We considered buffer pool and parallel hash join for optimization. We implemented buffer pool, yet the performance was worse than without using that. We created a BufferPool class, which upon instantiation, creating a pool of buffers of size 4096 bytes. It has methods to get a free buffer and return a buffer that has been used back to the buffer pool. These two methods are implemented to be thread-safe with locks. In theory, the program would have less allocating and garbage collection cost if a buffer pool is used and the size of the buffer pool is properly tuned. I think the performance is worse because of some unknown problem in our implementation. We decided not to include our code since it's not a very important feature. Besides, we considered parallel hash join but we didn't have enough time to implement that. We read through the algorithms in the Kitsuregawa paper and the API for parallelism. We are not familiar previously with the Java 8 features introduced in the parallelism API, and other courses also have a lot of work, so we could not implement it in time.