**COMP 418 TME 2**

**Jason Bell 3078931**

**March 30, 2016**

**Part 1**

1. **Explain what a typical optimizer does, and what strategies can be used to generate a good plan.**

An optimizer attempts to find an efficient plan with which to carry out a query. It enumerates a set of possible solution plans and uses stored statistical knowledge and heuristics to assign each a cost. This set of plans will not be all possible plans, as this is likely to be more costly to evaluate than the time saved by the optimization. Therefore, the optimizer results in an efficient plan, but this may not be the most efficient possible plan.

What determines whether the heuristic rates a plan as a good one include factors such as how early data is reduced, the smallest joins possible, and statistical knowledge recorded previously (ie, have previous queries on a table indicated that certain queries are faster?).

1. **Briefly explain the concept of blocked I/O and why it is cheaper to read pages using blocked I/O rather than reading them using several independent requests.**

Blocked I/O allows one to read a sequence of pages in a block in one I/O, rather than one at a time. Thus the cost is the time to seek the block and read the block of pages. Independent requests require reading pages one at a time, with one I/O per page plus a seek per page that is not contiguous with the previous page. Clearly the number of seeks and I/O required to read pages one at a time makes this much costly than blocked I/O.

1. **Explain why two different hash functions are used in hash join, and how to select them.**

The first hash function partitions the tables being joined based on the column being joined. This allows us to only compare tuples shared between corresponding partitions, rather than comparing all tuples in one to all tuples in the other.

The tuples found in corresponding partitions have the same hash value according to the hash function used to partition them. Therefore, a second, different, hash function is needed for equality tests within the corresponding partitions.

1. **For each of the following join techniques, discuss the contexts for which they are most suitable and for which they are not (list their pros and cons): hash join, sort-merge join, and block nested loops join.**

Hash join: an advantage is that hash functions are extremely fast, and that this method requires very little memory (less than sort-merge and block nested loops). However, it is vulnerable to data skew: the hash function may result in widely different sized partitions. As an extreme example, suppose that the data is partitioned into a number of partitions, but one of them holds 90% of the data? There is likely to be any performance gain in this case.

Sort merge: one advantage is that if the data files are already sorted on the join column, they are already partitioned for faster merging of equivalent partitions. Sort merge is also less vulnerable to data skew than hash join. Finally, by the nature of the algorithm, the returned join is sorted, which may be advantageous depending on the situation. In other words, if you want sorted results, you may as well use a sort merge since it already incorporates the cost of doing that.

If there is sufficient memory, block nested loops join is as fast as the hash join, without being vulnerable to data skew. Otherwise the hash join is preferable.

1. **Define the concept of reduction factor, and explain how it is calculated for equality clauses using the statistics available in database catalogs (consider only the WHERE clauses: column = value and column1 = column1).**

(I am assuming that the question means column1 = column2: a join of two tables on a column).

The reduction factor is the number of tuples that satisfy a condition, typically a join or selection. For example, if there are 100 tuples in a table, and 10 match an equality selection, the reduction factor is 0.1. This knowledge is important for choosing an optimal access path: the sooner that the path reduces the number of tuples being worked with, fewer tuples will need to be worked with later in the path. However, this information isn’t precisely known ahead of time and must be estimated.

For selections (column = value), when the column is not indexed with a hash index, a rough estimate of 0.1 is used as a reduction factor. If there is a hash index, the reduction factor can be estimated inverse of the number of unique keys in the index. If there are 20 unique keys, it can be estimated that the reduction factor is 1/20 or 0.05. This assumes uniform distribution: each key corresponds to roughly equal numbers of tuples.

For joins (column1 = column2), if neither column has a hash index, an estimate of 0.1 is again used. If one of them has a hash index, the reduction factor is estimated as the inverse of the number of unique keys in that index. If both have a hash index, the estimate is the inverse of the number of unique keys in the table that has fewer unique keys: we can be certain that the table with more unique keys will have many columns that do not match and can be discarded.

**Part 2**

1. **As a database administrator, you are in charge of regularly tuning the performance of the system. After a long period of operation, you noticed that queries making use of a specific file start to slow down. This particular file used to be sorted but is not anymore. After examining the type of queries processed on the file, you decided that sorting the query would improve its speed. Given that there is a B+ tree index on the file with the same search key, you know you have two options:**

* **Retrieving the records in order through the index.**
* **Retrieving the records in random order and then sorting them.**

**Compare the estimated cost of the two approaches to figure out which one is better to use. (Note: you are not required to do an exact calculation of the cost. It’s rather a general comparison).**

Retrieving records without the index and manually sorting them is the clear winner: records can be loaded in batches, reducing the number of I/O operations. Conversely, finding the records through the index will require loading each record individually, as it is not stated that the index is clustered.

