Project Documentation

Structure:

Python is the selected programming language for the implementation of the normalizer. In order to make the normalizer work, I created two python files, one is schema_parse.py and the other one is the normalizer.py.

I didn't include the main.oy file, in which I was originally planning to use to output a SQL file or text description file of the normalized table. Instead, I just use the print() method in the normalizer.py, and it will print all the content regarding the users' selection. So the main.py file does not exist in my project folder.

Also, there is a __init__.py file included for the package import purposes.

Data:

The input data is the *CoffeeShopData*. I modified it a bit, so it only includes the original 25 attributes table and primary keys and the functional dependencies.

SchemaParser:

For schema_parser.py, the SchemaParser class include several functions listed and explained below:

def __init__(self, file_path)

Standard initialization function which initializes the filepath, the table(stored as a dictionary with "*CoffeeShopData*" as the key) ***Note: My program is extremely data dependent.***

And the rest stores primary keys, functional dependencies(here I categorized them as non_atomic ones, regular FDs and the MVD, all stored in a list)

load_schema(self)

It is a simple I/O method that reads the file path and scans the table by index and rows in the excel sheet, here it recognizes the content by matching "Primary Key" and "-->" or "-->>" to call the associated extracting functions below.

extract_primary_key(self, primary_key_text)

Find the attribute of primary keys and clean the format, and store them.

classify_dependency(self, dependency_text)

Conditionally stored different kinds of FDs in different lists, "-->>" considered as MVD FDs, only used for 4NF.

"-->" is considered as regular FDs, however, if the left, or right content includes the "non-atomic", it is stored to the non_atomic FDs and it is used for 1NF.

Normalizer:
For normalizer.py, the Normalizer class includes several functions listed and explained below:
Note: I did not do 5NF because the input parser needs to be modified and I have no clue how to deal with the restriction of a 5NF table.
__init__(self, schema_parser)
Standard initialization function to the schema parser and all the functions below

normalize_to_1nf(self)
Loop through all the non-atomic FDs and use them to create a new table and set the left side as the primary key and save the FDs, primary keys and schema(table attribute) in a dictionary, print the result. Note: I believe other test inputs won't work if they are already in 1NF since here the actual original table is the 1NF normalized tables. The attribute of right side of the non-atomic FDs will be eliminated from the *CoffeeShopData* table.

normalize_to_2nf(self)
Get information from 1nf table dictionary, loop through the regular FDs, since we don't need to worry about the non-atomic FDs and the MVD FDs, if the left side of the FDs is not a complete primary key set, then use it to create a new table, and eliminated the right side from the new "original" table passed from 1nf. Hence eliminated partial dependency.

normalize_to_3nf(self)
Loop through the information from the 2nf table, and use double for-loop and brute force to see if there are any FDs, in which those attributes are appearing in both left side and right side. If it did, use the FD in which the matched repeated attribute in the left side to create a new table, and eliminate the attribute from the original table. Store the new tables, and update the original table

normalize_to_bcnf(self)
Loop through the table dictionary and check all the FDs if their left side is the primary key of the current table. And print the output with FDs replaced as pass the BCNF.

normalize_to_4nf(self)
For 4nf, I used the shortcut and directly used the MVD FDs list, and loop through the 3nf tables(here I need to change it to 2nf, I don't know why but clearly there is some problem with storing 3nf tables after decomposition). However it worked. The MVD FDs were split and created 2 new tables, hence 4nf achieved.

selection(self)
A bunch of if statements based on the user's input and execute relative functions accordingly.