CSC 173 Project 4
Maryfrances Umeora and Kharissa King
NETIDs: mumeora & kking33

Program Files Used

main.c
Tuple.c
Tuple.h
HashMap.c
HashMap.h
LinkedList.c (provided by Ferguson with important changes)
LinkedList.h (provided by Ferguson with important changes)

How To Run Our Project

Our project can be run on any IDE or terminal.
Navigate to where the folder containing our files is stored, type “`make`” then “`./database`”

How The Code Runs (PLEASE READ)

First, all of Part I runs at once. Since a repl was not required, we simply tested our function, and if you scroll through the large chunk of text explaining what is happening, or even the code in the main method in main.c, you will see that we test all of the required methods.

To move on to Part II, make sure you only press enter **ONCE**, and that you cursor is after the final . in the sentence. Part II is a repl that prompts you for information. Please follow the directions.

Once again, press enter only once with your mouse at the current position in order to move on. Like Part I, Part III is not a repl. Please scroll through the printed text and/or the code for proof that our methods work.

PART I

1. Implement Relations

Found In: Tuple.c, Hashmap.c

We have two major structs, Tuple and Relation.

- A Tuple has an array of attributes, and a Tuple* is type-defined as TupleInstance.
- A Relation has an array of headers (the schema), an array that holds the index of the keys of the relation, and an array of LinkedList of Tuples. This array is the hashmap. A Relation* is type-defines as Table.

We have five relations to implement: SNAP, CSG, CDH, CP and CR. We have methods that allocate memory for everything a Table needs, then we call populateSNAP(), populateCSG() etc to fill the tables with the required data.

2. Insert, Lookup, Delete

Found In: Hashmap.c

Insert: Of course, to populate the tables, we need to insert. Insert takes in Table and a tuple. After calling the hashing function to find the value of h(k), which we do by adding up the values of the ASCII characters of each of the key attributes, we iterate through the array of LinkedLists associated with the tuple and insert, sieving out duplicates and chaining when necessary.

Look Up: This also only takes in a Table and a Tuple. The tuple may or may not have some attributes as "*". If all the keys are available, we can simply find what h(k) was by running the hashing function and running through the Linked List stored at that index. If not, we iterate through the entire table, looking for the Tuple that matches our patter.

Delete: This does the same things as Look-Up, except we delete instead of adding. In order to do this, we messed with Ferguson's Linked List a little to work well with our Tuple struct.

3. Demonstrating Operations

Found In: main.c

We demonstrate all operations on each and every relation, with informative printed methods along the way.

4. Save and read from external file

Found In: main.c

For this part, we first save the current contents of the five relations to file. CSG is saved to csghash.txt, SNAP is saved to snaphash.txt, CP is saved to cphash.txt, CDH is saved to cdhhash.txt and CR is saved to crhash.txt.
Next, another instance of a table is created for each relation and the data that was read to file is read into these new tables. These tables are then printed to demonstrate that the tuples from the original tables were successfully written to and read from file.

PART II

1. What GRADE did NAME get in COURSE?

Found In: main.c

For this part, we first search the SNAP database for the ID number of the student(s) with the specified name. A lookup is then performed with this ID number and the Course that was entered, and if any tuples are found matching the criteria, the corresponding grade is printed.

2. WHERE is NAME at TIME on DAY?

Found In: main.c

Like with the previous query, we simply looped through all the relations required to get the information, printing informative methods along the way. This is a repl, so you can test this yourself.

PART III

1. Selection

Found In: Hashmap.c

Our Selection, which returns a table by the way, calls another function named SelectionHelper. As Selection is meant to be able to select on multiple attributes, SelectionHelper selects on just one, and Select calls SelectionHelper for each attribute, sieving out the unnecessary. SelectionHelper performs a fancy look up and inserts the matching tuples into the returnTable.

2. Projection

Found In: Hashmap.c

For Projection, we perform a fancy loop to get all the indexes of the headers we are projecting on. Then, looping through the entire table, we create a new tuple that grabs only the projected attributes and adds them to the returnTable.

3. Join

Found In: Hashmap.c

We first create a return table and allocate memory for all of its attributes using the input tables as a guide. It is ensured that the keys and schema of the previous tables are retained when the return table is created. The schema of the first table are added to the return table, then the schema of the second table that are not already contained in the return table are added afterwards. We then find the positions of the attributes that we are joining on. A nested-loop join is performed where we check to see if the join attribute of the first table is the same as the join attribute of the second table for all values of the tables. We then add the necessary tuples from both tables to the return table.

4. All Three

Found In: Hashmap.c

For this part, we first call the join function on the two tables that are being operated on. We then call the selection function on the resulting table, and the projection function is called afterwards to give the final result.

BONUS

The first bonus option was to write “generic” representations of tuples and tables. Our entire code was generic, so we believe this fulfills the bonus.

And that’s it!!