

CMPT-454 Fall 2009
Instructor: Martin Ester
TA: Yi Liu

Review for the Final Exam

The following chapter exercises and their solutions have been selected from the textbook (first edition) and the online material provided by the authors.

Exercise 11.3.1

The Megatron 777 disk has the following characteristics:

- There are ten surfaces, with 10,000 tracks each.
- Tracks hold an average of 1000 sectors of 512 bytes each.
- 20% of each track is used for gaps.
- The disk rotates at 10,000 rpm.
- The time it takes the head to move n tracks is $1 + 0.001 * n$ msec.

a) What is the capacity of the disk?

The disk has $10 * 10,000 = 100,000$ tracks. The average track has $1000 * 512 = 512,000$ bytes. Thus, the capacity is 51.2 gigabytes.

b) What is the maximum seek time?

The maximum seek time occurs when the heads have to move across all the tracks. Thus, substitute 10,000 (really 9999) for n in the formula $1 + 0.001n$ to get 11 milliseconds.

c) What is the maximum rotational delay?

The maximum rotational latency is one full revolution. Since the disk rotates at 10,000 rpm, it takes $1/10000$ of a minute, or 6 milliseconds.

Exercise 11.7.2

Suppose that a disk has probability F of failing in a given year, and it takes H hours to replace a disk.

a) If we use mirrored disks, what is the mean time to data loss, as a function of F and H ?

To compute the mean time to failure, we can compute the probability that the system fails in a given year. The MTTF will be the inverse of that probability. Note that there are 8760 hours in a year.

The system fails if the second disk fails while the first is being repaired. The probability of a failure of one of the two disks in a year is $2F$. The probability that the second disk

will fail during the H hours that the other is being prepared is $FH/8760$. Thus, the probability of a failure in any year is $2F^2H/8760$, and the MTTF is $4380/F^2H$.

- b) If we use a RAID level 4 or 5 scheme, with N disks, what is the mean time to data loss?

The system fails if any of the other $N-1$ disks fails while the first is being repaired. The probability of a failure of one of the N disks in a year is NF . The probability that a second disk will fail during the H hours that the other is being prepared is $(N-1)FH/8760$. Thus, the probability of a failure in any year is $N(N-1)F^2H/8760$, and the MTTF is the inverse of this probability.

Exercise 13.3.1

Suppose that blocks can hold either ten records or 99 keys and 100 pointers. Also assume that the average B-tree node is 70% full, i.e. it will have 69 keys and 70 pointers. We can use B-trees as part of several different structures. For each structure described below, (i) the total number of blocks needed for a 1,000,000 record file, and (ii) the average number of disk I/O's to retrieve a record given its search key. You may assume nothing is in memory initially, and the search key is the primary key for the records.

- a) The data file is a sequential file, sorted on the search key, with 10 records per block. The B-tree is a dense index.

First, there are 100,000 data blocks. If there are an average of 70 pointers per block in the bottom-level nodes of the B-tree, then there are $1,000,000/70 = 14286$ B-tree blocks at that level. The next level of the B-tree requires $1/70$ th of that, or 204 blocks, and the third level has $1/70$ th of that, or 3 blocks. The fourth level has only the root block. The number of blocks needed is therefore $100,000 + 14,286 + 204 + 3 + 1 = 114,494$ blocks.

Since the B-tree has four levels, we must make a total of five disk reads to go through the B-tree to the desired data block.

- e) The data file is a sequential file, and the B-tree is a sparse index, but each primary block of the data file has one overflow block. On average, the primary block is full, and the overflow block is half full. However, records are in no particular order within a primary block and its overflow block.

To begin, there are $1,000,000/15$ primary blocks, or 66,667 primary blocks, each with 10 records, and the same number of overflow blocks, each with 5 records. The number of first-level B-tree blocks is $66,667/70 = 953$. At the second level there are $953/70 = 14$, and the third level has only the root. Thus, there are 133,334 data blocks and 968 index blocks, for a total of 134,302 blocks.

A search requires three disk I/O's to go through the B-tree to the data file. Once there, we surely need to read the primary block, and $1/3$ of the time we shall need to see the overflow block as well. Thus, the average number of disk I/O's is $13/3$.

Exercise 13.3.4

What are the minimum numbers of keys and pointers in B-tree (i) interior nodes and (ii) leaves, when

a) $n = 10$, i.e. a block holds 10 keys and 11 pointers.

Interior nodes: at least 5 keys and 6 pointers. Leaves: at least 5 keys and pointers, not counting the optional extra pointer to the next leaf block.

Exercise 13.4.6

Suppose keys are hashed to four-bit sequences and blocks can hold three records. If we start with an empty hash table with two empty blocks (corresponding to 0 and 1), show the organization after we insert records with keys:

a) 0000, 0001, . . . , 1111, and the method of hashing is extensible hashing.

When we insert 0011, there are four records for bucket 0, which overflows. Adding a second bit to the bucket addresses doesn't help, because the first four records all begin with 00. Thus, we go to $i = 3$ and use the first three bits for each bucket address. Now, the records divide nicely, and by the time 1111 is inserted, we have two records in each bucket.

