DESIGN DOCUMENT

CMPUT 291 MINI PROJECT 2

GROUP C291G19: KELLY CHIN (1391757), CALVIN HO (1436255), JESSICA HUYNH (1394832)

Table of Contents

General Overview

This program allows users to normalize relational schemas into third normal form (3NF) and Boyce-Codd normal form (BCNF). After normalization, the program is able to decompose the original table instance into the new relational tables. Specifically, after the normalization to BCNF, the program will indicate whether or not the resulting decomposition is dependency preserving. The program can also compute the attribute closure for a given attribute set and a table of functional dependencies. Finally, the program is able to determine whether or not two sets of functional dependencies are equivalent.

User Guide

The program does try attempt to take care of invalid user input such as extra spaces and commas. However, if it cannot it will simply state that the input is invalid.

Initial Welcome Screen

Prompts the user to input the filename of a database file (.db) that contains the input tables. The user may also choose to quit at this stage ("-quit"). Following the connection to the database, the user is then prompted for an action: [0] Synthesize table to 3NF, [1] Decompose table to BCNF, [2] Compute attribute closure, [3] Check if two FD Sets are equivalent, [4] Reconnect to database and [-quit] Quit program.

Synthesize Table to 3NF

The program will take the input relational table and its functional dependency table and decompose output functional dependency tables and output relational tables based on the 3NF algorithm. The user should see some progress messages on the screen, followed by the message "Synthesis into 3NF complete!". The user should then be able to see the new tables in their database. The user is then given the option to decompose the original table instance into these output tables.

Decompose Table to BCNF

The program intakes a relational and functional dependency table and brings it into BCNF normal form if it is not already. The program will display all the initial relations and functional dependencies, then displays if it is in BCNF or not. If it is not in BCNF, the relations and functional dependencies in BCNF will then be displayed followed by a statement if it dependency preserving or not. Database will contain tables of the relations and functional dependencies. The user will have the option to fill these tables with data from the original tables.

Decomposing Original Table Instance

Once the desired decomposition is performed on the input schemas, the user will be given the option to fill up the output tables with the data contained in the input relational schema instance. The tables are not printed and must be checked separately using a db browser of sqlite shell.

Compute Attribute Closure

The user is prompted to input a set of attributes in the form: A,B,C or A or B,H etc. After this the user is prompted to enter the table name of the set of functional dependencies to be evaluated. If this table does not exist, the user is prompted again (the user may also quit at any time). The closure is then computed and shown to the user in the form: A,B+ = A,B,E,D (for example).

Checking Equivalence Between Two Functional Dependency Sets

This option is contained in the main menu and prompts the user to input the tables pertaining to the first fd set they wish to compare with and afterwards is asked to input the tables for the second fd set. Only two sets are allowed to be compared at a time. The input must match how it appears in the database. If the user wishes to provide an fd sets that is the union of various tables, each table name must be typed properly and separated by commas.

Algorithm Descriptions

Third Normal Form 3NF

The following pseudo algorithmic steps to normalize a schema into 3NF was taken from Dr. Sander, CMPUT 291 F16. The implementation was completed by Jessica Huynh.

Get Minimal Cover Fm

The FDs were extracted from the table following the naming convention "Input_FDs_", into a set of tuples (LHS, RHS). The minimal cover is done in three steps.

1. Make the RHS of all FDs into single attributes: The program makes a temporary set of FDs (empty) and adds a tuple automatically if there is only one attribute on the RHS. If there are more than one attribute, tuples are created with the same LHS and split up RHS, and added to the temporary set. This set then is returned in the function.
2. Remove LHS Redundant Attributes: the program will loop through each FD and if the LHS has more than one attribute, it will try to remove one attribute at a time and compute its closure. If the original RHS of the FD is contained in the closure than the attribute that was removed was redundant. It is them removed from the FD set LHS. The function is recursive to ensure all redundant LHS attributes are removed.
3. Remove redundant FDs: the program loops through each FD and temporarily removed the FD in question. It computes the closure of the removed FD's LHS attributes, in terms of the FD Set (with itself removed). If all of the attributes in the RHS of the FD is not contained its closure, then it is not redundant and keep it. The function is recursive to ensure that all redundant FDs are removed.

Partition U into sets U1, U2,..., Un where LHS of Ui are the same

This simply checks if there are more than one occurrence of the LHS Attributes of each FD in the set and joins them together as a set of tuples.

Form Schema For Each Ui, Ri = (Ri, Ui)

The program will compute Ri based on both the LHS of each partitioned set, plus its RHS. For example, Ri = BHCK for a Ui ((‘B,H’, ‘C’), (‘B,H’, ‘K’)).

