CHAPTER 13

# Data Storage Structures

Solutions for the Practice Exercises of Chapter 13

## Practice Exercises

**13.1**

**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 it 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 the reduced availability of contiguous free records.

**13.2**

**Answer:**
We use "↑ *i*" to denote a pointer to record "*i*".

a. See Figure 13.101.

b. See Figure 13.102. Note that the free record chain could have alternatively been from the header to 4, from 4 to 2, and finally from 2 to 6.

c. See Figure 13.103.

| | | | | |
|---|---|---|---|---|
| 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 |

**Figure 13.101** The file after **insert** (24556, Turnamian, Finance, 98000).

| | | | | |
|---|---|---|---|---|
| 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 |

**Figure 13.102** The file after **delete** record 2.

## 13.3

**Answer:**

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

| | | | | |
|---|---|---|---|---|
| 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 |

**Figure 13.103** The file after **insert** (34556, Thompson, Music, 67000).

| course_id | sec_id | semester | year | building | room |
|---|---|---|---|---|---|
| BIO-301 | 1 | Summer | 2010 | Painter | 514 |
| CS-101 | 1 | Fall | 2009 | Packard | 101 |
| CS-347 | 1 | Fall | 2009 | Taylor | 3128 |

The relation *takes* with five students for each section is as follows:
See Figure 13.104.

See Figure 13.105.
The multitable clustering for the above two instances can be taken as:

**13.4**

**Answer:**

a. The space used is less with 2 bits, and the number of times the free-space map needs to be updated decreases significantly, since many inserts/deletes do not result in any change in the free-space map. However, we have only an approximate idea of the free space available, which could lead both to wasted space and/or to increased search cost for finding free space for a record.

b. Every time 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.

| 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 |

**Figure 13.104** The relation *takes* with five students for each section.

c. 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 a proper-sized one is found, 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.

**13.5**

**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.

**13.6**

**Answer:**

The table can be partitioned on (year, semester). Old *takes* records that are no longer accessed frequently can be stored on magnetic disk, while newer records can be stored on SSD. Queries that specify a year can be answered without reading records for other years.

A drawback is that queries that fetch records corresponding to multiple years will have a higher overhead, since the records may be partitioned across different relations and disk blocks.

| | | | | | |
|---|---|---|---|---|---|
| BIO-301 | 1 | Summer | 2010 | Painter | 5 |
| 17968 | BIO-301 | 1 | Summer | 2010 | n |
| 59762 | BIO-301 | 1 | Summer | 2010 | n |
| 78546 | BIO-301 | 1 | Summer | 2010 | n |
| 89729 | BIO-301 | 1 | Summer | 2010 | n |
| 98988 | BIO-301 | 1 | Summer | 2010 | n |
| CS-101 | 1 | Fall | 2009 | Packard | 1 |
| 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 | 3 |
| 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 |

**Figure 13.105** The multitable clustering for the above two instances can be taken as:

**13.7**

**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 Section 13.8a.

**13.8**

**Answer:**

The database system does not know what are the memory demands from other processes. By using a small buffer, PostgreSQL ensures that it does not grab too much of main memory. But at the same time, even if a block is evicted from buffer, if the file system buffer manager has enough memory allocated to it, the evicted page is likely to still be cached in the file system buffer. Thus, a database buffer miss is often not very expensive since the block is still in the file system buffer.

The drawback of this approach is that the database system may not be able to control the file system buffer replacement policy. Thus, the operating system may make suboptimal decisions on what to evict from the file system buffer.