CS539 Assignment 2

SimpleDB Join algorithms (SNL, PNL and SMJ)

Ashish Jindal (AJ523)

1 Design Decisions

1.1 Separation of Join Algorithm's implementation using Factory design

Currently all the join algorithms implementation is expected to be in one class Join, I decided to separate there implementation into multiple classes as different join algorithms required different bookkeeping methodologies which might be useful for one algorithms but redundant for others. This way we can also avoid long if-else or switch-case chains over joinType in class Join. I created an abstract class AbstractJoin which extends DbIterator and moved the common functionality among join algorithms to this abstract class and then other join algorithms simple extend this abstract and implement the remaining methods. After implementing multiple join classes I decided to use the existing class Join as a factory class which will update the instance of join algorithm according to requirement and rest of the functions of this class simple call that join algorithms methods. This way I also avoided making changes to api expected by join test cases as they expect a class Join rather my implementation of AbstractJoin.

1.2 Implementation of BufferedDBIterator for Page wise iteration over DB

Page nested loop requires page wise iteration over inner and outer relation. DBIterator (HeapFileIterator) implements this, but doesn't expose the notion of Page through its api. So I decided implement a wrapper over DBIterator "BufferedDbIterator", which will read one page at a time using the underlying DBIterator instead of one Tuple. But this also required a way to store the content of extracted page temporarily. For this I modified the existing HeapPage class and implemented the "addTuple()" method in this class to facilitate using of HeapPape instance as a temporary buffer for Page wise iteration of DBFile. So "BufferedDbIterator" gives us the next page and then HeapPage returned already has a tuple iterator implementation which i used to iterate over its tuples.

1.3 Additional method for HeapFileIterator - seek()

HashMap provide O(1) access for all the elements in the data-structure so I decided to use it for storing our buffered pages. We can check for existence of a page in buffer pool in O(1), fetch the same in O(1) and also insert or remove an element from it in constant time.

2 Feedback on assignment

Assignment gave a great insight into the practical implementation of DB Catalogue, heap file, buffer pool and buffer replacement policies. Approximately it took me around 7hrs to finish the assignment; which includes time spent on code analysis, debugging and implementing the desired code features. One thing i struggled with was understanding the memory allocation for header in heap file. I think the implementation of header (Store bits in integers) and related arithmetic on the bits in the header, was scattered all over the file, which made it a bit confusing to analyse.