Database Systems- Project

Lab 1: SimpleDB Storage

Chirag Shivakumar 1004996, Devanshi Joshi 1005162, Suthar Prachi Jayesh 1005421

# Design Decisions made during the Project

## Exercise 1- Database Classes, Field Tuples

### Tuple.java

Identifying Requirements- This includes storing the value of each field in the tuple

The Tuple class includes methods for creating a new tuple with a specified schema, retrieving the TupleDesc representing the schema of this tuple, retrieving the RecordId representing the location of this tuple on disk, setting the RecordId information for this tuple, changing the value of the i-th field, retrieving the value of the i-th field, and resetting the TupleDesc of this tuple. Lastly, the string method is to return all the tuple outputs as a string format.

### TupleDesc.java (Used to describe the Schema of Tuple)

The TupleDesc class has two constructors: one that accepts an array of Type objects as well as an array of field names, and one that accepts only Type objects. These constructors create a new TupleDesc object with a given number of fields, each with a given type and, optionally, a given field name. If no field name is specified, the field is treated as anonymous.

The class provides several methods for accessing tuple schema information. The number of fields in the tuple is returned by numFields(), the name of the field at index I is returned by getFieldName(int I the type of the field at index I is returned by getFieldType(int I and fieldNameToIndex(String name) returns the index of the first field with the specified name is returned by fieldNameToIndex(String name).

## Exercise 2- Catalog

### Catalog.java

This class keeps track of all the tables in the database schema.

The class includes methods for adding a new table to the catalog, getting a table's id, getting a table's tuple descriptor (schema), getting a table's database file, getting a table's primary key, iterating over all table ids, getting a table's name, and deleting all tables from the catalog. There is also a stub method that reads the schema from a file and creates the necessary tables in the database. The catalog is thread-safe.

## Exercise 3- BufferPool

### BufferPool.java

The SimpleDB BufferPool class is in charge of caching recently read from disk pages in memory. It stores up to numPages pages and has a fixed number of pages defined by the numPages parameter. The SeqScan operator's constructor and BufferPool.getPage() method must be implemented. An exception is thrown if more than numPages requests are made for different pages.

## Exercise 4 and 5- Heap File access method

SimpleDB's heap file access method divides data into pages, each with a fixed number of bytes for storing tuples with a bitmap header. The HeapFile class handles disk page reading and writing, and each table in the database is represented by a single HeapFile object. The page size and tuple size are used to calculate the number of tuples that can fit on a page, and the header size is determined by the number of tuples per page. The HeapPageId and RecordId classes represent a page's and record's unique identifiers, respectively, while the HeapPage class implements the Page interface to represent a single page in a HeapFile.

### HeapPageId.java

HeapPageId.java includes a class that uniquely identifies a page in a HeapFile, as well as a method for serializing and deserializing the page id. It includes instance variables for table ID and page number, as well as methods for accessing them, and it supports equality comparison, hashing, and serialization.

### RecordId.java

RecordId.java includes a class that identifies a specific tuple on a page and methods for serializing and deserializing the record id.

### HeapPage.java

HeapPage.java classes represent a single page in a HeapFile and implements the Page interface used by the BufferPool. It includes methods for retrieving and setting tuples on the page, as well as retrieving the page header and determining whether a tuple slot is empty.

The HeapPage class contains two fields, m\_isDirty and m\_dirtyTid, which are used to track whether the page has been modified and by which transaction. The Constructor reads bytes from the disk to initialize the header and tuples.

There are several methods in the HeapPage class, including getNumTuples(), readNextTuple(), setBeforeImage(), getBeforeImage(), getId(), and getPageData(). The getNumTuples() method determines how many tuples are on the page. Based on the slot id, the readNextTuple() method reads a tuple from the file. The old data byte array is set by the setBeforeImage() method, and the getBeforeImage() method returns a new HeapPage object from the old data byte array. Finally, the Page interface requires the getId() and getPageData() methods, which return the ID and byte array of the page contents, respectively.

### HeapFile.java

A Heap File's purpose is to retrieve tuples from a file on disk and implement methods for calculating the number of pages in a file, reading a page from the file, and iterating through the tuples of each page.

The code creates a HeapFile, which is a collection of tuples stored on fixed-size pages. The HeapFile collaborates closely with the HeapPage, and the HeapPage constructor describes the format of HeapPages. The code contains methods for reading and writing pages, calculating the number of pages, and inserting tuples. Some of the important methods implemented are;

1. HeapFile(File f, TupleDesc td): A constructor that takes two parameters, a File object and a TupleDesc object, and sets the file, id, and td instance variables. When a new HeapFile object is created, this constructor is called.
2. getFile(): Returns the File object that represents the HeapFile object's on-disk backing store.
3. getId(): Returns a unique integer ID for this HeapFile object. The hashCode() method is used to hash the absolute file name of the file underlying the HeapFile to generate this ID.
4. getTupleDesc(): Returns a TupleDesc object describing the schema of the tuples in this HeapFile.

## Exercise 6- Operators

The purpose of SeqScan.java is to provide an implementation of a sequential scan access method that reads each tuple of a table in no particular order. It oversees performing a sequential scan over the specified table as part of the specified transaction, resetting the table and alias of the operator, returning the table name and alias, and retrieving the tuple description with field names from the underlying HeapFile. It also opens, closes, rewinds, and looks for the next Tuple in the database file.

# Non-Trivial parts of code

## Exercise 4 HeapPage.java last part

### Public Iterator <Tuple> iterator

This Iterator interface is implemented by this class, which is used to iterate through the tuples of a heap page in the database.

Understanding the SimpleDB architecture and the Iterator interface is required for the implementation of the HeapPageIterator class. If there are more tuples in the heap page, the hasNext method returns true; otherwise, it returns false. The next method advances the cursor and returns the next tuple in the heap page. The remove method is incompatible and throws an exception.

The HeapPageIterator class was designed using the iterator pattern, which allows you to access the elements of a collection sequentially without exposing the underlying implementation. The class encapsulates the heap page and provides a method for iterating over its tuples.

## Exercise 5 HeapFile.java

### Function DbFileIterator

This code implements the ImplementedIterator iterator (heapfile iterator), which returns tuples from a HeapFile.

HeapFile's iterator(TransactionId tid) method returns a new ImplementedIterator instance containing the specified transaction ID.

There are several fields in the ImplementedIteratorr class, including an iterator for tuples on a page, the current page number, the transaction ID, and the underlying HeapFile.

The open() method sets the current page number to 0 and starts the iterator on the first page with tuples. The readNext() method returns the next tuple in the iterator, or null if no more tuples are available. If there are more tuples to read, the hasNext() method returns true, and the next() method returns the next tuple. The rewind() method closes and then reopens the current iterator.

# Justify any changes made to the API- None

# Any missing or incomplete elements in your code- None