
# Database System Implementation

## Solutions for Chapter 3



## Solutions for Section 3.2

### Exercise 3.2.1

First, note that SQL2 dates require 10 bytes, and SQL2 times require 8 bytes if there is no decimal point; this material is in Section 3.1.3.

a) The bytes requirsed by each of the fields is 15 + 2 + 10 + 8 = 35.

b) Round each of the four field lengths up to a multiple of 4, to get 16 + 4 + 12 + 8 = 40.

c) Round up again, to a multiple of 8, to get 16 + 8 + 16 + 8 = 48.

### Exercise 3.2.3

The elements in the header also require rounding upward in (b) and (c). Thus, the lengths including header fields and adding the lengths for the bodies as calculated in Exercise 3.2.1 are:

a) 4 + 4 + 1 + 35 = 44.

b) 4 + 4 + 4 + 40 = 52.

c) 8 + 8 + 8 + 48 = 72.

### Exercise 3.2.5

For parts (a) and (b), the header takes 40 bytes, while for (c) it takes 80 bytes. The calculations are thus:

a) (4096-40)/44 = 92.

b) (4096-40)/52 = 78.

c) (4096-80)/72 = 55.

Return to Top

# Solutions for Section 3.3

### Exercise 3.3.1

The Megatron 747 has 8192 cylinders, so we require 2 bytes for the cylinder. There are 8 surfaces, or 8 tracks per cylinder, so 1 byte suffices for the track. The average number of sectors per track is 256, and we have chosen to use 8 sectors per block (which is not a property of the disk itself).

If we used blocks consisting of a single sector, and the track with the greatest number of sectors had more than 256, then we would need 2 bytes for the sector. However, at 8 sectors/block, we could have wildly differing numbers of sectors per track, and still not get close to 256 blocks/track. Since 256 integers can be represented in 1 byte, we think that 1 byte is sufficient for the block within a track, and we think the correct answer is 4 bytes for the block address on a Megatron 747.

### Exercise 3.3.3(a)

Assuming 4096-byte blocks, we need another 2 bytes for the position within a block, or 6 bytes total.

### Exercise 3.3.7

On the first day, since the deletion occurs before any insertions, there really isn't a deletion, and there are two records and two pointers, for a total of 204 bytes (this number could be slightly higher if you make the assumption that pointers must be aligned at a multiple of 4 or 8). At each subsequent day, the deletion leaves a tombstone in the offset table, but one of the 100-byte record areas can be reused. Thus, the space occupied grows by 104 bytes (one record area plus two pointers). On day 39, the requirement for space will exceed 4096. We shall then require 204 + 38*104 = 4156 bytes and be unable to make the second insertion.

### Exercise 3.3.10

Suppose $c$ is the cost of swizzling an individual pointer. This question asks for what values of $p$ is the cost of swizzling fraction $p$ of the pointers at cost $c$ greater than swizzling them all at cost $c/2$? That is, $pc > c/2$, or $p > 1/2$.

Return to Top

# Solutions for Section 3.4

### Exercise 3.4.1

The fixed-length fields require 30 bytes. For the variable-length fields we need in the header a record length and pointers to all but the first of the three variable-length fields, or another 12 bytes. Thus, we need 42 bytes plus whatever space the variable-length fields themselves take.

### Exercise 3.4.2

The average name field takes 30 bytes, the average address 50 bytes, and the average history 500 bytes. Thus, the average length of a record is 42 + 30 + 50 + 500 = 622 bytes.

### Exercise 3.4.5

To start, when $r$ is much smaller than 1000, we can surely save by utilizing the extra space in blocks

for split records. However, let's see how close to $r = 1000$ we can still save. Without splitting records, we can only fit one record per block. Thus, we can save if we can fit two fragments in less than 1000 bytes, i.e., $r + 2*16 < 1000$, or $r < 968$.

To check, suppose $r = 967$. Then the first record leaves 33 bytes, so we can take a 16-byte fragment header and 17 bytes of the next record, and put it in the first block. That leaves 950 bytes plus a 16-byte header for the next block, leaving 34 bytes. Similarly, each split record leaves one more empty byte, until eventually there are 967 bytes free on a block and we can slip an extra record into an already-used block, thus saving (a very small amount of) space.

Return to Top

# Solutions for Section 3.5

## Exercise 3.5.1(a)

Method (i) requires only one disk read if we know the proper block for a record, while method (ii), with its addition of overflow blocks, requires more than one read if the desired record is on an overflock block. Thus, (i) is ``better'' than (ii). However, in the long run, (i) keeps blocks between full and half-full, so the average space utilization will be 75%. Method (ii) can keep all but the first block in a chain full, so for long chains the space utilization of (ii) is better while its average retrieval time goes down. In addition, (i) requires more index entries, and thus eventually more space for the index and more time to examine the index during each record lookup.

Return to Top