If No Schema Includes the Superkey of R Add One

This takes in the keys of the input table, therefore the entire set of attributes and compares them to the closure of each FD's LHS attribute set. If any time the closure equals that of entire attribute set, we prove that the scheme includes the superkey and will not add a new schema. If not, it will take the FD that has the most amount of attributes in its closure and augment to both the LHS and RHS the missing attributes. The program then does a similar function that will remove redundant LHS from the new schema. It checked if the closure of a LHS with one attribute removed can still be the super key and if it can it is redundant. This is done recursively until all redundant LHS attributes are removed.

The program finishes off by creating the tables for the output schemas and output FD tables.

Boyce-Codd Normal Form BCNF

The program will find any BCNF violating FDs. If there's none it does nothing other than create the tables. If there are violating FDs, in laymen terms, the current relational set is subtracted by the right hand side of the violating FD. Then from the current set of FDs, if a f in FD has an attribute of the right side of the violating FD, remove it from the right side of f. If the right side is empty remove that f. If the left side of f has an attribute of the right side of violating FD, remove f. Keep going until no more FDs exists or no more violating FDs exists.

Decomposing Original Table Instance

Assuming the output tables always exist, a query is calculated using sqlmaster for the all the output tables. Once we have all the output tables, they are iterated through individually to grab the attributes of that table from its name. The attributes will then be used to query the input table to get the data stored in those respective attribute columns. This list will be turned into a set to since we are only considering a few values of the original row, these values may not be unique among other rows and duplicates will not be allowed to be added into the output schemas due to primary key constraints. Each row with the extracted attribute data will be added to the output schemas sequentially.

A new cursor is used each time to ensure the database is updated with the previous insert.

Attribute Closure

To start closure, the program makes a tuple (LHS, RHS) that is equal to (LHS, LHS) of the attribute set in question (here LHS equals the attribute set). The program then loops through the given FD set and if the LHS of the FD set is fully contained in the RHS/closure then we add these attributes to the RHS/closure. The function is called recursively until no more attributes can possibly be added.

Equivalence of Two Functional Dependency Sets

According to the definition of Equivalence, a set of FDs (F and G) are equivalent if each fd inside F entails G and vice versa. F and G do not have to contain the same fd's, their semantic meaning(entailment) should be the same to imply equivalency. Following the algorithm found on this site: *http://www.mathcs.emory.edu/~cheung/Courses/377/Syllabus/9-NormalForms/FD-equi.html*

We first check each fd inside F and calculate the attribute closure according to the fd's LHS with respect to G. The attribute closure will give us all the entailed fd's in the form LHS -> 'result'. If the RHS of our fd is a subset of 'result', it can be safely assumed that LHS -> RHS is entailed by G due to the Reflexivity rule of Armstrong's Axioms. The same process is repeated for set G, this time the attribute closures of the LHS of each fd is calculated with respect to F. At the very end, if each and every fd is contained inside the attribute closures we can say these two sets are indeed equivalent.

Testing Strategy

Test Scenarios and Coverage

3NF: the example databases given, hardcoded tests, and class/lab examples were used to test each step of computation.
BCNF: used example inputs and some examples from the labs and lectures
Attribute Closure: with each input FD table, random attribute sets were testing and compared to hand computed closures as well as class examples were hard coded to test
FD Set Equivalence: Using the examples provided, a table should be equivalent to itself and the example provided on the previously mentioned website was also tested. Union-ed fd sets were not tested in depth.
Decomposition of instances: The examples were tested by deleted the data inside the output schemas and verifying visually that the same data was placed back into them.

Assumptions and Limitations

A few assumptions had been done. One assumptions is that the naming convention for input tables and the input FD table are restricted to the form "Input_" and "Input_FDs_" respectively. Since the program automatically finds these tables when computing 3NF and BCNF, databases that do not follow these conventions will not be able to run properly.
Another assumption was when computing closure, if the attribute set given contains an attribute that is not contained within the FD set, the program simply adds it to the closure and no error is given. Example: FD Set A->B and C->D. Containing attribute C+= CD and a attribute that is not contained will simply return something like Q+= Q.
When inputting the fd tables for equivalence, the user input is assumed to be correct and also the table that is mentioned is assumed to exist inside the database. During decomposition of a schema instance, the output tables must be empty because we cannot fill in duplicate data due to key constraints.

Group Work Break-Down Strategy

This mini project was coordinated and tracked using Github. Git issues were created, assigned, and closed as a to-do list as each member tracked their tasks. The versioning of different code was addressed using Git branches.

Kelly Chin

- Equivalent FD Sets (2h)
- Decomposing Schema Instance (6h)
- Document (30 mins)

Calvin Ho

- BCNF (10hrs)
- Dependency preserving (2 hrs)

- Document (30 mins)

Jessica Huynh

Main responsibilities were:

- General skeleton of code/user prompts (2 hours)
- Attribute Closure (30 mins)
- Third Normal Form 3NF (5 hours)
- Design Documentation (2 hours)