Cmput 291 Mini Project 2 Winter 2018
Design Document

Jeffrey Cheng - ccid: jeffrey2
Lab H04

Thomas Chu - ccid: tchu
Lab H06

General Overview

***The user must provide a database file with schemas according to assignments format.**

This program can do the following tasks:

1. Normalize and Decompose a Relation into BCNF
2. Retrieve Attribute Closures for a list of attributes based on given schemas
3. Check equivalence of 2 functional dependencies

(1) Given a schema name, this program will decompose and normalize to BCNF, creating new schemas in the OutputRelationSchemas for each decomposed relations. We also create new tables for each decomposed relation and populate them with data from the original relation if there is instances in database. The program will also display whether or not decomposition is dependency preserving.

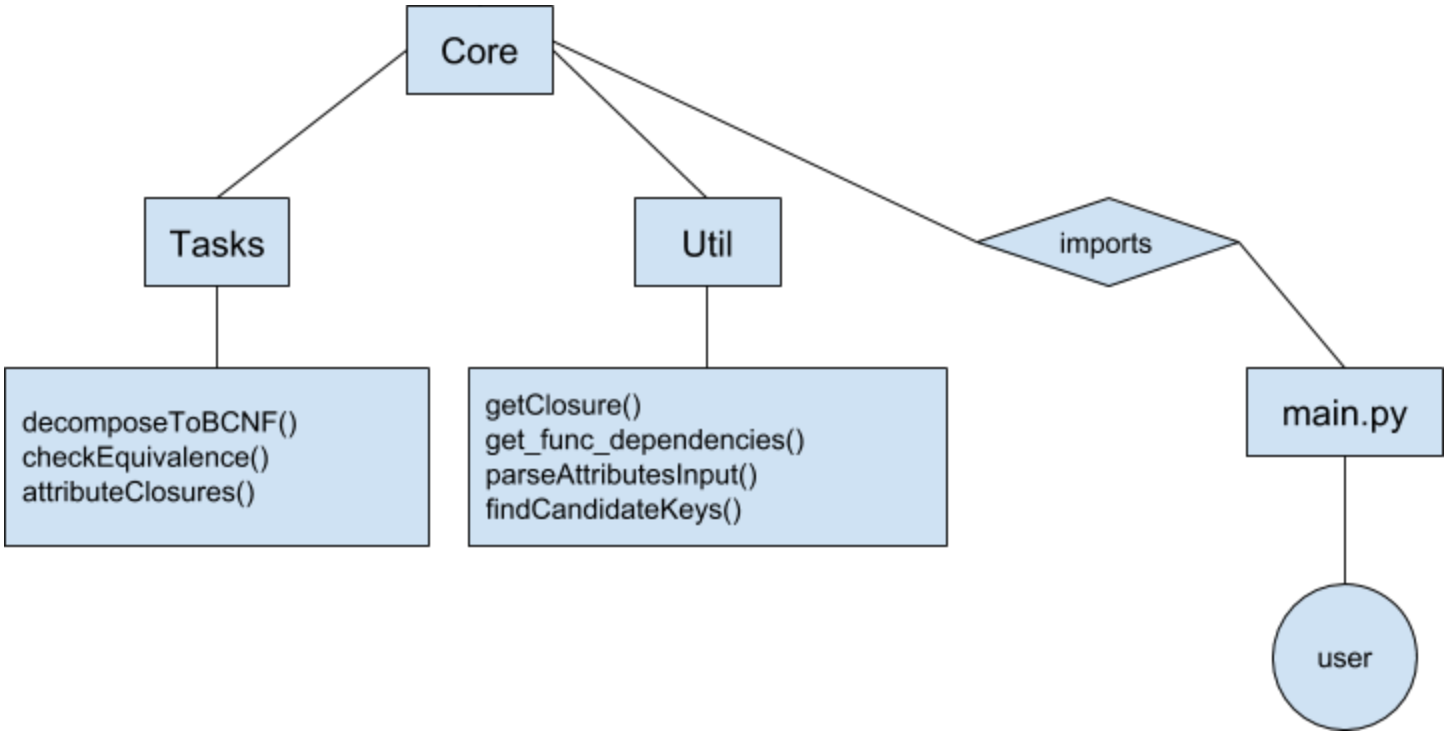
(2) Given a list of relations and attributes to compute closure, the program will display the following attributes and the list of attributes in the closure. In a format: {input_attributes} -> {attributes in closure}

(3) Given 2 list of schemas, the program will compute if the functional dependencies from both sides are equivalent to each other. Output displays “The two relations are equal!” if equivalent, otherwise “The two relations are not equal...”