Suppose that there are 1000 records. Reading these records through the index will require 1000 file I/Os. However, if they are found in 10 pages, and we can load 1 page at a time, we can read all records with 10 file I/Os. If we can load pages in blocks and the file isn’t fragmented, we may be able to load all of the records in a single file I/O. Loading pages instead of individual records also lends itself well to external sort algorithms.

1. As a system administrator for the accounting firm AFS, you are usually asked to import data from a regular source and upload the records to a heap file that needs to be sorted after the import. You want to write a script that will do that automatically, so you need to estimate the cost of the sort operation. Your DBMS uses external sort and makes efficient use of the available buffer space when it sorts a file. Information about the newly loaded file and the DBMS software available to operate on it are as follows:

The number of records in the file is 4500. The sort key for the file is 4 bytes long. You can assume that rids are 8 bytes long and page ids are 4 bytes long. Each record is a total of 48 bytes long. The page size is 512 bytes. Each page has 12 bytes of control information on it. Four buffer pages are available.

1. **How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be? (5 marks)**

A buffer of 4 pages means that we can read in 4 pages at a time. The total number of bytes used by the records are 4500 \* 48 = 216,000 bytes. The total file size is the total of the bytes required for the records plus the bytes required for the control information for each page in the file. Where N is the number of pages:

File size = 216,000 + (N \* 12)

Dividing by the page size, 512 bytes, gives the number of pages in the file:

N = [216,000 + (N \* 12)] / 512 = 432 pages.

Note that if we hadn’t considered the control information, the number of pages would have been calculated as 421.875: an inaccurate underestimate.

The number of subfiles produced by the first pass is determined by the number of buffer pages being used: 4. Therefore the first pass on this file will produce 432/4 = 108 subfiles. The control information per page has already been included, so we can find the size of each subfile as 4 \* 512 bytes = 2048 bytes.

1. **How many passes (including the initial pass just considered) are required to sort this file? (5 marks)**

Pass 1: as part a: each of the 432 subfiles has been internally sorted.

Pass 2: We can use three of the four buffer pages to read in pages for the merge sort (the fourth page is needed as an output buffer).

The number of subfiles resulting from a three-way merge sort is 532/3 = 177.333

Pass 3:

The number of subfiles resulting from this pass is 177.333/3 = 59.111

Pass 4:

The number of subfiles resulting from this pass is 59.111 / 3 = 19.7037

Pass 5:

The number of subfiles resulting from this pass is 19.7037 / 3 = 6.5679

Pass 6:

The number of subfiles resulting from this pass is 6.5679 / 3 = 2.1893

Pass 7:

Merge the three (the 0.1893 will be a third, separate file) into one sorted file

Therefore sorting this file will require 7 passes. We can confirm this with the formula given in the text:

Passes = [log3532] + 1 = 6.71 = 7 passes

1. **What is the total I/O cost for sorting this file?**

There are 7 passes, and each pass both reads and writes all pages of the file, for 2 batches of 432 page I/O operations per pass:

I/O cost = 7 passes \* (432 \* 2 per pass) = 7 \* 432 \* 2 = 6048 I/O operations

1. **As a database administrator, if your DMBS offers the possibility of using any of the join algorithms (sort-merge join, hash join, nested loops join), which one would you use in each of the following situations:**
2. **The join condition is not equality. (3 marks)**

A sort merge would work best: a hash join works best for equality conditions, and in the absence of an index, a nested loop join will be prohibitively expensive.

1. **The join condition is not equality and you have a B+ tree index. (4 marks)**

Nested loops join is preferable because you can use the relation with the B+ tree index as the inner relation. Since the inner relation must be read far more times, anything that facilitates this will be a good thing. For every row in the unindexed table, you can probe the index of the other table to find matches.

1. **The join condition is equality. (3 marks)**

A hash join is preferable as it provides fast equality selections. Hash functions are extremely fast, and can be used to partition the data of both relations, and then only need to compare the columns in corresponding partitions. In the general case, this may be faster than the other options. However, it assumes uniform sized partitions result from the hash function, which may or may not be the case. If it is not the case, a sort-merge join is preferable, though it in turn requires a larger buffer.

