1. What is a B-Tree and how does it work?  Explain and give an example. (3 pts)

A B-Tree is a ‘balanced’ tree – a data structure commonly used in relational database indexing. B-Trees are key-value pairs that have been sorted by key, giving it the advantage of fast lookup. This is especially helpful in OLAP databases, which need to be read quickly when at scale. The structure of a B-Tree is such that there are fixed-size pages that contain references to other pages and to keys. A root page in a B-tree is one that contains references to keys and child pages (which themselves store several more keys).

A common example of a B-tree at work is a date field – a page may refer to a year-month by which a user can sort and pull data. In an OLAP setting, the advantage to using a B-tree with dates as keys is that when pulling data for a given timeframe, it minimizes reference lookups from the root page to the child pages containing the key-value pairs.

1. What is Column Oriented Storage and how is it different than an RDBMS?  Give an example.  (3 pts)

Column oriented storage essentially inverts the row-oriented structure of a normal RDBMS table. When writing to the database, the values are grouped by column (instead of by row) and are accessed to create a cohesive record by their indices. This helps when selecting a few fields from many, saving the read time from accessing all columns in a *SELECT \** operation. A second key difference between column-oriented and RDBMS is that when deleting, all records at a given index must be deleted, or else values that come after will have a mismatch with other “tables” and the data will be useless.

A good use case for column-oriented storage is when some values are repeated; by using reference tables, we can cut down the amount of space the information actually takes up by encoding it. Highly-dimensional tables can take advantage of column-store by limiting the amount of work that has to be done to access the data.

1. What is an SSTable and how does it work?  What are some of the advantages over log segments with hash indexes?  (3 pts)

An SSTable is a storage of key-value pairs that are stored in ‘segments’. These segments may contain duplicate or updated keys, that can then be compacted to only reflect the latest state of a table. An advantage of using SSTables over log segments with hash indexes is that segment merging is simpler since it can compact data that exceeds available memory. Accessing specific keys is also easier because of the sparse memory allocation of the SSTable – it looks for an approximate match on the key in order to scan the keys that follow in order to access the byte offset.

1. What is tail recursion and how is it different than regular recursion?  (1 pt)

Tail recursion is specific in that the recursive call is the last execution from the function. This essentially eliminates the call stack by storing the intermediate value as a returned value. In regular recursion, there is a call stack that needs to be traced back once the base case is achieved because the recursive function calls itself. With tail recursion, the call stack is minimized because the values swap in place (that is, the call stack doesn’t grow because it returns at the same time a recursive call is made, avoiding the necessity of putting the original call frame in memory).