FD-EXPLORER: A Pedagogical and Design Tool for Functional Dependency Exploration

Julian M. Scher
Department of Information Systems
College of Computing Sciences
New Jersey Institute of Technology
Newark, New Jersey 07102 USA
Scher@adm.njit.edu

and

Canghui Qiu
Department of Electrical and Computer Engineering
Newark College of Engineering
New Jersey Institute of Technology
Newark, New Jersey 07102 USA
Cq2@njit.edu

Abstract

Functional dependencies are merely a type of relationship between attributes in a relation, or, alternatively, may be viewed as constraints on attributes, but their importance in the optimal design of databases is enormous. Normalization of a database, and the decomposition of relations, are totally dependent upon the database designer being able to identify functional dependencies, and manipulate them. Curricula in CS, IS and IT will almost always include a course in database design, with functional dependencies being a key topic in such a course. FD-Explorer is a new tool we have developed which enables both the student of database design, as well as professional database developers, to define a known set of functional dependencies on a relation, deduce new sets of functional dependencies, compute closures of individual attributes and the set of functional dependencies, and identify superkeys. This software tool, which we ultimately intend to make freely available for students in database design classes in institutions of higher learning, will provide the user with significant insight into the underlying explicit and implicit relationships between attributes, contribute to the optimal design of database structures in applications, and enhance the user's understanding of the fundamental principles of functional dependencies.

KEYWORDS: Functional dependencies, database design, Armstrong's axioms, normalization, attributes, closure.

1. DATABASE DESIGN IN THE IS CURRIULA

The capability for an Information Systems professional to understand, apply, and design database applications has been a key component in the various IS curricula recommendations issued by ACM and other professional organizations. For instance, in "IS2002 - Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems," jointly developed by ACM, AIS and AITP, a course in database design is one of ten required courses recommended for all students majoring in Information Systems (Gorgone et. al, 2002).. The bulk of database design course material for IS2002 is focused in the recommended course IS2002.8 (Physical Design and Implementation with DBMS), but also appears in IS2002.7 (Analysis and Logical Design). Furthermore, in the formal accreditation standards established by the Computing Accreditation Commission for Information Systems curricula,

Database Management is one of six areas required to be in the core of every fully accredited Information Systems curriculum (Computing Accreditation Commission, 2004).

The Year 2001 Model Curricula for Computing (CC-2001), created by a Joint IEEE Computer Society/ACM Task Force to update the 1991 curricula recommendations of the group, released the Strawman_Report in March, 2000, detailing the recommendations of this group. Information Management (IM) is identified as one of the thirteen 'knowledge areas' for computing disciplines, and IM4, Relational Database Design (functional dependencies, normalization, etc.) is identified as one of the eight components of the Information Management core. A previous discussion of the role of Database Design in the Computing curricula may also be found in (Mohtashami and Scher, 2000), which also details the relevance of Bloom's Cognitive Domain Taxonomy in teaching database design concepts.

Database Design is thus seen to be a key knowledge area for the Information Systems professional, and it could be said that the "heart" of optimal database design is normalization, and that the "heart" of normalization is functional dependencies. In the database design life cycle, the design team will initially create a high level logical model for a relational database by using an Extended ER model, an IDEFIX data model, a semantic object data model, or a UML style data model (Kroenke, 2004). Subsequently, the data model will be transformed into a relational design. During the conceptual design process, functional dependencies and keys will be identified. The relational design process requires that the database designer scrutinize each relation, and working with the enterprise for which the database application is being developed, identify the functional dependencies, particularly those which do not involve the primary key as a determinant. Once the functional dependencies have been established, the normalization process may proceed, and the database designer will seek to structure the relations into the possible highest normal form (e.g., Domain Key Normal Form).

2. PROPERTIES OF FUNCTIONAL DEPENDENCIES

A formal definition of a functional dependency states that if R is a relation schema, and A and B are non-empty sets of attributes in R, then B is functionally dependent on A iff each value of A in R has associated with it exactly one value of B in R, and the formal notation would be A→ B, where A is referred to as the determinant, and the attributes on the RHS are referred to as the dependent. A→ B is formally read as "A functionally determines B." Functional dependencies can also be viewed as integrity constraints, which every instance of the database must obey.

