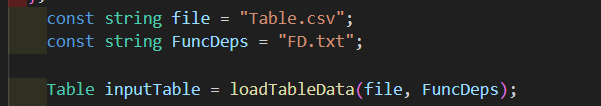
**Project Outline**

By: Dawson Ploudre

1. **Input Parsing**

The first thing that our program does is parse a .csv file that contains the file we plan on normalizing. It also parses another .txt file that contains the functional dependencies for that table. We do this by calling the loadTableData function that takes the two files as arguments. This function generates a Table based on the information provided in the .csv and also adds the functional dependencies from FD.txt. This function also prompts the user to input the keys of the table. 

1. **Helping Functions**

At the very top of the code, I added many helping functions that will be used to assist the normalization. An example of one of these helping functions is identifyDataType, which determines the type of data of each attribute so that we know how to output in the sql file.

1. **Conversions**

1NF: In order to normalize to 1NF, my program analyzes the attributes of the original table and then finds any multi-valued attributes. It is able to achieve this by finding any attributes that start and end with brackets which indicates a multi-valued attribute. Then, the program creates new tables for all of these multi-valued attributes and returns a vector with all of those new tables.

2NF:

In order to normalize to 2NF, my program first converts the table into 1NF and then identifies any partial functional dependencies. It is able to do this by finding any attributes that aren’t fully reliant on the primary key of the table. It then generates new tables for the partial functional dependencies and returns a vector with all of the tables.

3NF:

In order to normalize to 3NF, my program first converts the table into 2NF and then identifies any transitive dependencies. It is able to achieve this by finding the transitive dependencies and then creating new tables for each of the transitive dependencies. It then returns a vector with the new tables.

BCNF:

In order to normalize to BCNF, my program first converts the table into 3NF. Then, it finds any functional dependencies whose left-hand side is not a superkey. Then, it creates a new table for this violation. It returns the new tables in a vector.

4NF:

In order to normalize to 4NF, my program first converts the table into BCNF. Then, it determines any MVDs. We are able to recognize any MVDs because they are of the form OrderID -> DrinkID | FoodID. As such, we know that if there is a | on the right hand side of the functional dependency, then it is an MVD. Using this, we are able to identify any MVDs without the user having to specify. Then, my program creates new tables for each MVD and returns a vector with the new tables.

5NF:

In order to normalize to 5NF, my program first normalizes the program into 4NF. Then, we have the join dependencies input. Using these join dependencies, we are able to identify any non-trivial join dependencies. Then, we decompose the relation into it’s sub relations until there are only trivial join dependencies. We then return a vector with the new tables that we created.

1. **Identification**

1NF:

In order to determine if the table is in 1NF, we simply look through the attributes for multi-valued attributes. If an attribute starts and ends with brackets, we know it is a multi-valued attribute and we can return false. If none are found we return true.

2NF:

* Determines if any partial functional dependencies exist by checking if the left-hand side of a FD is the whole key.
* If none exist return true
* If some exist, return false

3NF:

* Determines if any transitive functional dependencies exits
* If none exist return true
* If some exist, return false

BCNF:

* Determines for each FD, if the LHS is a superkey
* If the LHS is not a superkey, return false
* If all are superkeys, return true

4NF:

* Determines if there are any MVDs using the method described in the 4NF conversion
* If any MVDs exist, return false
* If none exist, return true

5NF:

* Determines if any non-trivial join dependencies exist
* If any exist, return false
* If none exist, return true

1. **Output**

To generate the sql queries as output, we call the generateOutput function. This function generates the sql queries to create all of the tables that are given to it as an argument. IT does this by creating a table for each one, and then identifying all of the attributes and the type of each attribute. After this, it outputs the sql query into output.sql.