CMPUT 291: Mini Project 2

Julian Stys, Lab section H07

Kyle Fujishige, Lab section H02

Alex Li, Lab section H04

We did not collaborate with any other students outside of our group

a) **General overview:**

The mini project 2 program allows users to operate on a set of schemas as specified in project 2 outline. Upon execution, the program asks the user to enter a database file. If the user enters a valid file address, the program will then perform the operations listed below when prompted by the user:

i) **BCNF decomposition:** The Boy-Codd normal form decomposition implements the algorithm given in the lecture notes. A list of table names from *InputRelationSchemas* is shown for which table instances exist, and the user is prompted to enter one of the table names. Upon completion, the resulting decomposed schemas are stored in the *OutputRelationSchema* tables, and the table and attributes are named according to the assignment specification. The program then tells the user whether or not each schema is dependency preserving. Finally, for each resulting decomposition, a new table is created which is modified from the original table to include only the resulting attributes.

ii) **Attribute Closure:** The program allows users to compute the attribute closure of a set of attributes with respect to a set of functional dependencies. For the set of attribute closures, the user inputs them directly in the format: `A,B,C` to specify the set X={A,B,C}, for instance. To specify a set of FDs, the user selects 1 or more schema from the InputRelationSchemas, and the union of these sets are used. Based on the input provided by the user, the program then computes and reports the attribute closure back to the user.

iii) **Equivalence of 2 sets of FDs:** The third functionality allows the user to determine whether or not 2 sets of functional dependencies are equivalent. To specify a FD, the user is prompted to select 1 or more schema from *InputRelationSchemas* table, and the union of the input is used as the result. This format is used to specify both sets. The program then reports whether or not the sets are equivalent.

b) **Detailed component design:**

i) project2.py is the main module that will be called directly from command line. The module first prompts the user to enter a database name, and once the user enters a valid database name, the program connects the database through sqlite3. 4 options will be displayed to the user which correspond to the 3 functions described in a), and an option to exit the program. Each function from a) is implemented in a separate module to achieve code neatness and readability, and so the modules are called depending on what the user inputs.

ii) In each module, the cursor and connection attribute is passed as parameter. This avoids the need to re-connect to the database once functions from a new module are called.

iii) Only project2.py calls each of the modules, meaning that no modules interact or call one another. This was done by choice to remove the need for multiple identical function calls and conflicting libraries. The modules that will be called by project2.py are bcnf_normalization.py, equivalence.py and attributeClosure.py

c) **Testing strategies:**

Since there are many different implementations of the BCNF algorithm, each giving different results, we tested the schemas given in the project description, as well as new ones from the lecture notes to test our code. Some of the scenarios that we tested are outline below:

- Brief testing of the database filename input functionality: tested whether filename that user enters exist, and if it doesn`t program prompts user to re-enter

- BCNF decomposition: The majority of the testing that was done with decomposition was taken from the database provided to use in the project description, as well as new schemas that we made based off the lecture notes. We tested BCNF on all 4 tables in *InputRelationSchemas* and compared the results to those found in the lecture notes and project

description page. We used the example from slide 43 of Relational Database Normalization page on eclass to generate a new table, and found that the resulting decomposition matched that as found in that example.

- Attribute closure: The testing of the attribute closure module involved mostly testing the BCNF normalization module. Since attribute closure is an important step in computing BCNF decomposition, we used the same algorithm in each of these modules. Therefore, testing attribute closure entailed a lot of user input tests, as well as testing certain examples in the *InputRelationSchemas* and on eclass lecture notes.

- Equivalence of 2 sets of FDs: Once we had the *InputRelationSchema* with additional schemas found on eclass, we tested various combinations of FD sets through this table. We then checked the validity of the results by working on them by hand and seeing if the results matches.

d) **Group work description:** The majority of the group work was towards working out how to implement the BCNF algorithm, since that was the most challenging section, and since the rest of the project could be completed by taking components from the BCNF algorithm and using them for different functions. The group work of each individual member is as follows

- Julian: Coding of the bcnf decomposition and project2.py module, discussing how to implement the BCNF decomposition algorithm. Creating program documentation (Estimated work time: 16 hours)

- Kyle: Coding of the equivalence functional sets module, dependency preservation function and discussing how to implement the BCNF decomposition algorithm (Estimated work time: 10 hours)

- Alex: Coding of the attribute closure module and discussing how to implement the BCNF decomposition algorithm (Estimated work time: 10 hours)