CSE 444: `SimpleDB` Final Report

Linxing Preston Jiang   Winter 2018

March 12, 2018

1 Overall System Architecture

In a typical database architecture, there are four main components: Process Manager, Query Executor, Share Utilities, and Storage Manger (Balazinska, Mass, Lecture 3). For our implementation of `SimpleDB` in the labs, we focus on the Storage Manager and Query Executor: adding access methods for data store on disk in lab1, adding both file mutability (insert/delete tuples to/from file system, eviction from full BufferPool) and query operators (SeqScan, Join, etc.) & aggregates (min, max, etc.) in lab2, adding lock manager in lab3, adding log manager in lab4, and finally, adding parallel data processing in lab6.

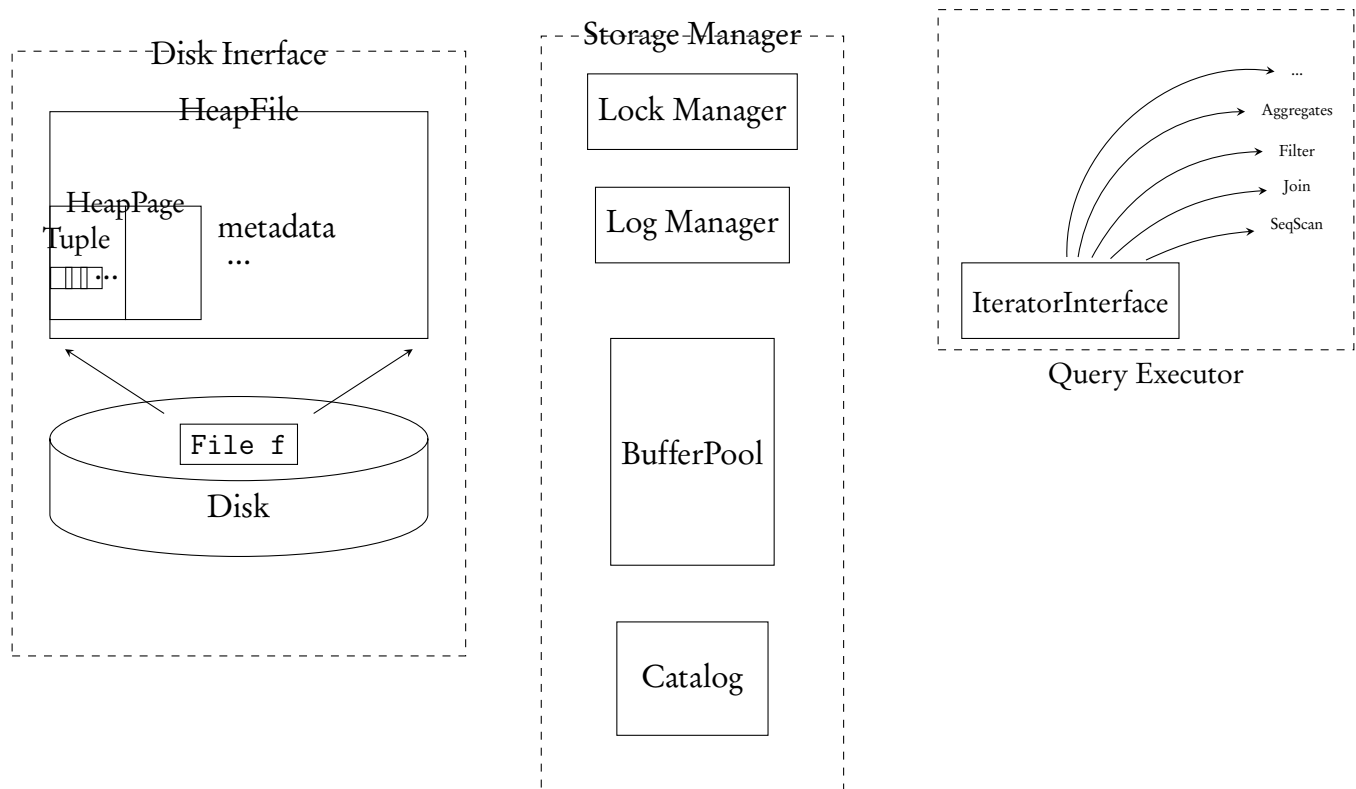Figure 1 shows the parts of the architecture of `SimpleDB` which we implemented in the labs.
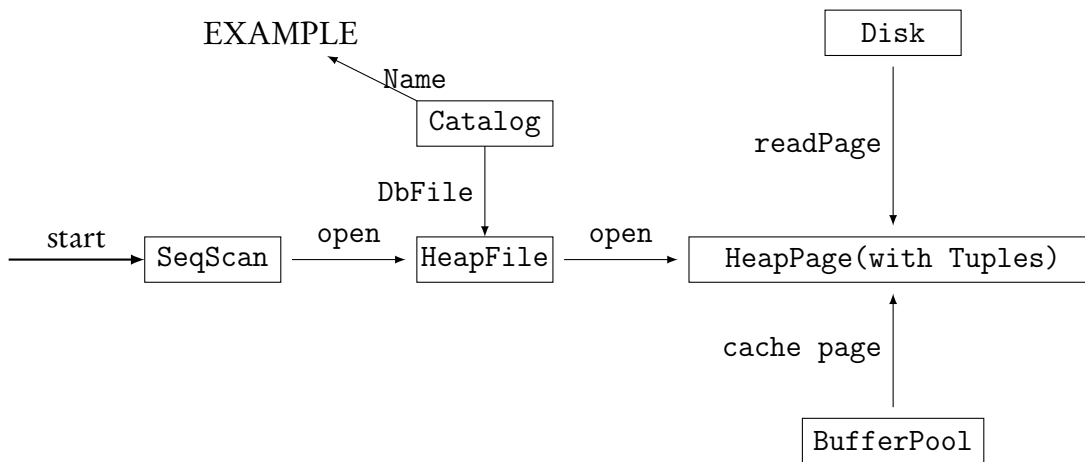

