Lab 3 **Hanjing Wang(516030910015)**
Database System ACM Class, Zhiyuan College, SJTU
Prof. **Feifei Li** Due Date: May 23, 2019

Design decisions

simpledb.Join

This class is a implementation of **Block Nested-Loops Join**. The mothed optimized the plain Nested-Loops Join by caching the outer relation block, which could significantly reduce the IO costs by a constant factor, ideally the IO costs of loading the cached block. The cache size is controlled by the argument **Join.BLOCKSIZE**.

simpledb.HashEquiJoin

In this class, students are required to finish a **Hash Join** alogorithm. Instead of caching the outer relation block, we built a hash table of the inner relation. The join operation could be done more efficiently since the matching and filering procedure are replaced by the calculation of hash value. However, we made a assumption here that the inner relation can be stored in the memory entirely. Because if we need to implement a block hash join algorithm, then we need to explicitly design a hash algorithm applying to strings and integers, which is tricky and not the central consideration of the lab exercise.

simpledb.IntegerAggregator/StringAggregator

simpledb.IntegerAggregator.Rec, considered as the wrapper of a certain group, is introduced here. First, we create a hash table of groups according to the value of their groupby field. When the merge method is called, the group information of the new tuple is determine, and then the corresponding group wrapper executes merging operation. Finally, a tuple list is contructed based on the attributes of the wrappers and its iterator is returned.

Difficulties

I spent roughly 12 hours on the lab3, and the most time-consuming part is the implementation of Hash Join algorithm. There are some confusing points in both the JavaDocs and the annotations from my personal persepective. For example, according to the annotations of

HeapPage.java, if a tuple is deleted from a certain page then it has to be updated indicating it no longer belongs to the page. The most straightforward way for me would be setting its RecordId to be null after deletion. Hoewever, in the test case, the RecordId of a tuple is called after its deletion. It would be better if the JavaDocs could be more detailed.