In identifying functional dependencies between attributes in a relation, it is crucial that we distinguish clearly between the values held by an attributes at a specific point in time, and the set of all possible values that an attribute may hold at different times. Thus, a functional dependency is a property of a relational schema rather than a property of a particular instance of the schema. (Connolly and Begg, 2002).

In surveying users to obtain the necessary information for a database, (Pratt and Adamski, 2002) recommend a design methodology based upon a survey form, which helps to identify entities, attributes, relationships, and functional dependencies. It is acknowledged that users probably will not understand what a functional dependency is, and it is critical, then, that appropriate questions are asked in the survey to help one identify functional dependencies. Appropriate questions would be very specific, such as "If you know a particular employee number, can you establish other information, such as the name?" If this fact is ascertained, then one can state that the department number is functionally dependent on the employee number. An additional question would be "Do you know the number of the department to which the employee is assigned?" If this is ascertained for all employees, one can then state that the department number is functionally dependent on the employee number. On the other hand, if a given employee can be assigned to more than one department, one could then infer that the department number would not be functionally dependent on the employee number.

The process of determining functional dependencies is not merely a task for the database designer, but must clearly involve key personnel in the enterprise, who have an intimate understanding of the relationships between the attributes that are being used in a particular relation within the relational database design.

Given a client environment, the task of identifying valid functional dependencies could present a formidable task for both the student of database design, as well as the professional. Many of the difficulties associated with this process are discussed in the foundational work by (Kent, 1978) and have much in common with identifying user requirements for a database and in various aspects of model-building. We declare functional dependencies based upon the meaning of attributes, but there is a risk that some meanings could be subjective in nature. Kent focuses on the philosophical issues on how we perceive reality and this applies to all aspects of data modeling (which include identifying functional dependencies), and the difficulties in getting from reality to a data structure through a human language. The needed information is often "too amorphous, too ambiguous, too subjective, too slippery and elusive, to ever be pinned down precisely..." (Kent, 1978). A functional dependency is a structural relationship, and as (Kent, 1978) points out "Structure is process slowed down."

A key philosophical consideration in identifying functional dependencies is the issue of there being a single objective view of the social organization for a client in our database approach. And yet the belief in the existence of such a viewpoint is often implicit in database design, including functional dependency identification. In practice, the viewpoint from which a "corporate database" is constructed is often the viewpoint of its Information Systems people. This viewpoint has its own history, its own process of development. It is not merely a snap-shot of the company information structure, it is in actuality the product of a social process.

In the ideal, the database designer should be able to identify every legitimate functional dependency; however, in reality, the properties of functional dependencies enable us to make inferences of new functional dependencies from existing ones, which, in a sense, simplify the task of the database designer. The software tool we have developed, FD-Explorer, simplifies this task even further, by guiding the user through this inference process, and automating the logical computation that enable the inference of new functional dependencies from existing ones.

The software tool we have designed and implemented, FD-Explorer, invokes several of the well-known properties of functional dependencies, which we shall review.

The classic axioms regarding functional dependencies are due to Armstrong (Armstrong, 1974). Armstrong's Inference Axioms tell us that if A, B and C are subsets of attributes of a relation R, then the following axioms will hold:

Reflexivity Rule: If B is a subset of A, then A ----> B (this implies that A -→ A will always hold, and functional dependencies of this type are known as trivial functional dependencies)Augmentation Rule**:** If A ----> B,
then AC ----> BC
Transitivity Rule: If A ----> B and B ----> C, then A ----> C

The following rules can be derived from Armstrong's Axioms:

Union Rule: If A ----> B and A ----> C, then A ----> BC
Decomposition rule:If A ----> BC, then A ----> B and A ----> C
Pseudotransitivity rule: If A ----> B and CB ----> D , then AC ----> D

We would like to be able to explore all of the functional dependencies implied by a specific set of functional dependencies, and this motivates us to define the closure of a functional dependency set. If we let F represent the set of specified functional dependencies for some relation R, then we will define F^+ to be the closure of F, consisting of all functional dependencies that may be derived from the FD's in F. By repeatedly and exhaustively applying Armstrong's Axioms (and the Derived Rules), one may obtain all of the functional dependencies in F^+. Database designers and database students have been manually doing this procedure to obtain the closure of the attribute set, but with the advent of our FD-Explorer software, the closure will be determined by the program after the user has provided the original set of functional dependencies.

