CSC173 Project 4

Database Implementation

Yixin Zhang

Junkang Gu

Ziyi Wang

This project implements a generic relational database that reads in a file, creates tuples and relation, then perform all the required functions and generates an output file.

We mainly have three hierarchical structures that achieve our task —— a *struct Database* that contains an ArrayList of *struct Relation*, a *struct Relation* that basically has an ArrayList of *struct Tuple* as well as binary search trees and hash table. We will elaborate on these structures in the following text.

Struct Tuple is structured the same way as an ArrayList of size 10. It implements functions such as *Tuple_add_el*, *Tuple_remove_el*, and *print_Tuple*, which enable us to modify the tuples generically and add attributes as needed in the relations.

```
struct Tuple {
    char* array[10];
    int num;
};
```

Struct Relation is more complicated. It has, of course, a hash table for the primary key and an array of trees that implements multiple secondary keys for all attributes that are not primary. In addition, it has fields that keep tracks of its name, schema, and the number of tuples and attributes. As we know, when performing functions such as projection and join, looping through every tuple in the relation is required. Therefore, we also designate a simple ArrayList that stores the pointer for each tuple, so that no hashing or tree traversal is needed when looping. During the initialization of a Relation, we have a rather neat function called *Relation_set_KeySchema(int keynum, Tuple schema, Relation this)*, which takes in these parameters and automatically initialize the hash table for the given primary key index in the

```
struct Relation {
    char* name;
    Tuple schema;
    int key;
    ArrayList all_Tuples;
    Tree secTrees[10];
    LinkedList *hashT[20];
    int n_attr;
    int n_el;
};
```

schema, and also generates binary search trees for each non-primary attributes. In this way, we can perform the Lookup function with any parameter regardless of its key type.

For the *struct Database*, it's normally not necessary if we just want to implement the functionality (everything can be done in Relation.c). However, since we are dealing with external files, a overall structure would be needed as an object that executes this task. A *struct Database* contains an ArrayList of Relations. It first implements *readFile(char* filename)* that reads in a specially formatted txt that layout all content of the database (i.e. CSG, SNAP, CDH, CR) and initialize those relations, then, after doing some REPL, it generates an output file that overwrites the original txt —— that is, the resultant relations after implementing all functionality.

```
struct Database{
    ArrayList relationList;
};

typedef struct Database* Database;

Database new_Database(void);

void saveFile(Database database);

Database readFile(char* filename);
```

When running our program, the main function will first call *readFile(char* filename)*, then go through each part of the project in the REPL. It will ask the user to enter parameters and run the functions. As we mentioned above, finally, it calls *saveFile(Database database)* to save the output.

Most features of our project are described above. If you have any question, please email us. (Yixin Zhang: yzh223@u.rochester.edu, Junkang Gu: jgu8@u.rochester.edu , Ziyi Wnag: zwang104@u.rochester.edu)