CS 186
Fall 2024

Introduction to Database Systems
Alvin Cheung

DIS 2

1 True and False

(a) When querying for a 16 byte record, exactly 16 bytes of data are read from disk.
False, an entire page of data is read from disk.

(b) In a heap file, all pages must be filled to capacity except the last page.
False, there is no such requirement.

(c) Assuming integers take 4 bytes and pointers take 4 bytes, a slot directory that is 512 bytes can address 64 records in a page.
False, we have the free space pointer, which doesn't fit after 64 * (4 + 4) = 512 bytes of per-record data in the slot directory.

(d) In a page containing fixed-length records with no nullable fields, the size of the bitmap never changes.
True, the size of the records is fixed, so the number we can fit on a page is fixed.

2 Fragmentation And Record Formats

(a) Is fragmentation an issue with fixed length records on an unpacked page?
No. Since all records are the same size (fixed), we'll never encounter a case where we can't add a record to a page even though the record's size is less than or equal to the total amount of free space on the page.

(b) Is fragmentation an issue with variable length records on an unpacked page?
Yes. For example, even if a page has n bytes of space dispersed throughout, we may not be able to add a n-byte record because the space is not consolidated together. This can happen because variable length records can be of any size, causing gaps of varying sizes upon deleting them.

3 Record Formats

Assume we have a table that looks like this:

```
CREATE TABLE Questions (
      qid integer PRIMARY KEY,
      answer integer NOT NULL,
      qtext text NOT NULL,
);
```

Recall that integers and pointers are 4 bytes long. Assume for this question that the record header stores pointers to all of the variable length fields (but that is all that is in the record header).

(a) How many bytes will the smallest possible record be?

12 bytes. The record header will only contain one pointer to the end of the qtext field so it will only be 4 bytes long. The qid and answer fields are both integers so they are 4 bytes long, and in the smallest case qtext will be 0 bytes. This gives us a total of 12 bytes.

(b) Assume we have a page that is 1 KiB in size. What is the maximum number of records that can fit on a page?

50 records. If we have 1 KiB for data, this means we have 1024 bytes to use for all the records and the page footer slot directory. The slot directory always stores the slot count (4 bytes) and a free space pointer (4 bytes). Thus, 1024 bytes - 4 bytes (slot count) - 4 bytes (free space pointer) = 1016 bytes remaining for records and corresponding slots. In addition, for each record we store on the page, it has a corresponding slot in the page footer that takes up 8 bytes (4 byte record pointer and 4 byte record length). The minimum record size (12 bytes) and each slot takes 8 bytes, so we divide 1016 by 20. Since we can't have fractional records on a page, we round down to the nearest record. This is equal to floor(1016 / 20) = 50.

(c) Now assume the fields can be null so we add a bitmap to the beginning of our record header indicating whether or not each field is null. Assume this bitmap is padded so that it takes up a whole number of bytes (i.e. if the bitmap is 10 bits it will take up 2 full bytes). How big is the largest possible record assuming that the qtext is null?

13 bytes. The qid cannot be null because it is a primary key so there will be 2 slots in our bitmap (and thus 2 bits), meaning our record header will only be 1 byte longer than it was in part a. In the max case, both the qid and answer field will be present and still be 4 bytes each. Therefore, the fields haven't changed at all, and the total record length is now 13 bytes.

4 Calculate the I/Os with Linked List Implementation

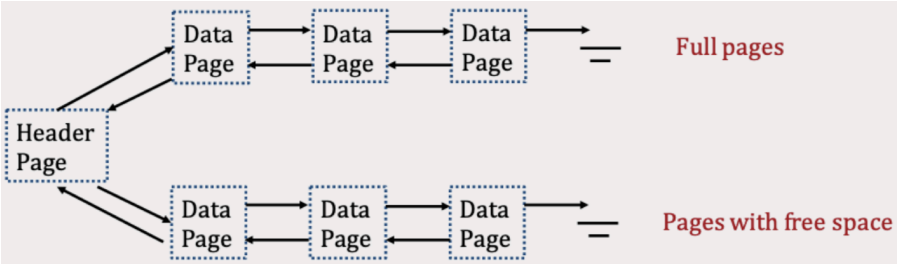
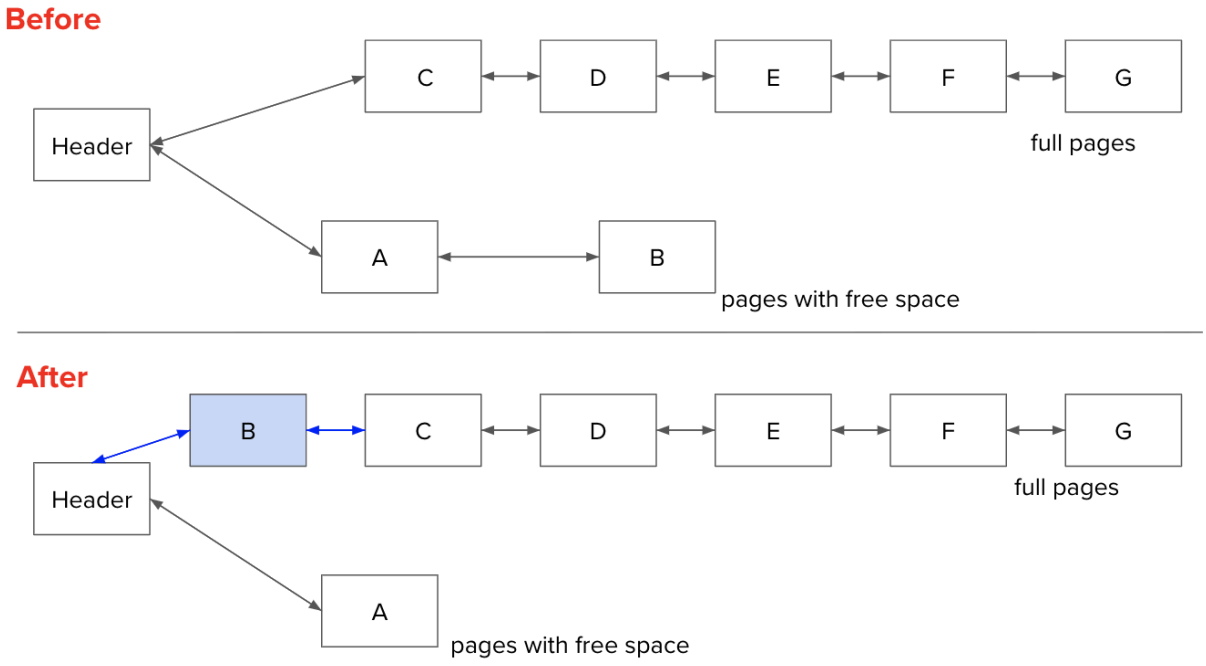


