50.043 Database Systems and Big Data

Lab 1 - Spring 2024

Group Members

Name	Student ID
Atul Parida	1006184
Chandrasekar Akash	1006228
Jignesh Motwani	1006178

Summary of Code & Design Decisions

Exercise 1: Tuple & TupleDesc

The *Tuple* class manages tuple data within a specified schema defined by a *TupleDesc* object, utilising a *Field* array for data storage. It includes methods for setting and retrieving field values, handling schema changes, and managing the *RecordId* for tuple location. *TupleDesc* encapsulates schema information through *TDItem* instances with methods for creating descriptors, retrieving field details, calculating tuple sizes, merging descriptors, and comparing for equality. These design decisions and non-trivial implementations simplify tuple organisation and schema descriptions within a DBMS.

Exercise 2: Catalog

The *Catalog* class database tables by storing their schemas and associated files. It uses a *ConcurrentHashMap* object to maintain a mapping of table IDs to *TableSchema* objects. The class includes methods for adding tables with specified file contents and primary keys, retrieving table information, and clearing the catalog. Additionally, it supports loading table schemas from a file. Design decisions involve using a concurrent map for thread safety, and the *TableSchema* class encapsulates details about a table's file, name, and primary key.

Exercise 3: BufferPool

The *BufferPool* class is responsible for managing the reading and writing of pages into memory from disk. The constructor for this class includes a variable to store the number of pages and a *ConcurrentHashMap* to store the pages itself. These pages are accessed through the *getPage()* function which returns the page from the *ConcurrentHashMap*. A ConcurrentHashMap is used since multiple threads might be accessing the database at the same time.

Exercise 4: HeapPageId, RecordId, HeapPage

The *HeapPageId* class is a unique identifier for *HeapPage* objects. The *RecordId* class is a reference to a specific tuple on a specific page of a specific table. The *HeapPage* class stores data for the HeapFiles and implements the Page interface that is used by *BufferPool*. One of the design decisions was to use the *Objects.hash()* function from *java.util.Objects* in *HeapPageId* and *RecordId* for implementing *hashcode()*.

Exercise 5: HeapFile

The *HeapFile* class in SimpleDB adeptly manages tuples stored in a heap file format, distinguishing itself by efficiently linking physical file storage with the logical schema through its constructor that pairs a file with a *TupleDesc*. It ensures unique file identification via a hashing mechanism of the file's absolute path, facilitating effective database management. Notably, its *readPage(PageId pid)* method exemplifies efficient disk access by calculating precise file offsets for page retrieval, thereby preventing memory overload, while incorporating robust error handling to maintain reliability. The class further streamlines data transversal with an iterator conforming to the *DbFileIterator* interface, enhancing data access and encapsulation.

Exercise 6: SeqScan

The *SeqScan* class effectively implements a sequential scan over a table, providing essential functionalities for scanning operations within a database system. The design focuses on flexibility, error handling, and adherence to database system principles, such as transactional integrity and namespace management. Through its implementation, *SeqScan* facilitates efficient and flexible table scanning, serving as a foundational component in the execution of query plans in SimpleDB.

Changes Made to API

Exercise 2: Addition of TableSchema Class

Adding the *TableSchema* class consolidates table-related details, such as the associated file, name, and primary key, into a single encapsulated object. This design improves code organization, readability, and maintainability, reducing the need for multiple data structures to manage different aspects of a table. Previously, we had used multiple hashmaps to implement this.

Exercise 5: Addition of HeapFileIterator Class

The addition of the *HeapFileIterator* class to the SimpleDB API significantly refines data traversal within heap files, providing a specialized mechanism for iterating over tuples. This enhancement decouples file management from tuple iteration, streamlining the architecture by assigning iteration logic to *HeapFileIterator*. This not only simplifies the codebase but also improves performance by efficiently handling large datasets. The *HeapFileIterator* represents a strategic API extension, improving maintainability and optimizing data access in SimpleDB.

Extra: Modification of QueryPlanVisualizer Class

We changed the symbols in optimizer/QueryPlanVisualizer.java for JOIN, HASH_JOIN and RENAME to their Unicode equivalents. This is due to compiler issues that we encountered, initially preventing the project from compiling on Windows due to the windows-1252 encoding. This allows for better cross-platform capabilities.

Incomplete Code Elements

- Synchronised Locking
- Validation of Dirty/Clean Pages

These implementations will be done in the coming lab submissions.