Υλοποίηση Συστημάτων Βάσεων Δεδομένων 2019/20

Άσκηση Δύο

Καλλίνικος Γρηγόρης - 1115201500056
Παιδάκης Θεοδόσης - 1115201500118

Instructions for testing the hash_file.c functions through Makefile:

[1] ***make hp***
Compiles the ht_main.c in examples which is given as default.

[2] ***make diff_buckets***
In this test we create and open 5 files, each file has a random number of buckets and 1700 records (which are inserted). Then we close the third file and we open a new one, to test if the array is working.

[3] ***make same_file***
In this test we create one file only and we open it 20 times, we insert the 1700 records, close it in one position of the array and then reopen it.

[4] ***make randRecords_delAll***
In this test we randomize the number of records for each file. For the final file we delete all its records, and check functionality with printAllEntries.

[5] ***make clear***
Deletes all data* files in the directory.

Implementation:

We have a global table array of size MAX_OPEN_FILES.

Every time an index is created, we create a new hashfile, which is marked with a '*' at its starting block. After the '*' we save the number of buckets given. Then we preallocate as many blocks as the buckets given, so that we have them ready for the initial hashing. Any new block will go below the map and will be linked to the last block of the same bucket.

Every block that is created saves the number of records at its start, and right after that an int which shows the next block of the block chain in the bucket.

Every time an index is Opened we open the file with the name given, then we read the number of buckets in the file, which is saved in the starting block, and we save that value to the array. The index descriptor for the file is the same as the file descriptor given by the system, so we don't save that information.

The hashing function we use is a simple modulo operation, with the number of buckets plus 1, so that we immediately find the bucket in which the id is mapped.

When we close the file, we also set the bucketNumber of the corresponding position of the array to 0.

To Insert an Entry we find the hashing value of the record id, get it’s bucket and go down the chain of blocks in the bucket until we find a block that has space for it. We then insert it in that space and update the value 'records' in the block.
If we don't find space for it, we allocate a new block, updating the last one to correctly chain to the new one, and we insert the record to the new block.

To Print all Entries with ID = NULL we run a print for every block in ascending order of block Number (first the 1rst block, second the 2nd block, ...).
To Print all Entries with a given id, we find its hashing value, and we search for it in the chain of blocks in the corresponding bucket.

To Delete an Entry we essentially find its id with the same way as in Print all Entries and then: either that record is the only one, so we just decrease the records number in the block, or there are more records, so we get the last record remove it and place it in the position of the record.

In our implementation, we don't actually memset the space of the record, but just decrease the number 'records' in the block. When a new record comes, it will be placed inserted right on top of the record we 'deleted'. In other words, the record stays in the bucket, but we don't see it and if a new one comes it will cover it.