In this report, I take sf100 dataset as the example dataset since it is much smaller, easier to analyze, and also a good representation of sf3000 dataset.

**1. Reducing the problem** In this part, I reduced the size of the problem. First, I removed person's friends(in knows.bin) who live in a different city. Second, I removed persons(in person.bin) with zero friends left over, and then update the interests.bin. Third, I removed non-mutual friends. In a word, we removed the redundant data in advance by using the query's filter conditions. By doing this, we do not do the corresponding filtering at query time, as all the data now satisfy the condition. As a result, the query time significantly decreased. In my case, it reduced the average time for a query from more than 100 seconds to 2 seconds, the number of person from 499000 to 212155, the size of interests.bin from 11692172 to 5919808, and the size of knows.bin from 46398896 to 1271695.

**2. Building the inverted index** In this part, I built the inverted index for artists, which significantly speed up the query. In this assignment, the inverted index is an index storing a mapping from artist to person. After reducing the problem, we still have 252155 person in the person.bin. In fact, we do not need to access all the person in each query. This is because for each query we only care about person who is related to four artists(A1, A2, A3, and A4) and their friends. After narrowing the search area by artists, we can significantly reduce the size of person we access. For example, in the sf100 dataset, we have 15197 artists. By leveraging the inverted index, we only need to access 4 artists, which have much less data.

In my opinion, the most difficult part of building an efficient inverted index is implementing the union-sum operation. I tried two methods, one is to merge the three person lists, remove the duplicate person and compute their score, which has complex boundary cases, the other is to merge the three person lists but don't remove the duplicate person, which has slower performance but it is easy to implement. In a word, the inverted index is a good method to speed up the query. In my case, it reduced the average time for a query from two seconds to 0.46 seconds.

**3. Partitioning birthday** In this part, I tried to partition birthday data into 366 different blocks and build an index for them. However, there is a bug in my program and I cannot fix it. The reason why I want to build an index for birthday is that we just need to access a small part of person who was born in a small range of dates. We do not need to access all person in our dataset. By partitioning the birthday, we can narrow the scope of the query and thus speed it up.

**4. Columnar storage** I also studied columnar storage. When we only care about several columns of the dataset, using this format is faster than row-major storage. This is because it reduces the amount of data transferred from disk into memory and subsequently from memory into CPU memory. In this assignment, I stored every column of original datasets in different data files. However, in my case, I did not see a big improvement. I guess this is because the size of the data is small and we actually need to access every column of the data.