Andrew Plum and Nathan Nguyen

CS 360

11/27/2023

Evaluating Functional Dependencies

Introduction

We were tasked with the creating of an evaluation system with the ability to grade student submitted proofs involving functional dependencies. This project is the next step in the creation of a tutoring system concerning concepts in normalization and database design theory. The majority of our project was developed on the backend. Our work is to be integrated with the existing work of previous projects to reach a complete tutoring and evaluation system. The grading system we created utilizes set definitions of functional dependencies. To understand our methodology, it is required to delve into the theory database design and normalization.

Background and Theory

Functional Dependencies

In database design theory, functional dependencies serve as the basis for defining relationships between attributes within a relational database. Let us consider the relation R with attributes A and B, denoted as R(A, B). In this relation, we assert that B is functionally dependent on A, represented as A → B. This implies that for every valid instance of A, there exists precisely one corresponding instance of B, meaning a unique relationship between the two attributes. Understanding and utilizing functional dependencies is of paramount importance in the process of database normalization to ensure lossless join decomposition in the relations of our database. Lossless join decomposition is a desired property of the relations in a database as we can then be confident in the integrity of the data in the database. A real-world example of a functional dependency in a relation would be a relation “Students” with attributes “Student ID”, “Student Name”, and “Course ID” where “Student ID” uniquely determines “Student Name” which can be expressed as the functional dependency “Student ID → Student Name” between the two attributes.

Proofs Involving Functional Dependencies

Various proofs involving functional dependencies can be done all with the goal of normalizing the database. In the tutoring and evaluation system being designed, students will submit these aforementioned proofs so that their understanding of concepts in normalization and database design can be evaluated and guided towards improvement when necessary. The proofs we were tasked with grading were proofs for the logical consequence of F and attribute closure.

Set Definitions of Functional Dependencies

Applying the set definitions of functional dependencies plays an integral role in our evaluation system for the logical consequence of F and attribute closure. This is because the set definitions are used as rules which are referenced in each of the steps in the proofs. The set definitions that are used as rules are listed here:

1. Reflexivity: If X, Y ⊆ U and Y ⊆ X, then  
   X → Y always holds.
2. Augmentation: If X → Y and Z ⊆ W ⊆ U , then XW → Y Z.
3. Transitivity: If X → Y and Y → Z, then X → Z.
4. Union Rule: If X → Y and X → Z hold, then X → Y Z holds. Recall Y Z = Y ∪ Z.
5. Decomposition Rule: If X → Y Z holds, then X → Y and X → Z hold.
6. Pseudo-transitivity Rule: If X → Y and Y Z → W hold, then XZ → W holds.

Proofs for the Logical Consequence of F

The logical consequence of F, where F denotes a set of functional dependencies, encompasses all of the functional dependencies which can logically be derived from F. An example of a logical consequence of F would be where F is the set {A→B, B→C} and where we say A→C is logical consequence of F by transitivity between the first and second given functional dependencies in F. To see if a functional dependency is a logical consequence of F, a proof can be performed where each step in the proof references a rule to demonstrate that the new functional dependency is a logical consequence of F. If F is a set of functional dependencies, the logical consequence of F involves finding all of the functional dependencies that logically follow from F, F\*, which means that the attribute closure of F and F\* should be the same.

Proofs for Attribute Closure

Attribute closure is all the possible attributes which can be derived, X+, when given an initial set of attributes, X, and set of functional dependencies, F. X+ is found by repeatedly applying set of attributes you currently have to the set of functional dependencies until no more attributes can be added to the current attribute set. Relating this to the previous example, if given X={A} and F={A→B, B→C}, the attribute closure X+ would be {A, B, C} as it encompasses all the attributes implicated by the given functional dependencies. A proof can be performed to derive X+ where each step references what new attributes can be derived while referencing the functional dependency in set F in which it was derived.

Methodology

Programming Languages and Tools

We decided to use the text editors Notepad++ and Visual Studio Code to write our code. Git and GitHub were used for version control. Since our project was mainly focused on backend operations as well as storing and retrieving data from the database we designed, we decided to use the programming languages PHP and SQL. We also utilized XAMPP and PhpMyAdmin in the development of this project.

Notepad++ and Visual Studio Code

Notepad++ had the advantage of being a lightweight text editor; it offers syntax highlighting and various plugins for coding to streamline the development process. Visual Studio Code on the other hand has even more robust extensions and features to offer than Notepad++ and was used for as an environment for debugging as well as coding. These tools in combination facilitated the coding development process due to their different strengths.