Data Structures used:

FD (A->B) is interpreted as ‘A’:’B’ inside a dictionary type in python.
Closures (A+) is interpreted as a list ordered alphabetically [a,b,c,d,...]
Decomposed Relations is interpreted as a list of dictionaries such that a dictionary contains {‘Attributes’:[], ‘FDs’:{FD}}

You can view the progress of this project here:
https://github.com/ThomasChuDesigns/CMPUT291_Project2



Detailed Component Design

Core:

Tasks:

- **FUNCTION decomposeToBCNF(connection, cursor):**
  - Takes input of a table and if it is a table that resides in the inputRelationSchema, takes it's attributes and functional dependencies in dictionary form by querying the database. It then computes the attribute closures using getClosure() for each functional dependency and finds the the attributes which are superkeys or candidate keys. From there it follows the algorithm shown in the class slides for decomposition, where it finds all FDs that do not have a superkey on the LHS (BCNF violation) and decomposes into the FDs of XY and S-(Y-X). ($X \rightarrow Y$ being the FD residing in the set of attributes S) It then appends the new decomposed FDs to a dictionary, and inserts them into a new table OutputRelationSchema, also checking if they are dependency preserving by calling checkEquivalence() and seeing if they return true, printing the outcomes as it continues.
- **FUNCTION checkEquivalence(cursor, tbl1, tbl2):**
  - Takes two tables and their FDs from the database. Then computes the attribute closures for each FD in each table if any closure is not equal, they are not equivalent so return false, if all closures are equal return true.
- **FUNCTION attributeClosures(cursor):**
  - Takes the inputs of the schema and attributes the user wants to perform attribute closure on, parses those attributes and converts them to access in a list and gets the FDs from the database based on the schema. It parses those into dictionary format using get_func_dependecies(), and goes through each attribute listed in the input and calculates it's attribute closure using the getClosure() function.

Utilities:

- **FUNCTION getClosure(attribute, func_dependencies):**

        - Takes an attribute and a functional dependencies list and stores the attribute in a closure set and checks each left hand side of the FDs seeing if the left hand side is in the closure set, if it is, joins the right hand side of the FD with the current attributes in the closure set. Then sort alphabetically through the attribute closure list and return.
    - **FUNCTION get_func_dependencies(FD_list):**
        - Takes a string FD list(from database) and parses each FD into a dictionary where the left side is the key and the right side is the value in the dictionary and returns it.
    - **FUNCTION parseAttributesInput(attr_input):**
        - Takes a string of the inputted attributes and parses it, breaking up the attributes in a list and returning the list.

Main:

- **FUNCTION main():**
    - Provides the main functionality of the program, including the connection to the database, user input and calling the various tasks depending on what the user specifies.
- **FUNCTION test():**
    - Used to test the tasks without committing to database

Testing Strategy

- Each task was tested in a simple test function which included connection to the example database provided in the mini project 2 specifications and would simply call the individual tasks to see if any errors would come up and if it was providing the outputs that it should according to the example database. After each task was tested to work on their own, other test schemas were added to the example database to see if the functions still worked.

Group Work Description

- **Thomas Chu:** Completed the entire program outline, all task functions, all utility functions and the main() function. Also created the written general overview portion of the design document.
    - Estimated time: 5 days
- **Jeffrey Cheng:** Created test cases and the rest of the design document.
    - Estimated time:
- **Method of cooperation:**
    - Outline of what needed to be done created by Thomas (functions, folders ,etc) and posted on gitHub, work was divided in the way specified above, any communication on progress was done through Facebook.