Exercise 15.3.2

Suppose $B(R) = B(S) = 10,000$ and $M = 1000$. Calculate the disk I/O cost of the nested-loop join.

The formula in Section 15.3.4 gives $10000 + 10000 * 10000 / 999$ or 110,101.

Exercise 15.4.3

If $B(R) = B(S) = 10,000$ and $M = 1000$, what are the disk I/O requirements of:

b) simple sort join.

The formula of Fig. 15.11 gives $5 * (10000 + 10000) = 100,000$. Note that the memory available, which is 1000 blocks, is larger than the minimum required, which is $\sqrt{10000}$, or 100 blocks. Thus, the method is feasible.

Exercise 15.6.1

Suppose there is an index on attribute R.a. Describe how this index could be used to improve the execution of the following operations. Under which circumstances would the index-based algorithm be more efficient than sort- or hash-based algorithms?

a) R SET-UNION S (R and S have no duplicates)

Our option is to first copy all the tuples of R to the output. Then, for each tuple t of S , retrieve the tuples of R for which $R.a$ matches the a -value of t . Check whether t appears among them, and output t if not. This method can be efficient if S is small and $V(R,a)$ is large.

Exercise 16.2.2

Give examples to show that

a) Projection cannot be pushed below set union.

Let $R(A,B) = \{(1,2)\}$ and $S(A,B) = \{(1,3)\}$. Then $pi_A(R \text{ union } S) = pi_A(\{(1,2), (1,3)\}) = \{(1), (1)\}$. But $pi_A(R) \text{ union } S pi_B(S) = \{(1)\}$, since the set union removes duplicates from the union.

Exercise 16.4.1

Below are the vital statistics for four relations W, X, Y, and Z:

W(a,b)	X(b,c)	Y(c,d)	Z(d,e)
T(W)=100	T(X)=200	T(Y)=300	T(Z)=400
V(W,a)=20	V(X,b)=50	V(Y,c)=50	V(Z,d)=40
V(W,b)=60	V(X,c)=100	V(Y,d)=50	V(Z,e)=100

Estimate the sizes of relations that are the results of the following expressions:

a) W JOIN X JOIN Y JOIN Z

Start by taking the product of the sizes of the relations: $100*200*300*400 = 2,400,000,000$. Since attribute b appears in two relations, divide by the larger of $V(W,b)$ and $V(X,b)$, which is 60. Similarly, for c divided by 100, and for d divide by 50. The result is 8000 tuples.

b) SELECTION $a=10$ (W)

The estimate is $T(W)/V(W,a) = 100/20 = 5$.

Exercise 17.2.2

For each of the sequences of log records representing the actions of transaction T, tell all the sequences of events that are legal according to the rules of undo logging, where the events of interest are the writing to disk of the blocks containing database elements, and the blocks of the log containing the update and commit records. You may assume that log records are written to disk in the order shown; i.e. it is not possible to write one log record to disk while a previous record is not written to disk.

a) <START T>; <T,A,10>; <T,B,20>; <COMMIT T>

There are five events of interest, which we shall call:

1. *A*: database element *A* is written to disk.
2. *B*: database element *B* is written to disk.
3. *LA*: the log record for *A* is written to disk.
4. *LB*: the log record for *B* is written to disk.
5. *C*: the commit record is written to disk.

The undo rules imply the following constraints:

1. *C* must be last.
2. *LA* is before *A*.
3. *LB* is before *B*.
4. *LA* is before *LB*.

Note that (4) comes from the fact that we assume log records are written to disk in the order created.

We can conclude that *LA* is first; every other event must be preceded by something. Either *A* or *LB* can be next. In the first case, the only possible order of events is *LA,A,LB,B,C*. In the second case, *A* and *B* can then be written in either order, giving us the following two sequences: *LA,LB,A,B,C* and *LA,LB,B,A,C*, or a total of 3 possible sequences.

Exercise 17.4.5

Consider the following sequence of log records: <START S>; <S,A,60,61>; <COMMIT S>; <START T>; <T,A,61,62>; <START U>; <U,B,20,21>; <T,C,30,31>; <START V>; <U,D,40,41>; <V,F,70,71>; <COMMIT U>; <T,E,50,51>; <COMMIT T>; <V,B,21, 22>; <COMMIT V>;

Suppose that we begin a non-quiet checkpoint immediately after the following log record has been written (in memory):

b) <T,A,61,62>

For each, tell:

(i) At which points could the <END CKPT> record be written, and

(ii) For each possible point at which a crash could occur, how far back in the log we must look to find all possible incomplete transactions. Consider both the case that the <END CKPT> record was or was not written prior to the crash.

i. The `END CKPT` record could appear anywhere after the record $\langle T, A, 61, 62 \rangle$. The reason is that the writing of dirty blocks can be performed independent of whatever actions the transactions are performing in the interrupt.

ii. The only active transaction when the checkpoint began was T . If the crash occurs after the `END CKPT`, then we need only go back as far as the start of T . However, if the crash occurs before the checkpoint ended, then any transaction that was active when the *previous* checkpoint ended may have written some but not all of its data to disk. In this case, the only other transaction that could qualify is S , so we must look back to the beginning of S , i.e., to the beginning of the log in this simple example.