SimpleDB WriteUp - Group 20

ZAPATA Juan David, MSADEK Mohamed Ali, RAHMANTO Krisostomus

December 10, 2024

Introduction

This report presents the most important design decisions made during the implementation of a basic Database Management System (DBMS) named SimpleDB. We first started by constructing the Tuple class which is essential at a low level for an appropriate data ordering. Then, we implemented two major components of any DBMS such as the Catalog and the Buffer Pool to next build the methods needed to access data using HeapFiles. Finally, we implemented the SeqScan operator which sequentially reads data from tables.

Implemented Classes

Tuple

The most time-consuming part was ensuring that the string representation of the tuple met the specified format. No changes were made to the API.

TupleDesc

This class uses a `List<TDItem>` to store tuple schema information, leveraging its dynamic sizing and easy iteration. A nested static class `TDItem` encapsulates each field's type and name, enhancing modularity and clarity. For the API, added a constructor to initialize `fieldNames` with default values for flexibility. Implemented `equals` to compare `TupleDesc` objects by their `tdItems` list and overridden `hashCode` to ensure consistency with `equals`, crucial for hash-based collections.

Catalog

The implementation of this class is based on using two dictionaries (HashMaps in Java) in order to connect ids to tables and names of these tables to ids. To accomplish this, a `Table.Java` class was created for the project, as it allowed us to organize better data around catalog tables (name, pKeyField, file). No changes were made to the API.

BufferPool

In `getPage`, the system first checks the cache for the page using its ID. If found, it is returned; otherwise, it is retrieved from the disk and added to the cache. If the cache is full, `evictPage` removes an older page to make space. For the API, added a constructor

to specify the buffer pool's capacity (`numPages`). Introduced `getPageSize` for accessing page size, and `setPageSize`/`resetPageSize` for modifying or resetting it during testing.

HeapPageId

The `HeapPageId` class uniquely identifies a page in a heap file by combining a `tableId`, which represents the file the page belongs to, and a `pgNo`, which specifies the page number within that file. It provides methods to retrieve these identifiers (`getTableId` and `getPageNumber`) and implements `equals` and `hashCode` for logical equality and compatibility with hash-based collections. No changes were made to the API.

RecordId

The `RecordId` class serves as a unique identifier for tuples in the database by linking a specific page (`HeapPageId`) with a tuple's slot number (`tupleNo`) within that page. It provides methods like `getPageId` and `getTupleNumber` to access these components. To ensure compatibility with hash-based collections and logical comparisons, the class implements `equals` and `hashCode`. No changes were made to the API.

HeapPage

The `HeapPage` class manages tuples and their metadata in a heap file. It tracks slot usage with methods like `getNumEmptySlots` and `isSlotUsed` and supports serialization via `getPageData`. An `iterator()` was implemented to traverse valid tuples, skipping empty slots. These methods align with API requirements and ensure efficient slot management.

HeapFile

The `HeapFile` class builds up the connection between the software layer and the physical layer by providing access to the files on disk. Using both a page number and the page size defined by the BufferPool, you can use this class to compute an offset which is then used to read the data corresponding to this page. No changes were made to the API.

SeqScan

Through a sequential scan, this class reads each tuple of a specified table. It uses the `DBFileIterator` to which it is connected for most of its methods. Additionally, it concatenates the table alias to the original `TupleDesc` to create the prefixed ones. No changes were made to the API.

Time Spent on the Lab

The time spent on the lab highly varies depending on the Java background of each of the group members, ranging from 8 for the most experienced ones up to 30 hours for the one with very little experience with this programming language. The most challenging implementation was the `readPage()` method for the `HeapFile` class as it involved the use of advanced Java features.