Rapport simple-db-lab1

Elliot BOUCHY- Victor MAYAUD

December 2023

Contents

1 Introduction

The lab serves as an introductory platform to understand the workings of a basic yet functional database management system, modeled on the structure and principles of larger, more complex systems. We delve into the creation and manipulation of tables, management of tuples, and basic querying processes, all within a Java-based framework. The report chronicles our journey through setting up the environment, understanding the provided codebase, and extending its functionalities.

2 Our different implementations

2.1 TupleDesc

The TupleDesc class provides methods to manipulate and access this information, such as adding fields, retrieving the number of fields, getting field names and types by index, and merging two TupleDesc objects. This implementation allows for the creation and management of tuple schemas within the SimpleDB database, providing a foundation for the underlying access methods like heap files to generate and handle tuples effectively.

2.2 Tuple

The Tuple class includes methods to create a new tuple with a specified schema, retrieve the tuple's schema and record ID, set and get field values, and present the tuple's contents as a string. The method setField allows for changing the value of a specific field, and the toString method ensures that the tuple's contents are formatted correctly. The constructor checks whether the provided TupleDesc has at least one field, throwing an exception if not. Similarly, in the getField method, it verifies that the provided index is valid before returning the corresponding field. The fields method returns an iterator over all the fields of the tuple, and the resetTupleDesc method allows for updating the tuple's schema.

The Tuple class prioritizes data integrity by enforcing schema validation, index verification, and controlled modification, ensuring robust error handling and reliable tuple operations in SimpleDB.

2.3 Catalog

The Catalog class manages tables in the SimpleDB database. It uses a map to store tables, where each table has a unique ID. The class provides methods to add tables, retrieve table information (ID, schema, file) and load table schemas from a file. It ensures uniqueness by deleting existing tables with the same name or ID before adding a new one.

When adding a new table with the addTable method, the code first checks whether tables with the same name or ID already exist. If this is the case, existing tables are deleted to avoid conflicts.

The loadSchema method reads table schemas from a file, extracting the necessary information such as table name, field names, types, and primary key. It then creates a TupleDesc object and a corresponding HeapFile file, and adds the table to the catalog.

2.4 BufferPool

The BufferPool class manages a pool of buffers used to store database pages in memory. When creating a BufferPool instance, the user specifies the maximum number of pages (page numbers) that can be stored in the pool. The constructor starts by initializing the variable `m_maxNumPages` with the number value passed as a parameter. Then, it creates a `HashMap` called `m_cache` which will be used to store pages based on their identifier (`PageId`). This `HashMap` represents the cache of pages in memory. Thus, the method initializes the buffer pool by setting its maximum capacity and creating a data structure to store the pages in memory. This buffer pool will be used by other components of the database management system to optimize access to the database pages.

2.5 HeapPageId

The HeapPageId class in the simpledb package is designed to uniquely identify pages within a heap file in a database. The constructor of the class contains the table ID and the page number. Each variable contains getter methods. Additionally, the class contains a method called equals, which allows us to check if two entities are equal, and we have the function called serialize, which allows us to represent this object as an array of integers.The method

hashCode combines the table number, page number, and a prime number (we picked 31) in order to hash the data correctly.

2.6 RecordID

A RecordId in a database is a unique identifier for a specific record (or row) within a database table. It typically consists of two components: a PageId, which identifies the specific page within the database where the record is stored, and a slot number, which pinpoints the record's exact position on that page. The constructor of the RecordId class contains the page ID and the tuple number. We have implemented the getter methods and also the equals method, which allows us to detect if it's the same RecordId. Finally, we added the hashCode method, which is quite similar to the hashCode method from the HeapPageId, except we combine the page ID and the tuple number. Here is the code for the RecordId class in the simpledb

2.7 HeapPage

This class manages data files. The constructor initializes the heap page ID, retrieves the tuple descriptor (TupleDesc), and calculates the number of slots. The getNumTuples method returns the number of tuples on the page using a given formula. The getHeaderSize method computes the header's size in bytes. A getter method for the heap page ID is also provided. The getNumEmptySlots method counts the empty slots in the page. The isSlotUsed method checks whether a specific slot in the page is in use. It achieves this by identifying the byte and bit position in the page's header and applying a bitmask to check if the bit is set. For iterating over tuples in the page, the HeapPageIterator class is used. It has two primary attributes: currentIdx for the current position in the page and heapPage for the page reference. The iterator starts with currentIdx set to 0. The hasNext method confirms the presence of additional tuples by incrementing currentIdx until an occupied slot is found or the end of the page is reached. The next method retrieves the next tuple, throwing a NoSuchElementException if no more tuples are available or if it reaches the end of the page without finding a valid tuple. The iterator method in HeapPage class provides an iterator instance for sequential access to tuples. This efficient design is crucial for managing tuples in a heap file within a database.

2.8 HeapFile

The HeapFile class, constructed with a TupleDesc and a file, provides methods to access and manipulate heap files in a database. Its getId method uniquely identifies each HeapFile by generating a hash code from the file's absolute path. The readPage method reads a specific page from the disk file based on its PageId. It calculates the page's offset in the file, and if valid, reads the page's data into a byte array, returning a new HeapPage. This method ensures proper reading within the file's bounds, avoiding reading beyond the file's end. The numPages method calculates the total number of pages by dividing the file's length by the BufferPool's page size and rounding up to account for any partially filled final page. The iterator method returns a HeapFileIterator, facilitating sequential tuple access within the heap file. This iterator, initialized with the open method, progresses through the file page by page, using the hasNext method to check for remaining tuples. The next method retrieves the next tuple, and the rewind method allows iteration to restart. The iterator's close method resets its state for potential reuse. The inner workings involve the getTupleIteratorForPage method, which interacts with the BufferPool to fetch page-specific tuple iterators. The HeapFile class and its HeapFileIterator play a crucial role in organizing and navigating the data stored within a database's heap files.

2.9 SecScan

SeqScan class function acces tuples in a table. It uses features such as retrieving the actual table name, handling the alias, resetting the table ID, opening, retrieving the tuple description with alias prefixes, checking the presence of a next tuple, retrieving the next tuple, closing the scan and resetting the scan. These features make it easy to navigate and manipulate data in a SimpleDB relational database.

3 Difficulties

We first spent some time about 15 hours to learn the Java language and then understand the structure of the different classes and to finish the lab . We also had difficulties to understand how iterators work and thus to

implement them. Finally, we have some problems to write the ReadPage() function of the HeapFile class and then test our different programs using Ant.

4 Conclusion

Through this lab we have better understood how a database management system works with these different functions, and thus learned how to implement one. We also discovered Java language as well as new software such as IntelliiJ as a development environment and Ant for compilation.