CSCE 315-505

Jialu Zhao

Jacob Vacek

Srishti Sanghvi

**Team Project 1 - Checkpoint 1**

**API Specification:**

Our project consists of three main classes: Database, Table, and Record. The following description corresponds with how we plan to implement our Database class.

**Class: Database**

|  |  |
| --- | --- |
| Database() | Default Database constructor; takes no arguments |
| void add\_table(Table\* tab, string name) | Add a new Table object to a database; adds table to the end of the database |
| void drop\_table(string name) | Removes a specific table from a database |
| vector<string> list\_Tables() | Return a vector containing the names of tables in a database |
| vector<Table\*> get\_Tables() | Return a vector of all Table pointers in a database |
| Table\* Query(string SELECT, string FROM, string WHERE) | Query member function |

In our Database class implementation, our private data members will be implemented using a map data structure.*.* Maps consist of a key value which are used to sort and identify elements, and a mapped value that store the content associated with the key. In our implementation, our key will be a string and our map will be a Table pointer. In our public implementation, we will have a constructor function for Database that will create an “empty” database for us to begin with. Next, we will have *add\_table()*, that will take in an object called tab of the Table pointer class, and a string called name. This function will be used to add to the end of the map, or in other words, add that table to the database. Furthermore, we will have a *drop\_table()*, which will be used to remove from our map, taking in a string called name. Our next function, *list\_tables(),* will be used to print names and another function, *get\_tables(),* will return pointers to the tables.

*Query()* will be a member function of the class Database. The query command will take in three arguments, a SELECT argument, a FROM argument, and a WHERE argument, and will return a table.

**Class: Table**

|  |  |  |
| --- | --- | --- |
| Table() | Default constructor | |
| Table(vector<string>\* input) | A constructor should also be supported that takes a list of attribute names | |
| void add\_attribute(string atb) | Takes attribute string, add it to the vector, goes through records vector, appending empty str | |
| void remove\_attribute(string atb) | Get atb elements number in attributes and then removes all data about this element from record | |
| void insert\_record(Record\* rec) | Push back record\* on to records | |
| vector<string> get\_attributes() | Returns a list of the attributes for that table , in order | |
| int get\_size() | The number of records in the table | |
| Record\* get\_record(int i) | Returns individual records from the table | |
| void set\_key(string atb) | Allows a single attribute name to be designated as a key for the table | |
| Table\* cross\_join(Table\* a, Table\* b) | Takes two tables as input and produces one table as output using cartesian product | |
| Table\* natural\_join(Table\* a, Table\* b) | A function that allows a single attribute name to be designated as a key for the table | |
| int Count(string atb) | Compute the size of a single attribute | |
| string Min(string atb) | Compute the minimum in a single attribute | |
| string Max(string atb) | Compute the maximum in a single attribute | |

In our Table class implementation, our private data members will consist of the following: a string called *name*, a vector of strings with a pointer to the vector called *attributes*, and a vector of pointers to the Record class, called *records*. Our public implementation will consist of a constructor, *Table()*, that will create an “empty” table, or in other words, a table without any rows and columns. We will then implement another constructor that will take in a vector of strings with pointer to the vector called input.

Next we will have *add\_attribute(),* which will take in a string called atb that will take the string and add it to the vector of Attributes, going through the records vector, adding empty string. Furthermore, we will implement *remove\_attribute(),* a function that will also take in a string. *remove\_attribute()* will get the input string’s element number in the *attributes* vector and remove that element *atb* from the Attributes and Records vector. Our *insert\_record()* function will take in the pointer to the Records vector called rec. This function will be used to push back the record pointer onto the Records vector. Next, we will have *get\_attributes()*, which will be of the type consist of a vector of strings, which will return the deference atb pointer, or in other words, will return a list of the attributes for that table, in order. Furthermore, we will have *get\_size()*, which is of type int that will return the number of records in the table. We will also have *get\_record(),* which will be used to return individual records from the table. Next, we will implement *get\_record(),* that will take in an int i, serving the purpose of returning individual records from the table. Furthermore we will have *set\_key(),* which takes a string. Our cross join command that takes two tables pointers as input and produces one Table as output will be implemented as *Table\* cross\_join(Table\* a, Table\* b)*, which will return a pointer to the Table class. The natural join function, *natural\_join(),* takes in two input tables. If any of the attributes in the first table match the key of the second table, they will be added to a new table, along with any other information from the second table. Additionally, to implement routines that take a single attribute name as input, and compute the count, min, and max, we will have three functions *Count(), Min(), and Max()*. All three functions will take in the a string, with differing return types of int for the count function and strings for Min and Max.

**Class: Record**

|  |  |
| --- | --- |
| Record (int size) | A constructor that allows creation of a record of arbitrary size, and initialization of the entries to a null string |
| int get\_size() | Gets the size of the record and returns how many entries it has |
| int set\_entry (int i, string val) | Set rec[i] = val |
| string get\_entry (int i) | Return rec[i] |

In our Record class implementation, our private data members will consist of a vector of strings called rec. Our public implementation will the constructor *Record()*, which will take in an int. This function will be a constructor that allows creation of a record of arbitrary size, and initialization of the entries to a null string.

Next, we will have *get\_size(),* which will return the size of the record, or in other words, how many entries it has. We will then have *set\_entry(),* which will take an int and a string, and *get\_entry()*, that will take in an int i and return rec[i]. These two functions will act as accessor functions, enabling users to access individual entries in the record.

---------------

**Team Organization**

In generic terms, each member will be responsible to implement their respective class, where they will be the main contributor. However, members will have the freedom to implement functions in other classes if requested by its main contributor. Below is a general plan for how we plan on splitting classes.

**Database:** Jacob Vacek

**Query, split from Database:** Srishti Sanghvi, Jacob Vacek, Jialu Zhao

**Table:** Jialu Zhao

**Record:** Srishti Sanghvi