Database designers are often interested in obtaining the set of attributes of a relation R that are functionally determined by a particular attribute A. This is referred to as the closure of A, denoted by A^+. A crucial use of the closure of an attribute for database designers is the identification of superkeys. (A superkey is a set of attributes that functionally determines all of the attributes in a relation.) So, if the database designer computes the closure of an attribute, and this closure of is the relation itself, then that attribute is a superkey of the relation R.

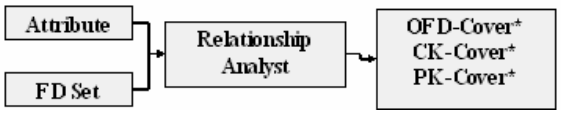
One method for obtaining the closure of an attribute A is to compute all of F^+ and then identify only those functional dependencies in F^+ which have A as the determinant, and for such functional dependencies, the union of the set of dependents will yield the closure A^+. However, a better algorithm appeals to the very definition of functional dependency, and is presented in numerous database design texts (see (Ricardo, 2004)) as follows:

Closure Algorithm for Attribute Set A
Result ←A;
While (result changes)
  For each functional dependency B→C
    If B is contained in Result
    then Result ←-Result U C;
EndWhile;
A^+ ← Result;

We also note that this algorithm to obtain the closure of an attribute has an alternative usage, and that is to help up determine whether a specific functional dependency is present in R. That is, if we have attribute sets A and B, and need to determine whether A functionally determines B, we merely compute the closure of A and observe if it includes B.

3. FD-EXPLORER FUNCTIONALITY

Our System Data Flow Diagram is as follows:



*OFD-Cover (Optimized Functional Dependencies Cover): After applying Armstrong Axioms to the user added functional dependencies, the set of new functional dependencies that we obtain is called the Optimized Functional Dependencies Cover.

*CK-Cover (Candidate Key Cover): The sets of combinations of attributes that can be uniquely used to identify a database record without any extraneous data.

*PK-Cover (Primary Key Cover): Choose from CK.

FD-Explorer provides the user interface to guide the user through the following steps:

- Creation of new attributes (via the New Attributes Input screen)
- Definition of functional dependencies using the created attributes (via the New FDs Building screen)
- Viewing of existing functional dependencies (via the Original FDs screen)
- Viewing the final report (via the Optimized FDs screen)

New Attributes Input screen: Figure 1.1 below is the initial screen provided by FD-Explorer to the user for entering the attributes associated with a relation. The attributes need to all be specified prior to establishing the functional dependency relationships between these attributes.

The Attributes table in the right panel provides the user with a full view of existing attributes (refer to Figure 1.1 below). The user can delete any attribute by first clicking on the attribute, and then clicking on the "Delete" button. Users can create new attributes by typing into the "Attribute Input" textbox in the left panel. FD-Explorer provides extensive error checking and will alert the user to all input errors, such as null values, and duplicate values.

