Collaborators: Alisha Grama and Akshay Desai

We used no code that was not our own.

We implemented a hashtable in order to store the databases. We used a simple hash function of mod 101, which enters the item into an array of length 101. At each index there is a linked list that will store other objects that fall into that bucket. We first read in all of the data from a text file named schemas.txt. We assemble different hashtables based on the different databases read in.

There is one file to compile and run (Proj4C.c). This will write the output to one file (output.txt). We have included a build.sh file to help everything run. Ensure that you have navigated to/are in the file when running.

For our nodes of information, we decided to use a general node that could potentially hold every piece of information because we figured that it we had a different struct for each table that could only hold what is accepted in that table, there would be a lot of code repetition with minor changes for each type of struct. It turned out to work well and we are glad we went that route. Also, we thought about making each table a global variable instead of passing them as arguments but using them as arguments turned out to be a good idea because each method was generic and could work on each table then instead of separate methods for each table.

For part 1, we implemented basic database operations on our generic nodes and only touched what needed to be. In later parts of the project where nodes are changed, we do not cheat and alter the attributes of the nodes, we read each node separately and create new nodes as if we couldn't rewrite and expand what each node from each table can hold.

For part 2, for getting the grade of a certain name in a certain course, we use SNAP -> CSG. For getting the room of a name at a certain time and day, we use SNAP -> CSG -> CDH-> CR. Both parts of part 2 use multiple helper methods.

Getting into part 3, for Projection there are two methods, one to project on an int (student id), and one to project on a String (all the other attributes). In both projection methods the "char column", the char given decides which column to use. For the int projection, it creates and returns an array of ints, for string projection, an array is given to the method and it is changed.

For Selection, the method takes the array to be selected upon, a char to know which attribute to check, and int for if it is a student id , and a char* for the other attributes. It also takes an array for the finished array.

For Join, it takes three arrays. The 1st two arrays are the arrays that are to be joined, the last one that will be the finished array.

When the program is run, it only prints “this” in the console, everything else is printed to the “output.txt”. At the end, when we are printing all of SNAP, each index looks as if they are printed twice, the first of each index is the 1st element in the bucket, the 2nd is the 2nd element in the bucket. If there are 2 nodes in a bucket, it will have 3 of that index printed, 2 with the data, 1 empty. In the Snap example, we deleted element with studentid== 22222 and added two elements.

EXTRA CREDIT: We also implemented Union and Intersection. For these, first we check to see if the two arrays are of the same type. Then we perform the necessary operation. For Union, the first array is returned as the union. For Intersection, there is a third array returned with the intersection of the two. There are no outputs for this, so the output file is kept neater.