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 algorithm's 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 currently used join algorithm according to requirement and rest of the functions of this class simply call the active join algorithm's methods. This way I also avoided making changes to the 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 DB File. 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. Page nested loop could also have been implemented using the seek() described below with some additional book keeping, but i think the current implementation of using BufferedDBIterator is more clean and extendible.

1.3 Additional method for HeapFileIterator - seek()

HeapFileIterator allows to sequentially go backward and forward from current position, but currently there is no way to go to a particular record position. So I implemented a function "Seek()" which takes a record ID as input parameter and using the page number and slot id it then goes to the corresponding tuple's location in DB File. Using this functionality we can go back to any known record in DB file as needed in sort merge join implementation to handle the case of duplicate outer relation join attributes.

2 Feedback on assignment

Assignment gave a great insight into implementation of join algorithms using an iterator based design. Approximately it took me around 30hrs to implement the 3 join algorithms. I struggled with a lot of boundary case scenarios while implementing joins. Other thing that i found a bit misleading was using DBIterator for sequential scan and a DBFileIterator for iterating over heap file. The two seam to be a bit synonymous as sequential scan is just a wrapper over HeapFileIterator.