Git and GitHub

Because this was a collaborative team project, we needed some form of version control for the code we developed. We chose to leverage Git through GitHub. GitHub has the benefit of having an easy user-friendly graphical interface to utilize Git. GitHub allowed us to track and manage the changes we made to the code as well as reverting changes to previous versions if needed. For collaborative coding the pull request and branching features were invaluable for our project as it allowed us to work on separate features simultaneously and merge the contributions seamlessly. It also encourages the development of more modular programming, which is always a desired feature in one’s code. These features all ensured a more organized and cohesive codebase. We also utilized GitHub’s feature to host our code in a repository there.

XAMPP

XAMPP, a cross-platform web server solution, was used for testing and debugging of PHP applications in its local development environment. Because it is consists of Apache, MySQL, PHP, and Perl, XAMPP simplified the setup of a complete web server stack on our local machines, and was chosen for it straightforward installation to facilitate a consistent development experience and compatibility between team members and the products.

PHP

Hypertext Preprocessor (also known as PHP for short) is a server-side scripting language which is very similar to C in its implementation. It allows for dynamic content generation and clean integration with web applications on frontend and databases on backend which was required feature for the programming language we needed for this project. We could have developed the backend of this project in python, but ultimately chose PHP because most of the preexisting tutoring project already used PHP so we decided to be consistent with this design choice; it also had the added benefit of being a fun programming language that we had never used before.

SQL

For database management and manipulation Structured Query Language (SQL for short). SQL allows for efficient retrieval, insertion, modification, and deletion of data in the database; and the specific distribution of SQL we used for this project was MySQL. The language’s standardized and natural English like syntax in conjunction with its relational database capabilities all helped in the successful development of the database for the project.

PhpMyAdmin

PhpMyAdmin was utilized because it offers a graphical user interface for MySQL within the XAMPP environment, making database administration more straightforward and efficient. It simplified features helps us in performing tasks such as executing SQL queries, creating tables, and managing relationships between and within tables. PhpMyAdmin enhanced our ability to understand the underlying data structure of the database because of its ability to visualize and manipulate the database schema.

Grading and Feedback

We decided to start simple in the grading scheme where a student proof submission had to state after being graded: it was correct, or it was incorrect. How we would determine if the student’s proof was correct would be if each of the steps of the proof was valid and then if the final step reached the desired correct answer the proof was trying to prove. In checking if each of the steps of the proof are valid, we would check in the sequential order in which they were written. If all the steps in the proof were determined to be valid, then we would determine if the desired correct answer was reached in the last step. If this was true, then the students’ proof would be graded as correct. And if any of the checks we performed on each of the steps of the proof failed, then we would stop where the first check failed, and the students’ proof would be graded as incorrect. In addition to indicating that the students proof was incorrect, we would return additional feedback of what step the specific check failed in the evaluation process. The additional feedback returned did not include whether every step after the failed check was correct because those steps may be considered as correct by the grading system we designed even though they were referencing an incorrect step.

Specific Grading Cases

Steps that give what could be a correct functional dependency but have an incorrect justification are marked as incorrect. Also, because unnecessary steps (steps which are not referenced by future steps) under the current grading system are marked as valid, they will still lead to the proof being graded as correct even though the proof is not concise. We decided to make this the case because unconcise proofs are still considered just as correct of a proof as a concise proof. One other thing to note is we did not have to worry about marking steps that would be valid if preceded by the correct steps as incorrect if out of order and not preceded by the correct steps because there would be no way the student could reference the previous steps if they did not already write out those previous steps to be referenced; this was enforced by the nature of the interface the student is working with when writing their proof. If the student did attempt to write a correct proof by writing what is a correct future step, then the step would be marked wrong because it has an incorrect justification because it could not be referencing the correct preceding steps that the student hasn’t yet written.

Binary Implementation

We had to figure out how to represent the functional dependencies so that the set definitions could be applied to each of them and their derived functional dependency in each step of the proof. We wanted to keep the way that the functional dependencies were when we were working with them and how they were when we stored them in the database consistent with each other. Noticing that each attribute in each functional dependency had to state – it was either present in its respective side of the functional dependency or it was not present – we decided to represent each of the functional dependencies in two binary numbers. So, if we have the relation ABCDE and we have the functional dependency AB→E, this functional dependency would be represented by binary number 11000 for its left side and 00001 for its right side. Similarly, if we have the functional dependency ACE→BDE, the left side of it would be represented by 10101 and the right side of it would be represented by 01011. Because we were storing the functional dependencies in binary rather than some other sort of data representation like a string, there was the ability to easily convert to and work with numbers in decimal and hexadecimal. Storing the functional dependencies in binary also had the advantage of being able to work with all the features and operations of numbers in the programming language we were working with.