Figure 1: Linked List Implementation

Assume we have a heap file implemented with a linked list. The heap file contains 5 full pages and 2 pages with free space, at least one of which has enough space to fit the record we are trying to insert.

(a) In the worst case, how many I/Os are required to find a page with enough free space to insert a record?
3 I/Os: We incur 1 I/O to read in the header page. In the worst case, only the final page in the list of pages with free space has enough space for this record. Thus, we read in the 2 pages in the list of pages with free space, incurring 2 I/Os.

(b) In the worst case, how many I/Os are required to insert a record into the 2nd page with free space? Consider what happens when after inserting the record, this page becomes full. Also, assume that we can modify the header page and insert to the beginning of the full pages linked list.



8 I/Os:

- Read header page (1 I/O)

- Read page A (1 I/O)
- Read page B (1 I/O)
- Add record to page B, change left pointer to refer to header page, change right pointer to refer to page C, write page B to disk (1 I/O)
- Change page A's right pointer to no longer refer to page B, write page A (1 I/O)
- Change header page's pointer for list of full pages to refer to page B, write header page (1 I/O)
- Read page C (1 I/O), change left pointer to refer to page B, write page C (1 I/O)

5 Calculate the I/Os with Page Directory Implementation

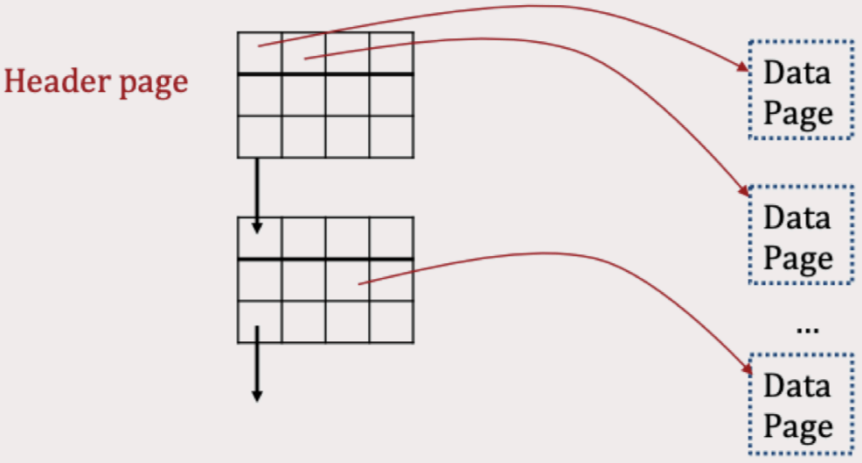


Figure 2: Page Directory Implementation

We have a heap file implemented with a page directory containing 54 data pages. Each page in the directory can hold 16 page entries. Assume we have fixed length records.

(a) In the worst case, how many I/Os are required to find a page with free space?
4 I/Os. The page directory will contain ceil(54/16) = 4 header pages. In the worst case, we will need to look through all 4 pages in the directory, incurring a total cost of 4 I/Os.

Note that the page directory contains the amount of free space for each page, so we do not need to read in the data pages to check whether they contain free space.

(b) In the worst case, how many I/Os are required to insert a record into this file? Assume at least one data page has enough space to fit this record.
7 I/Os. From the previous part, we need 4 I/Os to find a data page with enough free space to fit this record (in the worst case). Then, we need to read the page with enough free space into memory (1 I/O), insert the record, and write this page back to disk (1 I/O). Lastly, we need to update the amount of free space for this data page in our page directory, and write the page in the directory back to disk (1 I/O).