* Discuss the advantages and disadvantages of using (a) an unordered file, (b) an ordered file, and (c) a statichash file with buckets and chaining.
* Share your experience or research on database file structures for a targeted RDBMS that you have used in the past.In general, is database file stored transactions a heap (unordered) or ordered file? Why?
* Share any experience or research on creating indexes including types of indexes and outcomes of implementing them. Share your criteria to create indexes.
* Share your experience or research on query performance improvement using indexes. Did you create an index when you create a table, or you add an index as your database needs one?
* Share your experience or research on creating special indexes such as bitmap, functional-based, reverse indexes.

Unordered files are the simplest type of file organization, records are generally ordered in the order they were uploaded in the system, which allows for very fast insert. The disadvantage with unordered files is that search is usually much slower with unordered than ordered, where search can take O(n) time. Sorted files can provide faster search, especially using BST which can reduce search time to O(lgn), however will take longer to insert records. Using static has files can provide very fast insert (almost as fast as unordered) and fast search in some conditions. Some drawbacks to static hash files include additional complexity to implement and can show some inefficiencies for large databases.

I have no experience with database file structures, however I gather that depending on the size of the database you may want to try different things. For small databases, like the database project, it may be easiest to just use an unordered file. For smaller databases where search time is critical, a hashing technique can be employed to improve search time. For larger databases, it is likely that an ordered file would be best.

Indexes can be extremely useful in data retrieval, however the increased overhead of the indexes needs to be considered before implementation. Indexes should be avoided for small indexes or if queries comprise 10-15+% of the rows. Indexes should also be avoided for tables where the rows are updated often, as the indexes will also have to be updated. However, if these conditions are met, then indexes can greatly improve data retrieval. Indexes are even more useful in data retrieval when the query involves a JOIN or WHERE clause.

I have no experience with special indexes like bitmap or functional based or reverse indexes. I learned that function based indexes are appealing when handling computationally expensive functions, and can be precomputed and stored before queries. Bitmap indexes are helpful in data warehouse applications and are especially useful for queries that contain EXIST, UNIQUE or GROUP BY.