Notes:

**Physical Layer**

* The way data is stored in the physical layer is different than the way it is shown to the user as the schema
  + This is because it is more optimal for the machine to store it as binary
  + Users do not want to see binary,
  + We need to figure out how to map bytes to the schema (and vice vIn ersa)
* What kind of data is stored in a DBMS? Metadata and the actual data
  + Metadata is also stored in a table (tracks relationships between tables, the pk of a table, etc…)
* What is the criteria for data storage?
  + Size and speed (there is a tradeoff between the two)

**Data Types**

* Strings
  + Varchar: allows you to store a variable amount of characters
    - Define length of 10, but it’s only 5, it will only use 5 spaces
    - Variable length
  + Char: similar, but the length is fixed no matter what
    - Define length 10, but it’s only 5, it will append extra spaces to make it length 10
    - Fixed length
  + We separate into these different categories:
    - Tradeoff between size and speed
      * Fixed length: we can easily skip around when searching (will always be same length), but uses more space
      * Varied length: cannot easily skip around, but space is saved
* Date
  + Under the hood: dates are just numbers (integers)
* Boolean
  + Represent using a BIT data type
    - BIT is just a very small integer (as small as one can be, either 1 or 0)
* Integers, floating point, etc… (numbers)
* Data types have an effect on storage

**Records**

* If I want to know how a record is stored on a disk, we must understand the structure of the record
  + Data types of the column affect the size of the record
  + Not all records in the same table will be the same size
    - Could pick data types where all records are the same size (like ints)
    - Or you could pick variable length data types and then records will be variable length
  + Having variable length column in record makes it difficult to know where that column is, but it also makes it more difficult to figure out where the ends of that record is
* How to store variable length records/fields:
  + Use a termination
    - Ex: ‘hello$’ (where $ is the termination character)
    - This character might take up a byte
  + Whereas if you track the offset (start of the data) might use more space, because you are now storing an int (more bytes, but could make it easier to figure out where things are)
  + Or do something like this: ‘5hello’, because ‘hello’ has 5 characters so we know how long the data is
  + Could use any of these methods in conjunction

**Operations**

* What operations will be performed on these files? Read, insert, and delete
* The way we store our data has a large effect on these operations
  + Heap file: throw data at end
    - Downside of this approach: eventually, when data gets deleted there will be a bunch of gaps in the file
    - Maybe incorporate a header that tracks whether certain positions are occupied or not, this way we no longer need to tack at the end, just find the first empty space
      * Using a header assumes records are fixed length

**Heap Files**

* Simplest format (MySQL probably isn’t using heap files)
  + Records are unordered
* Insertion:
  + Append to end: O(1)
  + Use head and use space appropriately: O(n)
    - n is number of records
    - Traversing header, finding an open slot
* Search:
  + Using header: O(n), based on number of records
    - With header, we can actually do a little bit better than O(n) because we can skip over empty spaces
* Modify/delete:
  + Find the data first, then just “mark the bit in the header”
  + Or… what if the record knows where it lives
    - Then to delete just directly modify the header
    - To modify an existing record, check if the data will still fit in the same space
      * So then we need to check space first, and then insert. If it does not fit, then delete and then insert the data again
      * This is usually what happens
* Is it faster to manipulate data on disk or in memory?
  + Memory is much faster than disk
  + Must read in our physical data to memory to make changes to it
  + One file on disk per table in database
    - So as DB grows, the file grows larger and does not fit in memory anymore
    - To solve this, read by chunk, called pages

**Heap Pages**

* Can have multiple pages in memory (does not have to be one at a time)
* Files will have many heap pages, and each page will have its own header, and tuples (records)
  + Tuple: ordered set of values
* Are constant in size
  + How does this affect the design of your records?
    - If you want to append a record, but the heap page is full:
      * We can split record up and store in multiple heap pages (span record)
      * Or create a new heap page (wasted space)
    - Heap file’s job is to manage the heap pages (keep track of pages, where to access those pages, and create newpages)
    - The heap pages are the ones actually putting records in and taking them out
      * Manage the tuples (the records/rows in the tables)
* Heap file does not need a header (heap pages do though)
  + This is because heap pages are fixed size, and the file knows its size, then we can easily get the number of pages

**Exercise**

* Given a heap page of size m and a record of size n, determine how many records can fit on the page. Hint: don't forget the header!
  + Missing size of the header:
    - Size of header: x where x is the number of records
    - A record requires 1 bit in the header
    - Record is of size n+1
    - m\*8 / (n\*8 + 1)
      * If m and n were in units of bits, then we no longer need the factor of 8 -> m / (n+1)
* Develop pseudocode for adding a record to a heap file. The input should be a tuple.
  + Have to find a page with space available

foreach page in file:

if page has open slot:

add tuple to page

create new page

add the tuple to the page

* + Traverse header, find first 0, got to position in file and update with data
  + If there are no 0’s, check if there is enough space at end of record, and act accordingly (append to end, or new heap page)