1. **A programmer team in your organization is developing a web-based application that will be accessible to the customers. In order to make it efficient they need estimate the cost of different operations the system needs to do before answering a user request. One of the services to users involves joining two relatively large relations (RA and RB), so they asked you as a system administrator to provide an estimate of the cost for this join operation. You know that the DMBS supports only nested loops join algorithms. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.**

**Given the following information about the relations to be joined:**

**Relation RA contains 10,000 tuples and has 10 tuples per page.  
Relation RB contains 2,000 tuples and also has 10 tuples per page.  
Both relations are stored as simple heap files.  
Neither relation has any indexes built on it.  
52 buffer pages are available.**

1. **What is the cost of joining RA and RB using a page-oriented simple nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged? (5 marks)**

Relation RA has 1000 pages

Relation RB has 200 pages

The cost is the cost of reading all of RA’s pages, plus the code of reading each page of RB per page of RA:

Cost = 1000 + (1000 \* 200) = 201,000 I/Os

What if we reverse the relations?

Cost = 200 + (200 \* 1000) = 200,200 I/Os

Three pages is the minimum number of buffer pages for this cost to be unchanged: one page to read in a page from one relation, another page to read in a page from the other relation, and a page to store the joined tuples prior to writing to file.

1. **What is the cost of joining RA and RB using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?(5 marks)**

Relation RA has 1000 pages

Relation RB has 200 pages

We’ll use one buffer page to read in RA pages, another 50 to read in RB pages in batches. Therefore, RB will be read 200/50 = 4 times per scan of RA. The remaining page is to write out the joined tuples.

The cost is the cost of reading in all of RA’s pages, plus the cost of reading each batch of RB’s pages for every page of RA:

Cost = 1000 + (1000 \* 4) = 5000 I/Os

What if we reverse the relations?

Read in RA in batches of 50: 1000/50 = 20 times per scan of RB

Cost = 200 + (200 \* 20) = 4200 I/Os

Mathematically, 52 is the minimum number of pages necessary: one to load one relation by page, 50 to load the other relation in blocks, and one to store the joined tuples prior to writing to file. Any reduction in the number of pages in the buffer will lower the size of the blocks that can be read, and will impact performance.

1. **For the purpose of tuning your database that has begun to slow down for some queries, you are looking for the best plan to execute the following queries on the relation described below:**

**The relation you are dealing with is Employee with attributes ename, title, dname, and address; all are string fields of the same length.**

**The ename attribute is a candidate key.  
The relation contains 10,000 pages.  
There are 10 buffer pages.**

**(When answering the questions, make sure to describe the plan you have in mind.)**

**The first query is:**

**SELECT E.title, E.ename FROM Employee E WHERE E.title=‘Administrator’**

**Assume that only 10% of Employee tuples meet the selection condition.**

1. **Suppose that a clustered B+ tree index on ename is (the only index) available. What is the cost of the best plan? (5 marks)**

Without an index that will help reduce the number of tuples that need to be inspected (such as an index on the title column) a full scan must be performed. Since the B+ index is only on ename, and we’re selecting on title, we have the scan the entire heap data file directly. Therefore, the cost will be the cost to read 10,000 pages with 10 buffer pages: 1000 I/Os. The only benefit that this index provides is that the result will be sorted on ename.

1. **Suppose that a clustered B+ tree index on title is (the only index) available. What is the cost of the best plan? (5 marks)**

Since it is a clustered index, the index can tell us where in the heap file title = “Administrator” starts and ends, and we only need to scan that range: we need to scan the 10% of pages that contain tuples that match the condition. Scanning 1000 pages with a buffer of 10 pages means that the cost will be 100 I/Os.

**The second query is:**

**SELECT E.ename FROM Employee E WHERE E.title=‘Administrator’ AND E.dname=‘Finance’**

**Assume that only 10% of Employee tuples meet the condition E.title =’Administrator’, only 10% meet E.dname =’Finance’ and only 5% meet both conditions.**

1. **Suppose that a clustered B+ tree index on dname is (the only index) available. What is the cost of the best plan? (5 marks)**

The presence of an index on dname means that we can first use the conditional (dname = “Finance”) to reduce the number of tuples that must be inspected. The index is clustered, so the first instance of “Finance” in the index will point to the first instance of “Finance” in the heap file. The result is that we only need to scan that range of the heap file. This means we only need to scan 10% of pages: 1000 pages. With a 10 page buffer, the cost will be 100 I/Os.

1. **Suppose that a clustered B+ tree index on <dname, title, ename> is (the only index) available. What is the cost of the best plan? (5 marks)**

This index provides all data needed for this query, and so we can do an index-only scan. The cost of this is relatively negligible.