CS 186 Introduction to Database Systems
Fall 2020 Alvin Cheung, Aditya Parameswaran

DIS 2

1 True and False

Solution:

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

(b) Writing to an SSD drive is more costly than reading from an SSD drive.
True, a write can involve reorganization to avoid uneven wear and tear (wear leveling).

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

(d) 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.

(e) 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.

Which of the following are true about the benefits of using a record header for variable length records?

(a) Does not need a delimiter character to separate fields in the records
True, using a record header eliminates this requirement.

(b) Always matches or beats space cost when compared to fixed-length record format
False, extra space is required for the record header.

(c) Can access any field without scanning the entire record
True, can calculate position of any field using arithmetic.

(d) Has compact representation of null values
True, null values don't take any space (except the pointer in the record header).

2 Fragmentation And Record Formats

Solution:

(a) Is fragmentation an issue with packed fixed length record page format?
No, records are compacted upon deletion.

(b) Is fragmentation an issue with variable length records on a slotted page?
Yes.

(c) We usually use bitmaps for pages with fixed-length records. Why not just use a slotted page for pages with fixed-length records?
Bitmaps have better space complexity since slotted pages use pointers which are addresses.

3 Record Formats

Assume we have a table that looks like this:

```
CREATE TABLE Questions (
    qid integer PRIMARY KEY,
    answer integer,
    qtext text,
);
```

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).

Solution:

(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) Now assume each field is nullable 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. We have 3 fields so there will be 3 slots in our bitmap (and thus 3 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 IOs with Linked List Implementation

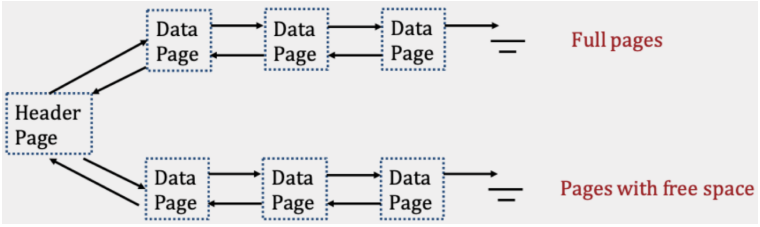

Figure 1: Linked List Implementation

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

Solution:

(a) In the worst case, how many IOs are required to find a page with enough free space to insert a record?
3 I/Os, you read in the header and then the 2 pages in the free pages linked list before finding a page with enough free space in the final page.

(b) In the worst case, how many IOs are required to write a record to the 2nd page with free space? Consider what happens when after writing, the page becomes full and assume that the header page can insert at the beginning of the full pages linked list.
8 I/Os. From the first part, you need 3 I/Os to find the final page. Then you need 1 I/O to write the record to the final page, 1 I/O to change the pointer from the previous free page, 1 I/O to change the header page (since the final page is now full and in the front of the full pages), and 2 I/Os to change the previous front of the full pages (read from disk, modify, and write to disk).

5 Calculate the IOs with Page Directory Implementation

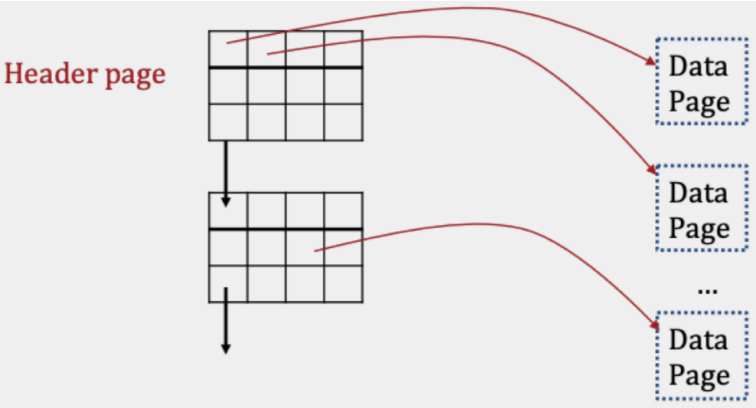

Figure 2: Page Directory Implementation

Assume we have a heap file B implemented with a page directory. One page in the directory can hold 16 page entries. There are 54 pages in file A in total.

Solution:

(a) In the worst case, how many IOs are required to find a page with free space?
4 I/Os. There will be ceil(54/16) = 4 pages in the directory. In the worst case, we will need to look through all 4 pages in the directory. (The directory contains the amount of free space for each page.)

(b) In the worst case, how many IOs are required to write a record to a page with free space (assuming at least one free page with enough space to insert a record exists)?
From part a, we need 4 I/Os to find the free page. We also have to read the page with the free space (1 I/O), write a record to that free page (1 I/O), and write to the page directory (1 I/O), for a total of 7 I/Os.