocks.***

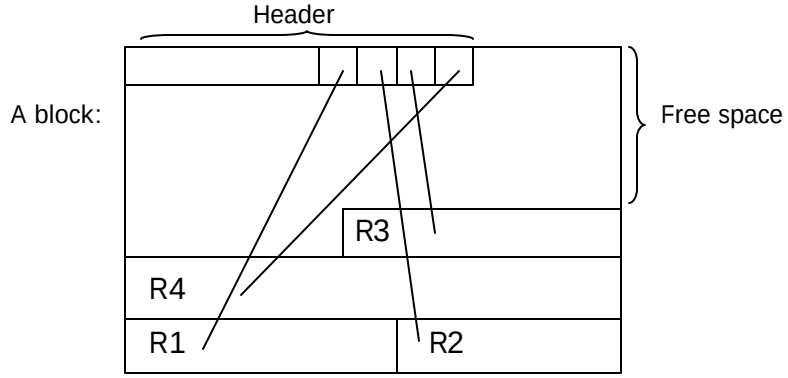
Problem 4. (15 points)

For the dense secondary index of problem 3 (on *M.P*) you decide that you want to create a bucket implementation to handle duplicate keys. Duplicate keys will be managed by inserting a single copy of the key into the secondary index, and associating with this key a 10 bytes pointer to the beginning of a list of data pointers for that key. The lists of data pointers for the keys are packed into a sequence of 4096 byte "bucket" blocks (50 bytes of each block is still used as a header). Each block may contain a list for multiple keys; that is the pointer lists for the values "$10,000," "$11,000," "$12,000" and so on may coexist in the same bucket. Similarly, a list for one value may span multiple blocks; e.g. the list for "$13,000" may begin in one block and end in another. Assume that there are 10,000 unique *M.P* values. How many bucket blocks are needed, for each storage organization (sequential and clustered)? What is the new size of the dense secondary index under each storage organization? Construct the index and buckets to use the minimum space possible without spanning index entries or pointers.

For both storage organizations, there are 135,000 data pointers needed. Each bucket block can hold ë4046/10û = 404 pointers, so there are é135,000/404ù = ***335 bucket blocks needed****, for either storage organization.*

The dense secondary index has 202 entries per block. If there are 10,000 unique M.P values, we only need 10,000 index entries. é10,000/202ù = ***50 blocks needed for the index.***

Problem 5. (20 points)



Assume we are using the "indirection-in-block" approach (shown above) for storing records in each block. Suppose that we have 4096-byte blocks in which we store fixed-length records of 40 bytes. The block header consists of an 8-byte pointer to the "next" block and an offset table, using 2-byte pointers to records within the block. On an average day, three records per block are inserted, and two records are deleted. A deleted record must have its pointer replaced by a "tombstone," because there may be dangling pointers to it. For specificity, assume the deletions on any day always occur before the insertions. If the block is initially empty, after how many days will there be no room to insert any more records?

On day 1 before any insertions, the block has an 8-bytes header only. After day 1's insertion, the block has an 8-bytes header, 3 2-bytes pointers, and 3 40-bytes records for a total of 134 bytes. On subsequent days, when we perform deletion, we replace two pointers with two tombstones and free two 40-bytes records. We then add 3 2-bytes pointers and 3 40-bytes records. Thus after day 1, the block only grows by 3 x 2 + 40 = 46 bytes.

Let n be the number of days so far, then the number of bytes used in the block is 134 + 46 x (n-1). Since the block is only 4096 bytes, we can determine when the block will run out of space by solving 134+46 x (n-1) > 4096. We get n >87. Therefore we will run out of space ***after 87 days*** *or* ***on day 88.***