Abstract

MiniBase is a database management system in java. This assignment deals with the understanding of the same. It is a relational database which was designed for the purpose of giving hands on experience for the students in which they can further enhance the same. The MiniBase that has been given is made of several parts, they are a buffer manager, disk manager, heap and index storages, b+ tree structure and Iterator for data scan, Query Processor, Optimizer and parser, Execution planner. All these components paves way for the management of the data and for processing queries.

OS: Ubuntu 14.04

Introduction:

This test report deals with five major components of the MiniBase, which are buffer manager, disk manager, heap files, B+ Tree structure and index files. All these tests helped me in understanding the working behind them, and the extent to which they can be stretched, helped me in understanding their limits, vulnerabilities and boundaries.

Tests Description:

Buffer Manager:

The buffer manager component consists of buffer pool and hash table. Buffer pool is made up of series of frames which can contain data from disk pages.

- Hash table is used for containing the mapping between the page number and the frame number.
- The buffer pool structure contains page number and Pin count.

Each frame as a status to mark if it is available or pinned and page replacement algorithm is cock.

Test1: **Normal Buffer Manager Operations:**

- The total number of pages is set as 50, this test tries to allocate 50 pages and verifies if allocation is successful. When a set of pages are memory allocated, first page is automatically pinned and returned back. Then first pinned one is unpinned and verified. There is no change in the data so while unpinning the dirty flag is set as false. Data is written on all pages, the page is first pinned before writing any data and then finally the pages are unpinned after written. The data on page is a page number added to 99999. While pinning the empty flag is set as true and while unpinning the dirty flag is set, because data is changed.

- The written pages are then read to be verified if the data is correctly written. The pages are again pinned, read the data, verified for their correctness and then unpinned. The data is verified by comparing each page data with corresponding pagenumber+99999. While pinning the empty flag is set as false, as there is already data present in the pages read. Finally all the allocated pages (50) are made free from memory.

Test2: **Illegal buffer manager operations**

- In this the test tries to pin more number of pages than the number of frames.
- Pins the new pages that are allocated and makes sure that there are no more frames to accommodate new pin requests.
- Tries to pin a new page and checks if it fails as expected. It fails as there are no frames to pin.
- Pins a page that is already pinned, now the pin count becomes 2 for this frames. Now Tries to deallocate/free the doubly pinned page and makes sure it fails as expected.
- The test tries to unpin a page that is not already in the buffer pool. The last page, that is page 51, is not in the buffer. So the test unpins page 51. And it fails as expected.
- Finally it deallocates all the pages that were allocated at the start of the test i.e. they are freed from the memory.

Test3: **Internals of the buffer manager**

- In this test we pin some pages and return back them but while doing so we leave out some random page pinned and without unpinning them.
- Now we try to pin all the pages that were pinned in the before step and we unpin them, while doing so we come across those pinned pages that were left out in the previous step and they will also be unpinned successfully. Thus in this case the test succeeds.

Disk Management Tests

Disk Manager:

Takes care of allocation and deallocation of pages in database.

Test1: **Creating a new database and testing normal operations**

Allocates a new set of files(6) on disk. Allocate pages (30) write data into some of the allocated pages (20) and we deallocate (10) the rest of the pages. All these operations are successfully completed.

Test2: **Deleting and Getting the File entries**

- Delete 3 file entries that were created, for remaining pages select the next three pages.
- We wrote data in 20 page sin the previous step and left them without deallocating them. Now we read the data and verify the data that is written onto these pages.
- Then finally we deallocate all these pages.

Test3: **Verify Conditions**

- In this test it looks up for the deleted file entries and verifies the same if it fails. Since the files are already deleted this run will for sure fail.
- Then it tries to delete an already deleted file entry, since the files have all been deleted it should fail to delete an already deleted file which is expected output result.
- In this step it tries to delete a file entry which was never ever created and expected it also fails.
- Then we try to look for a file which was never created which also fails.
- Then we try add a file entry which is already present which also fails as expected.
- Then we try to add a file entry name which is huge and we expect a failure result.
- Then it tries to add a set of pages which are too long, which are negative and deallocates the negative page in which of all cases we expected a failing result. Which is obtained.

Test4: **Test Boundary conditions**

- We compare the number of unpinned pages with total number of buffer frames and make sure that there are no pinned pages. Then we allocate all pages accounting for a DB overhead. Then we try and allocate one more page which would fail s expected.

