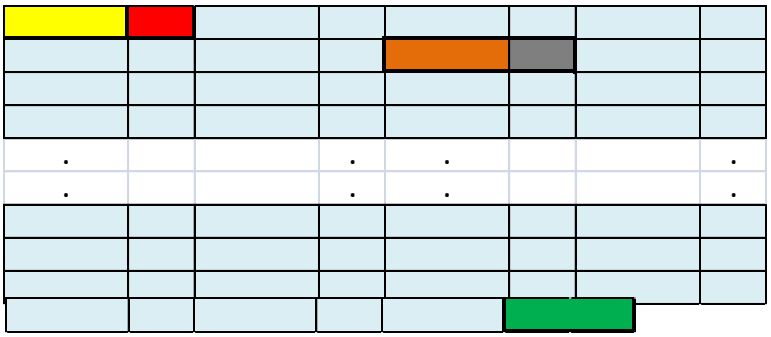
CS222: Project 1Report

Kewal Gandhi: 87887035 *Group Number: 16*
Sai Sundar Raghavan: 65525790

PAGED FILE MANAGER

The Paged file Manager is used to create a HEAP FILE. The pages file manager is used to manage these files. We use a directory based structure to manage these heap files. The directory structure uses header page to point to data page. The FIRST HEADER PAGE is always located at position 0. The structure of the header page is shown below:



	Total No of Pages in File/ Unused	4 Bytes
	Unused	2 Bytes
	Pointer to Next Header Page	4 Bytes
	Record Entries, Consist of:	6 Bytes
	Actual Page Number	4 Bytes
	Free Space in that Page	2 Bytes

*For the first header page the first four bytes are total no of pages in the file, for the rest they are unused

Where, Actual Page Number: (Byte Number of start of Block)/PAGE_SIZE
Virtual Page Number: Pointer to entry in Header Pages, which contains Actual Page Number

The functions of the page file manager are:

- Create and destroy files

It creates the first header page for a newly created file.

- Associate FileHandle objects

It also associates FileHandle objects to perform read/write functions on the file. It takes care that only one stream associated with the file has write permissions, multiple can have read access. This is done using a <String Filename, int Count> map.

- Read and Write Pages

It also allows reading and writing pages into the file

- Append new pages in file

It appends new pages in the file and makes an entry in the directory page for that file

- Insert new Header Pages

It also inserts new header pages when required and links them

- Utility functions

It also performs some utility functions like getting the no of pages in a file, translating the virtual page number to actual page number, getting header page numbers, etc

The following routines have been implemented (or added newly) in RBFM:

```
RC createFile     (const char *fileName);                          // Create a new file
RC destroyFile    (const char *fileName);                           // Destroy a file
RC openFile       (const char *fileName, FileHandle &fileHandle); // Open a file
RC closeFile      (FileHandle &fileHandle);                         // Close a file
INT32 insertHeader  (FILE* fileStream);                    // Inserts Header Page
    INT32 translatePageNum(INT32 pagenum);
   // Translate the Virtual Page Number to Actual Page Number
    INT32 getNextHeaderPage(INT32 pageNum);
         // Get the Next Header Page given a Header page
    INT32 getHeaderPageNum(INT32 pageNum);
   // Get the Header Page Given The VIrtual Page Number of a data page
    RC readPage(PageNum pageNum, void *data);            // Get a specific page
    RC writePage(PageNum pageNum, const void *data);      // Write a specific page
    RC appendPage(const void *data);                      // Append a specific page
    unsigned getNumberOfPages();               // Get the number of pages in the file
```

The following variables have been used in the PFM Layer

```
FILE* stream;       // Stream of the file associated to this Handle
string fileName;    // Name of the file associated to this Handle
bool mode;          // 0 for read, 1 for write
```

RECORD BASED FILE MANAGER

The record based file Manager uses the functions implemented in the page file manager predominantly. The RBFM is responsible for the following

- Updating the free space associated with the data pages, in the directory.

The RBFM updates the free space in each data page, after every insert and delete in the associated data page.

- Converting records from application format to disk format.

If the application record format is

Integer	Stringlength	String	float

The format of the disk record will be

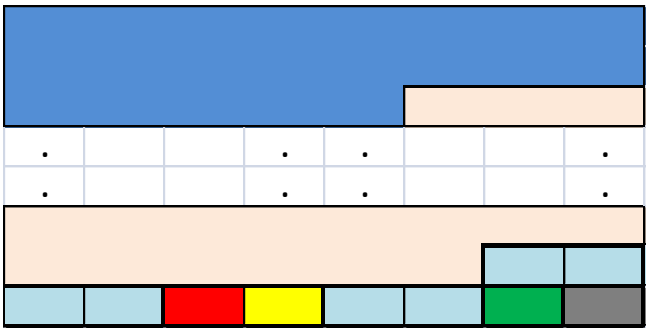
3	Offset1	Offset2	Offset3	Integer	String	Float

This also ensures that the record attributes have O(1) access.

- All record based operations such as read, insert print etc.

- Maintaining Bookkeeping information for every record

The RBFM maintains a slot directory in the data pages, in which the length and offset (within a page) of the records are updated. It is the combination of the virtual page number and the slot that gives a record its unique position within the data base. The Structure of a data Page is shown below:



	Pointer to Free Space Block	2 Bytes
	Number of Slots	2 Bytes
	Slot Entries, Consist of:	4 Bytes
	Offset Pointer from start of Block	2 Bytes
	Length of Record in that Slot	2 Bytes
	Data with or without Holes	
	Free Space Block	

- Also abstracts the PFM layer from the application

The application layer does not know about the PFM layers presence. The RBFM layer intercepts all record based queries.

The following routines have been implemented (or added newly) in RBFM:

- `RC createFile(const string &fileName);`
- `RC destroyFile(const string &fileName);`
- `RC openFile(const string &fileName, FileHandle &fileHandle);`
- `RC closeFile(FileHandle &fileHandle);`
- `INT32 findFirstFreePage(FileHandle &fileHandle, INT16 requiredSpace, INT32 &headerPageNumber);`
- `void* modifyRecordForInsert(const vector<Attribute> &recordDescriptor,const void *data,INT16 &length);`
- `RC modifyRecordForRead(const vector<Attribute> &recordDescriptor,const void *data,const void *modRecord);`
- `RC insertRecord(FileHandle &fileHandle, const vector<Attribute> &recordDescriptor, const void *data, RID &rid);`
- `RC readRecord(FileHandle &fileHandle, const vector<Attribute> &recordDescriptor, const RID &rid, void *data);`