Figure 1: `SimpleDB` Architecture

Figure 2: `SimpleDB open()`

1.1 `BufferPool` **and** `Opertors`

BufferPool is responsible for both caching pages in memory that have been recently read from disk and handle concurrency and transactions. All operators read and write pages from various files on disk through the buffer pool. Operators are responsible for executing query plans. In `SimpleDB`, each operator implements the `OpIterator` interface, which supports `open`, `hasNext`, `next`, `rewind`, and `close`. Operators are connected together into a plan by passing lower-level operators into the constructors of higher-level operators (Lab1 ReadMe). Programs call `next` on the root operator and then fetch tuples recursively through the plan tree in one pass top-down and another pass bottom-up.
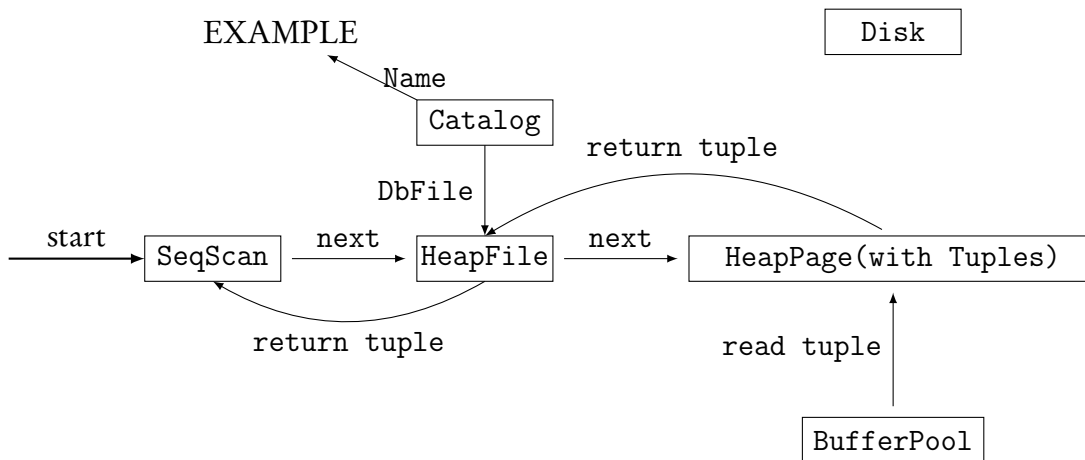

Figure 3: `SimpleDB next()`

In Lab 1, we implemented `getPage` which is needed for reading Pages into memory and getting tuples, together with `SeqScan` operator to scan the file and return tuples. The workflow of opening opertators to read file and caching pages into BufferPool is shown by Figure 2, and the workflow of recursively getting tuples through `SeqScan` using `next()` is shown by Figure 3.