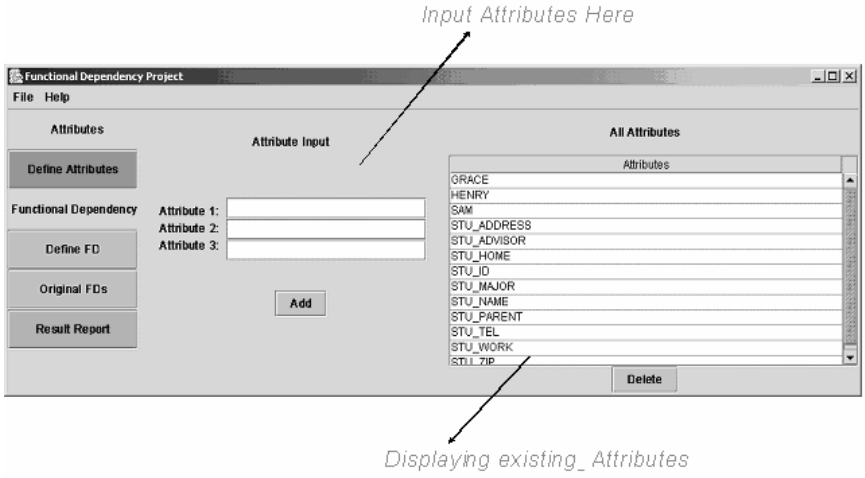


Figure 1.1

New FD Building screen: If the user clicks on the "Define FD" button under the Attributes section on the extreme left panel, FD-Explorer brings the user to the New FD Building screen of Figure 1.2.

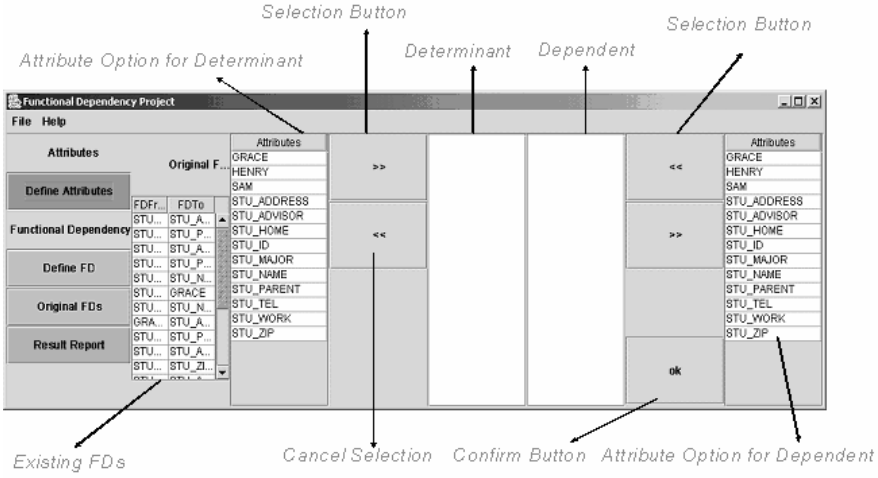


Figure 1.2

FD-Explorer will initially display a list of all existing functional dependencies (if any), so that the user can view the precise set of functional dependencies already created. The user may then start building a new functional dependency by first choosing the appropriate attribute(s) from the "Attributes" list on both sides; clicking on an attribute on the LHS will initiate the action to bring the corresponding attribute into the determinant, while clicking on an attribute in the RHS will initiate the action to bring the corresponding attribute into the dependent. The corresponding ">>" and "<<" buttons will commit the action to add (or delete) the corresponding attributes to the functional dependency being constructed. When the user has completed the building of the functional dependency, the "ok" button is clicked, the screen will be refreshed and the newly constructed functional dependency will be displayed on the FD list on the left side of the panel. (If the identical attribute appears in both the "determinant" and the "dependent," an error message with specific information will be generated to so alert the user, as in Figure 1.3)

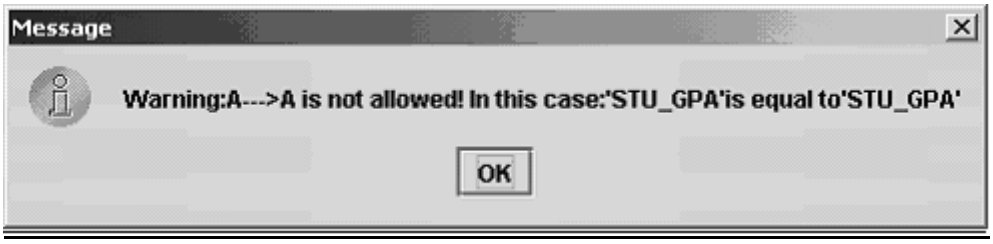


Figure 1.3

Original FDs screen: FD-Explorer will launch the "Original FDs screen" when the user clicks on the "Original FDs" button in the left hand side menu (see Figure 1.4 below). Thus, clicking on the "Original FDs" button gives the user a list of all the original functional dependencies the user created. The program will not alter this original set of functional dependencies. (A list of optimized functional dependencies determined by FD-Explorer can be viewed subsequently on the "Optimized FDs" screen.)