- Then we try to deallocate a set of 7 pages from 3 to 9 and deallocate 8 page set from 33 to 40 and we allocate memory for the same 33 to 40 and we see that it gets succeeded.

- Then we try to deallocate two sets of 8 page set and then in a one go we allocate 16 pages which will also succeed.

- Then we add file entries so that the directory surpasses a single page. To verify the same, we allocate all remaining pages. The test would fail because the directory would take 2 pages to store the file entries.

Heap File Tests:

Test1: **Insert and scan records**

A heap file is created and 100 records are added into the file.

And then each record is got from the scan and verified for the record length and the correctness of record data. Finally it is made sure that the scan didn't leave back any of the page pinned.

Test2: **Delete fixed-size records**

- In this test we deleted a fixed amount records and verify for the correctness of the same.
- We choose the records with pid as odd number and delete them all. And verify the same successfully.

Test3: **Update fixed-size records**

- In this test, the records values are updated and then verified if they are correctly updated.
- The test opens a scan, updates each of the value by multiplying it with 7. And then closes the scan. And then at last again a scan is used for verifying the correctness of the record

Test4: **Error conditions**

In this test we try to update the size of the record with a smaller and larger value than it is set in which both the cases we see the test fails.

B+ Tree Tests

Test1: **Inserting a record**

In this test we enter a record into the b+ tree. Option 5 is used here.

Test2: **Deleting a record**

In this we try to delete the record from the b+ tree. So we delete two records and verify that the records are deleted by printing them out in the output and checking with them.

Test3: **Initialize a scan and access records**

- In this test we start a scan by giving it the lower and higher integer value, next we access the records by using the option 13.
- We delete the value also by using the scan and option 14.

Test4: **Print by page**

In this test we try to print out all the leaf nodes of the tree that is available in the page. Each page as a number and we give this number in order to access the page using other options. Option 4 is used here.

Test5: **Delete a record on the end of the scan**

We try to delete a record at the scan which fails for sure because there won't be any record to be deleted at the end of scan.

Test6: **Insert n records in order**

- In this test we insert n records in a order if we say specify 100 it will create 100 records from 1 to 100.
- We can print the tree and verify whether they have been implemented. Option 7 is used here.

Test7: **Insert n records in reverse order and random order**

In this similar to the previous test we try to insert some number of n records in reverse and random order by choosing the various options available and then try to check them by printing the tree and by using the scan. Option 8 and 9 are used here.

Test8: **Insert n records and delete m records**

This test involves two things together in this test we insert n number of records at the same time we also delete m number of records from the inserted number of records. Option 10 is used here.

Test9: **Delete some records**

By using this test we can delete a record in the b+ tree option 11 is used for the same.

Test10: **File operations:**

In this we use the options 16, 17 and 18 to access the files. 16 is used for closing a file and 17 is used for opening an existing file. Each file as a number and we access the file using that number. Option 18 will destroy the file given a number all these cases the file must be present else it will throw an error.

Test11: **Naive delete and Full delete**

This test is used for creation of a new file in the database.

Test12: Option 19 to exit.

Index Tests

Test1: **BTree Index creation**

The test verifies the creation and validation of BTree. The test creates a heap file and inserts records into the table from an array data1 and we create a BTree index file. It creates a scan for the heap file. And then all the records from the heap file are scanned and then inserted into the index. Then the scan for the index is created and all the records from the BTree index are scanned and verified with the second ordered array data2. data2 is just the sorted version of the array data1. The index is also verified.

Test2: **Identity search and range search**

We open the B tree created before and add condition expression selectively to give identity, then scan records using index and identity slection condition and verify the records as same identity before and after creation and then we add a range condition to index and scan to verify the index returns only records withing range.

Test3: **Range scan on integer column**

In this test we create a heap file and insert 1000 records each of int, string and float. And then we create a Btree index on file with key as integer, scan records from 100 to 900 by range condition and verify the records from 100 to 900 whether they are stored.

Conclusion:

These tests help us understand the complete working of the database management system and also the limits of each and every component in the system. I could learn about the buffer management, disk space management, B+ tree and the indexing. I guess with the help of this assignment I wouldn’t be having any troubles in completing my final project.