In Lab 2, We added `insert`, `delete` funcionalities in `SimpleDB`, which insert/delete tuples to/from the
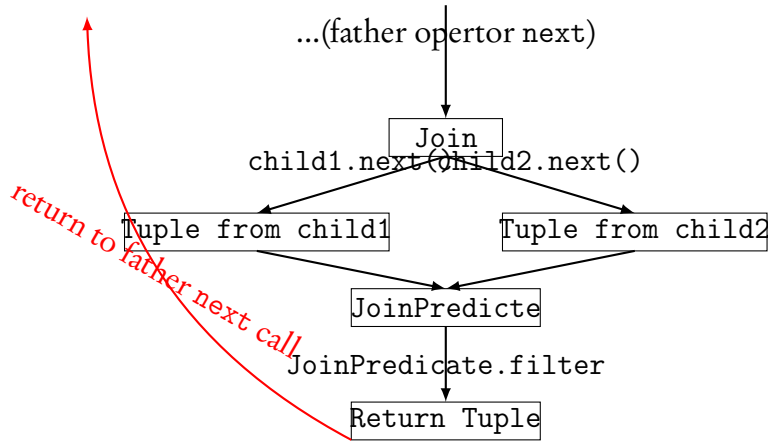
...(father opertor next)

Join
child1.next(child2.next()

Tuple from child1   Tuple from child2

JoinPredicte

JoinPredicate.filter

Return Tuple

return to father next call

Figure 4: `SimpleDB join`

insert

BufferPool

insert

HeapFile

insert

Page in BufferPool?

yes   no

Return Page   pull a new page (evict if needed)

insertTuple   insertTuple

updated HeapPage

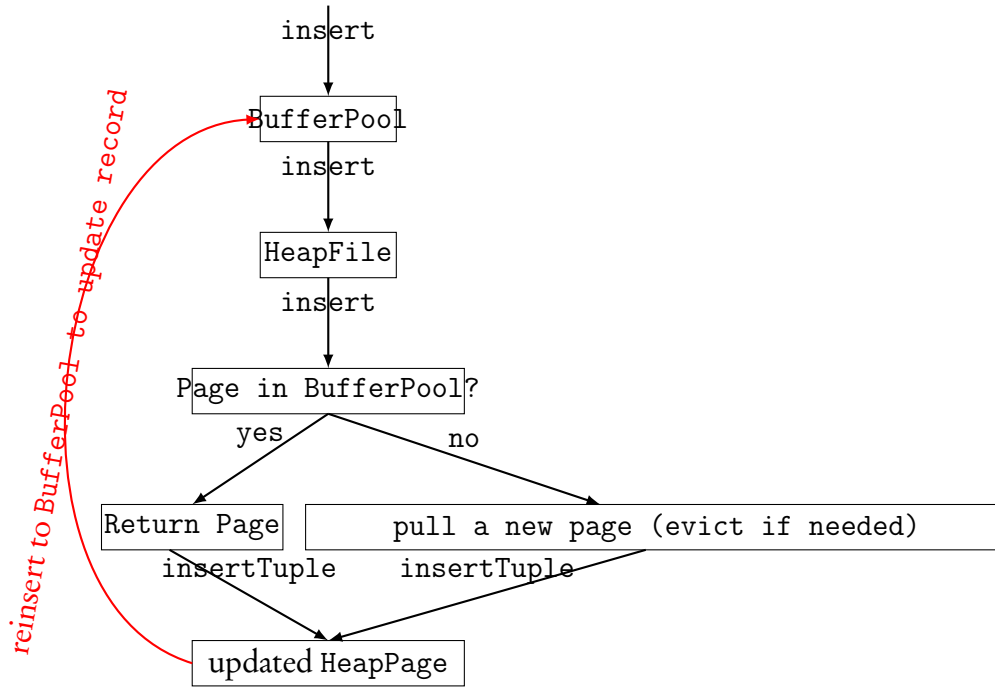reinsert to BufferPool to update record

Figure 5: `SimpleDB insertTuple`

pages in BufferPool by calling the method from `HeapFile` to update the page, then `BufferPool` updates
the records by re-inserting the pages into the `BufferPool`. When the `BufferPool` is full and a new page
need adding, `writePage` from `HeapFile` will be called to write the dirty page to disk and add the new
page into the `BufferPool`. `BufferPool` is in charge of updating the pages because `Insert/Delete`
operators directly call `insert/delete` of `BufferPool`. Besides, we also implemented `Filter`, `Join`
operators and the aggregates. As an example, Figure 4 shows the workflow of using `Join` opeator and
Figure 5 shows the workflow of inserting tuples.

2 Parallel Data Processing

3 Discussion