If the user wishes to delete any one of the functional dependencies, just select it, and then click the "Delete" button at the bottom of the screen.

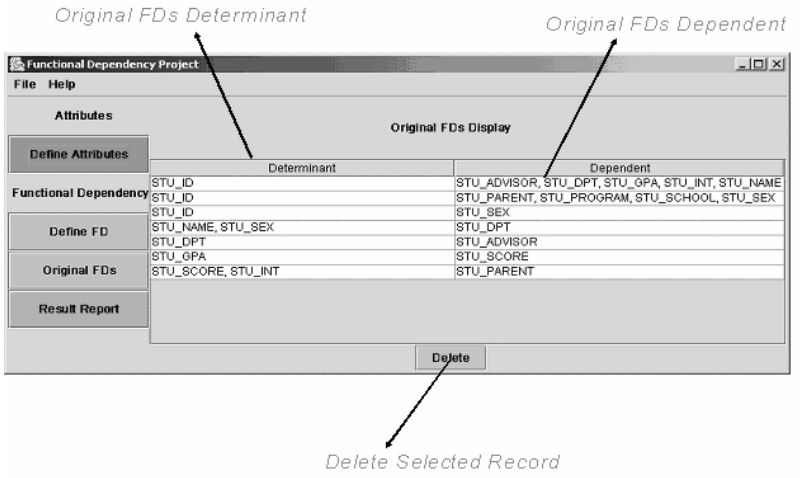


Figure 1.4

The Optimized FDs screen: This is the most crucial component of FD-Explorer. Clicking on the "Result Report" button will trigger a sequence of seven analysis steps, as follows:

1. Alphabetic Coding: Attribute names are internally coded to optimize performance.

2. Apply Armstrong's Union Rule: In this step, we determine if Armstrong's Union Rule will result in any elimination of redundant functional dependencies. For

instance, if the user has defined A->B, A->C, A->D, then Armstrong's Union Rule provides a new functional dependency A->BCD, which means that the three original functional dependencies are redundant and may be deleted

3. Transitive Rule and Reorder, eliminate the same attribute algorithm:

For instance, if A->BCD, BC->DE, then the result will be as follows::

A->BCDDE and BC->DE yields A->BCDE and BC->DE

4. Apply the Pseudotransitivity Rule to see if any new functional dependencies can be implied. For instance, X->Y, WY->Z implies WX->Z

5. Eliminate similar alphabetic code algorithm - in order to eliminate equivalent functional dependencies, this algorithm is applied. For instance: ABC->D, BC->D can be replaced by ABC->D, and thus BC->D can be deleted.

6. Sort code by alphabetic order algorithm: After a sequence of analysis and combination steps, the result will contain some duplicate values, so those combinations of alphabetic codes will be re-ordered and identical values eliminated.

7. Compute the candidate keys of the given relation R, by first determining all the superkeys (a superkey of relation R is a set of attributes which functionally determines all the attributes in R). A superkey will be a candidate key if it is minimal and contains no "extra" attributes (i.e., it has no proper subset which is also a superkey of the relation R).