Binary Operators

What was particularly useful about the binary representation of the functional dependencies is that we could use the binary operators when applying the set definitions. For example, if you have set X and set Y, X can be described as a subset of Y if the AND between X and Y is equal to X. And because X and Y are represented in binary in our grading system, we can simply use the built in AND operator (in PHP it is “&”) to apply the AND (so it would be written “X & Y = X”). So if set X={B} and Y={ABC} then it would be determined X is a subset of Y but Y is not a subset of X. X would be a subset of Y because X in binary is 010 and Y in binary is 111 and applying the rule accordingly 010 & 111 = 010. As another example, the difference between sets X and Y can be represented by the XOR operator (in PHP it is “\”). Determining the difference between sets can be useful in determining if a functional dependency is derived by decomposition from another functional dependency. By using the set operators, we can easily apply all 6 of the set definitions when checking if a functional dependency can be derived through one of the definitions.

How the Binary Operators are Applied in Each of the Set Definitions

MAYBE NATHAN CAN FINISH THIS PART

I’D ALSO LIKE IF YOU CAN TO WRITE ABOUT THE SPECIFIC CASES THAT WE INCLUDED AS VALID AND THE ONES WHERE WE DIDN’T – REMEMBER YOU HAD A BACK AND FOURTH WITH OUR PROF WHERE YOU WERE SAYING IF ITS LOGICALLY CORRECT THAN WE SHOULD HAVE IT AS MARKED CORRECT AND HE WAS SAYING NO – I THINK IT WAS OVER THE EMPTY SET AND THINGS LIKE AUGMENTATION

Database Design Implementation

MAYBE NATHAN CAN FINISH THIS PART

Display Design Features

Our project was primarily developed on the backend. However, we still needed an effective way to display the results of our algorithm. The graphical user interface is designed to present the results of a testing evaluation algorithm for proofs involving functional dependencies for the logical consequence of F and attribute closure. The interface is primarily focused on displaying the process of evaluating logical consequences and the resulting proofs in a clear and organized manner.

Display for Logical Consequence of F

Initial Question Information

Features of the interface and display of results we created include the ability to evaluate student submissions of proof using specific Submission IDs corresponding to the specific submissions made by each student, allowing for the tracking and referencing of specific submissions. Each submission displays the initial question being asked. This includes the kind of question being asked along with the logical constraints given as a set of functional dependencies, and what the student is trying to prove.

Student Answer

Evaluation Results and Feedback

FINISH HERE

AI WRITE DISCORD CONVERSATIONS

and provides a clear structure.

The interface supports the input of student entries using precise technical language such as Pseudo Transitivity and Augmentation. The user interface then presents the justification for the entries, providing immediate feedback on correctness. Moreover, the interface utilizes a validation mechanism to determine the correctness of each step in the evaluation process, and visually indicates whether the submission is correct or incorrect.

The graphical user interface effectively combines technical elements such as logical constraints, functional dependencies, student entries, and justifications in a structured and well-organized format. This allows for efficient evaluation and review of proofs involving functional dependencies and logical consequence in a classroom or experimental setting.

FINISH HERE

Display for Attribute Closure

DESCRIBE THE INTERFACE HERE

Testing and Evaluation

From all the testing we have performed, we are confident that our algorithms will perform as desired. For one, we believe that our application of the underlying theory is sound. To be sure of it, we have performed extensive testing. We tried to perform a variety of tests where our answers highlighted different applications of the theory in order to catch as many edge cases as we could. Some of these tests have included cases provided by our instructor as well as cases designed by us. If you are not yet confident of our algorithm implementation, it is easy to create and perform your own tests. To do so the data should first be entered into the database, and then on the web page interface you can perform the tests of the cases entered the database. From there, you can visually evaluate if our results align with what they should actually be.

Future Improvements

Our Accomplishments

Despite all of the future improvements that can be made to the project, we have still achieved a great deal with this project. Many of the milestones of the project have laid the groundwork for future groups. One of our primary accomplishments was the creation of evaluation algorithms for questions concerning proofs with the logical consequence of F and proofs for attribute closure. We also designed a database and successfully queried it. Additionally, we developed a testing display which showcases the testing results in a clear and informative manner. One of the more important aspects of our work involves how we creatively applied the theory surrounding the algorithms through the binary implementation. The solution we designed ourselves was arrived at after spending considerable time brainstorming and exploring what would become failed attempts at solutions because we initially had no similar problem to model ours after. The solution we designed can now be used a reference for future project groups, which means they will be spared the intensive time that we took to develop our solution as their implementation should be quite straightforward as they will only have to expand on the work we have using the approach we created.

Conclusion

My partner and I take pride in the accomplishments we have achieved. Our innovative approach to creating evaluation algorithms for the logical consequence of F and attribute closure required conscientious thought and extensive brainstorming. Along with algorithm implementation, our design and successful querying of a tailored database showcased our technical proficiency. The binary implementation of our work was notable as it will save future groups of the time we undertook in the creative design process. While we acknowledge there is always room for improvement with our project, we have provided a project that future contributors can easily build upon; it will be interesting to see how they expand upon it. Ultimately, we think our project stands as a testament of our learning this semester.