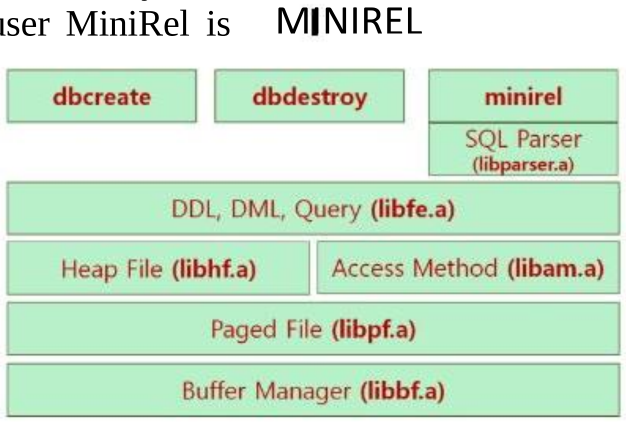
ADB_MINIREL

Brief review about Part I&II

Date: 11.04.2017
Team members:
Simon Kindstroem
Gadimli Nushaba
Taylor Johnathon Soulis
Anastassia Gubska

INTRODUCTION

This project is about to build a simplified relational database system called MiniRel. The implementation of a single-user MiniRel is composed of five layers.



Part I (the BF layer) and Part II (the PF layer) involve the implementation of code to manage a buffer pool and to manipulate files that contain a sequence of pages, respectively.

Part III (the HF layer) involves the implementation of heap file management to provide the abstraction of random and sequential access to fixed-length records stored in files.

Part IV (the AM layer) involves implementing procedures for B^+ tree index structures to speed up associative access to records.
For Part V (the FE layer), this layer implements catalog management, data definition language (DDL), and data manipulation language (DML).

PART I – THE BUFFER POOL

The buffer pool layer handles the memory between the application and the disk. It ensures that only a limited amount of data is in main memory at any given time and will reuse memory when possible. By using a buffer pool, we can ensure that we do not harm the system operability by using excessive resources and performance increase is gained by reusing memory. Before a deeper description of the implementation can be given a short introduction to the used data structures have to be given.

The most important data structure is the ***BFpage***. It has all the information regarding a page: memory buffer, PF file descriptor, UNIX file descriptor, page number, dirty flag and pin count. In addition, it also has pointers to other *BFpages* allowing it to be used as nodes in a doubly linked list, which is it in the *LRU-list* and also as nodes in the singly linked *Free-list*.

Free-list contains any pages that are not in use and actually works more like a stack, pushing and popping the head. *LRU-list* (Least Recently Used) contains the most recently used pages and is a list that can be compared to a FIFO queue, in the simplest cases, where the elements are removed as the queue gets full. However, there are some edge cases which have to be taken into consideration. First of all, a page cannot be removed from the *LRU-list* if it's pinned (in use). That means that sometimes the element to get removed is not the last one. A request to remove the pages of a file can also be issued and would result in non-last elements to be removed. If a page is touched (used) it will be moved to the front of the list because now it's the most recently used page.

To enable quick lookups *a hash table* is also implemented. It maps a file descriptor and page number to a specific page. The *hash table* is implemented in a chaining fashion to allow easier lookups. The hashing function is a simple implementation, but guarantees that consecutive pages from a single file to have unique hashes while the number of pages is smaller than the size of the table. Because data access often occurs in this pattern it was deemed a reasonable algorithm.

With all of these parts working together it's possible to have efficient reads and it also enables lazy writes in a write-back fashion.

PART II – THE PAGED FILE

The Paged file (PF) is the second layer comes after Buffer Manager, and its provide resources to allow client to do I/O in terms of pages. *Interface* of PF layer function provides to create, open and close files, to scan through a given file; to read a specific page of a given file; and to add new pages to a given file.

The important data structure of PF layer is the ***PF file table*** ensures that as files are open they will keep under track. It has an information about maximum file entries, PF file descriptor (different than UNIX file descriptor), file header, file names, also number of pages in the file.

The Paged file interface connected with buffer manager as a buffer initialization is a first job that it has to do, which also supports PF layer into to keep track of the opened file table, communicates with buffer manager to read/write pages of the file, in case of if page gets any changes before closing, it first flushes the buffer and then updates the file header and then it can be close.

You may face to some *PF_PrintError* function error when you try to run *pftest* file, as it was not defined or mentioned anywhere in the documentation, hereby we've emailed and asked about it from professor. Relying on an answer from professor, we didn't implement *PF_PrintError* function. Please be sure, to include this function to **pf.c** file before to run *pftest*.

END NOTE:

From now our team will work on next remaining layers to complete the MiniRel database.

When all the five parts of the MiniRel project are completed, the final deliverables will produce the followings:

- Three executables: dbcreate, dbdestroy and minirel,
- Six function libraries: libbf.a, libpf.a, libhf.a, libam.a, libfe.a and libparser.a.