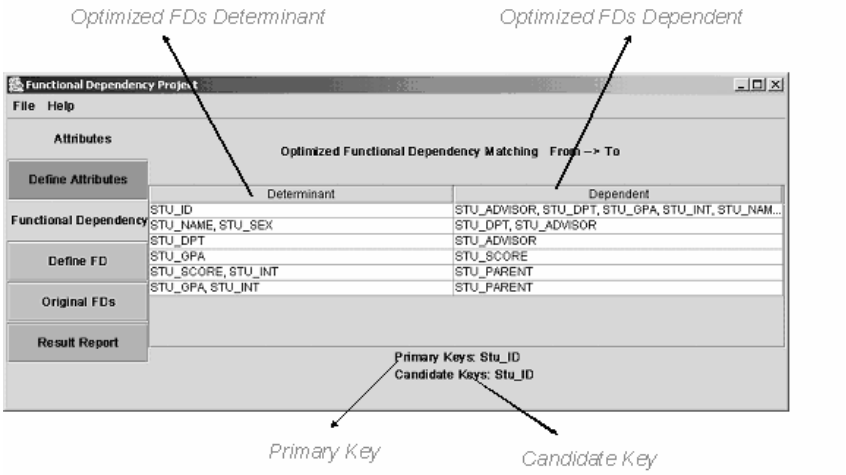


Figure 1.5

FD-Explorer also maintains a Log file (in ASCII text format), which records the detailed intermediate steps of analysis and derivation based upon the functional dependencies obtained from the initial "Define FD" phase. This Log file, which is user-accessible from FD-Explorer, provides the user with a (transparent) inner analysis perspective of how Armstrong's Union Rule, Transitivity Rule and Pseudotransitivity Rule are applied to derive additional functional dependencies. In Figure 1.6 below, we present a partial view of the Log File for a user interaction, which depicts the application of Armstrong's axioms and the derived rules.

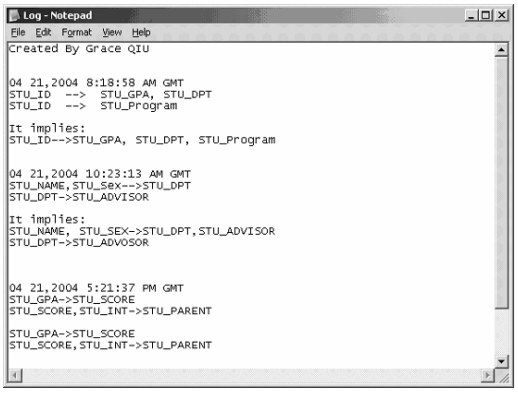


Figure 1.6

4. CONCLUSIONS

Identifying functional dependencies constitutes an integral part of the database design process, and yet, like many information design problems in the real world, represents a particular challenge to the user, whether the user be a database designer, or a student of database design. Deducing new functional dependencies from an existing set of functional dependencies is a well-understood process with known theoretical and procedural methodologies to assist us, though for a significant number of attributes, this deduction process could become burdensome. We have designed a software tool, FD-Explorer, which focuses on this process of exploring functional dependencies, and applying known theoretical procedures and rules which will both assist and instruct users. There is still work remaining in terms of usability studies of this tool with both students of database design as well as professionals, and we hope use the evaluation instrument to fine tune the tool for the future.

5. REFERENCES

Armstrong, W.W., 1974, "Dependency Structures of Data Base Relationships," Information Processing 74, J. L. Rosenfeld, Editor, pp. 580-583, Stockholm, Sweden, August 5-10, 1974. North-Holland, ISBN 0-7204-2803-3.

Computing Accreditation Commission, 2003, "Criteria for Accrediting Computing Programs - Effective for Evaluations During the 2004-2005 Accreditation Cycle," ABET-CAC, Inc., Baltimore, MD.

Connolly, T. and Begg, C., 2002, Database Systems: A Practical Approach to Design, Implementation and Management, Third Edition. Addison-Wesley, Essex.

Gorgone, J., Davis. G., Valacich, J., Topi, H., Feinstein, D., Longenecker, H., 2002, "Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems," Association for Information Systems.

Kent, William, 1978, Data and Reality, North Holland, Amsterdam

Kroenke, D., 2004, Database Processing: Fundamentals, Design and Implementation, Ninth Edition, Prentice-Hall, Upper Saddle River, NJ.

Mohtashami, M and Scher, J., 2000, "Application of Bloom's Cognitive Domain Taxonomy to Database Design," The Proceedings of the ISECON 2000 Conference, v 17 (Philadelphia): §918.

Pratt, P., and Adamski, J., 2002, Concepts of Database Management, Fourth Edition, Thompson Course Technology, Boston, Mass.

Ricardo, C., 2004, Databases Illuminated, Jones and Bartlett Publishers, Sudbury, Mass.