CHAPTER 10

Storage and File Structure

Practice Exercises

10.1 **Answer:** This arrangement has the problem that P_i and B_{4i-3} are on the same disk. So if that disk fails, reconstruction of B_{4i-3} is not possible, since data and parity are both lost.

10.2 **Answer:**

a. It is stored as an array containing physical page numbers, indexed by logical page numbers. This representation gives an overhead equal to the size of the page address for each page.

b. It takes 32 bits for every page or every 4096 bytes of storage. Hence, it takes 64 megabytes for the 64 gigabyte of flash storage.

c. If the mapping is such that, every p consecutive logical page numbers are mapped to p consecutive physical pages, we can store the mapping of the first page for every p pages. This reduces the in memory structure by a factor of p. Further, if p is an exponent of 2, we can avoid some of the least significant digits of the addresses stored.

10.3 **Answer:**

a. To ensure atomicity, a block write operation is carried out as follows:

   i. Write the information onto the first physical block.

   ii. When the first write completes successfully, write the same information onto the second physical block.

   iii. The output is declared completed only after the second write completes successfully.

   During recovery, each pair of physical blocks is examined. If both are identical and there is no detectable partial-write, then no further actions are necessary. If one block has been partially rewritten, then we replace its contents with the contents of the other block. If there

has been no partial-write, but they differ in content, then we replace the contents of the first block with the contents of the second, or vice versa. This recovery procedure ensures that a write to stable storage either succeeds completely (that is, updates both copies) or results in no change.
The requirement of comparing every corresponding pair of blocks during recovery is expensive to meet. We can reduce the cost greatly by keeping track of block writes that are in progress, using a small amount of nonvolatile RAM. On recovery, only blocks for which writes were in progress need to be compared.

b. The idea is similar here. For any block write, the information block is written first followed by the corresponding parity block. At the time of recovery, each set consisting of the n^{th} block of each of the disks is considered. If none of the blocks in the set have been partially-written, and the parity block contents are consistent with the contents of the information blocks, then no further action need be taken. If any block has been partially-written, it's contents are reconstructed using the other blocks. If no block has been partially-written, but the parity block contents do not agree with the information block contents, the parity block's contents are reconstructed.

10.4 Answer:

a. Although moving record 6 to the space for 5, and moving record 7 to the space for 6, is the most straightforward approach, it requires moving the most records, and involves the most accesses.

b. Moving record 7 to the space for 5 moves fewer records, but destroys any ordering in the file.

c. Marking the space for 5 as deleted preserves ordering and moves no records, but requires additional overhead to keep track of all of the free space in the file. This method may lead to too many "holes" in the file, which if not compacted from time to time, will affect performance because of reduced availability of contiguous free records.

10.5 Answer: (We use "↑ i" to denote a pointer to record "i".)
The original file of Figure 10.7.

header				↑ 1
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				↑ 4
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

a. The file after **insert** (24556, Turnamian, Finance, 98000).

header				↑ 4
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	24556	Turnamian	Finance	98000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

b. The file after **delete** record 2.

header				↑ 2
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	24556	Turnamian	Finance	98000
record 2				↑ 4
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

The free record chain could have alternatively been from the header to 4, from 4 to 2, and finally from 2 to 6.

c. The file after **insert** (34556, Thompson, Music, 67000).

header				↑ 4
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	24556	Turnamian	Finance	98000
record 2	34556	Thompson	Music	67000
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

10.6 Answer:

The relation *section* with three tuples is as follows.

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-347	1	Fall	2009	Taylor	3128	C

The relation *takes* with five students for each section is as follows.

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-347	1	Fall	2009	A
12345	CS-101	1	Fall	2009	C
17968	BIO-301	1	Summer	2010	null
23856	CS-347	1	Fall	2009	A
45678	CS-101	1	Fall	2009	F
54321	CS-101	1	Fall	2009	A-
54321	CS-347	1	Fall	2009	A
59762	BIO-301	1	Summer	2010	null
76543	CS-101	1	Fall	2009	A
76543	CS-347	1	Fall	2009	A
78546	BIO-301	1	Summer	2010	null
89729	BIO-301	1	Summer	2010	null
98988	BIO-301	1	Summer	2010	null

The multitable clustering for the above two instances can be taken as:

BIO-301	1	Summer	2010	Painter	514	A
17968	BIO-301	1	Summer	2010	null	
59762	BIO-301	1	Summer	2010	null	
78546	BIO-301	1	Summer	2010	null	
89729	BIO-301	1	Summer	2010	null	
98988	BIO-301	1	Summer	2010	null	
CS-101	1	Fall	2009	Packard	101	H
00128	CS-101	1	Fall	2009	A	
12345	CS-101	1	Fall	2009	C	
45678	CS-101	1	Fall	2009	F	
54321	CS-101	1	Fall	2009	A-	
76543	CS-101	1	Fall	2009	A	
CS-347	1	Fall	2009	Taylor	3128	C
00128	CS-347	1	Fall	2009	A-	
12345	CS-347	1	Fall	2009	A	
23856	CS-347	1	Fall	2009	A	
54321	CS-347	1	Fall	2009	A	
76543	CS-347	1	Fall	2009	A	

10.7 **Answer:**

a. Everytime a record is inserted/deleted, check if the usage of the block has changed levels. In that case, update the corresponding

bits. Note that we don't need to access the bitmaps at all unless the usage crosses a boundary, so in most of the cases there is no overhead.

b. When free space for a large record or a set of records is sought, then multiple free list entries may have to be scanned before finding a proper sized one, so overheads are much higher. With bitmaps, one page of bitmap can store free info for many pages, so I/O spent for finding free space is minimal. Similarly, when a whole block or a large part of it is deleted, bitmap technique is more convenient for updating free space information.

10.8 **Answer:** Hash table is the common option for large database buffers. The hash function helps in locating the appropriate bucket, on which linear search is performed.

10.9 **Answer:**

a. MRU is preferable to LRU where $R_1 \bowtie R_2$ is computed by using a nested-loop processing strategy where each tuple in R_2 must be compared to each block in R_1. After the first tuple of R_2 is processed, the next needed block is the first one in R_1. However, since it is the least recently used, the LRU buffer management strategy would replace that block if a new block was needed by the system.

b. LRU is preferable to MRU where $R_1 \bowtie R_2$ is computed by sorting the relations by join values and then comparing the values by proceeding through the relations. Due to duplicate join values, it may be necessary to "back-up" in one of the relations. This "backing-up" could cross a block boundary into the most recently used block, which would have been replaced by a system using MRU buffer management, if a new block was needed.

Under MRU, some unused blocks may remain in memory forever. In practice, MRU can be used only in special situations like that of the nested-loop strategy discussed in Exercise 10.9a.