A CLAIM TO INCORPORATE FUNCTIONAL DEPENDENCIES IN DEVELOPMENT TOOLS
Benchmarking and Checking Functional Dependencies Algorithms

Manuel Enciso Garcia-Oliveros, Angel Mora Bonilla, Pablo Cordero and Rosario Baena
Research Group in Mathematics Applied to Computing, University of Malaga, Malaga, Spain

Keywords: Database design, Functional dependencies.

Abstract: In this work we summarize the state of the art in the use of database functional dependencies. We compare the low impact that it has in the commercial environment with its successful acceptation in the academic environment. Particularly we remark that there does not exists any commercial development tool which uses the information provided by functional dependencies and this useful information is also deprecated by the database management systems. As a result of this, large database designs have to be re-built a few years after their establishment. In this work we identify the reasons which causes this situation and we propose a guideline to wide spread the effective use of Functional Dependencies in commercial design and tools.

1 INTRODUCTION

Functional Dependencies (FD), were first defined by E.F. Codd in the relational model (Codd, 1970) and later used in Armstrong's axiomatic system. It has been well studied in the past and the design of a normalization method may be consider one the most important result to take into account to produce high quality database design.

Database designers do not explicitly use FDs as it is shown in (Antony and Batra, 2002). In fact, its use is sparse or inexistent in database design tools and they do not play a central role in DBMS (Database Management Systems) (they appears only in the form of *key* constraints).

A tool which really uses this information is the exception and it can be found in academical and research institutions with an interesting approach from the Knowledge Engineering perspective (Concepcion and Villafuerte, 1990). Unfortunately, this work is outdated and discontinued.

In this work, we summarize some result that we consider as valuable elements that may open the door to solve this problem, more concretely, we have:

- A strong theoretical background proposal: the $\mathbf{SL}_{FD}$ logic, a true executable logic for FDs.
- A careful and unified implementation of all classical FD-Closure algorithms.
- The group's $\mathbf{SL}_{FD}$-Closure algorithm, based on a free and open framework of reusable components for efficient FD-algorithm and data structures integration (common framework).
- A method to generate random test sets for FDs algorithm benchmarking with several parameters and strategies and suitable to be extended.
- An multi-language web site to illustrate the execution of these algorithms and theirs performances.

Our intention with the proposal of these elements in a unified tool is to motivate the research in the design of efficient methods to manage FDs and to promote their use and the integration into database tools.

2 THEORETICAL BACKGROUND

Throughout the literature there have been a lot of algorithms for the management of FDs. In (Mora et al., 2006) the authors carried out a empirical study of the efficiency of closure algorithms for FDs. One conclusion of this preliminary study was the need for a careful implementation of the algorithms to allow further comparisons with new methods.

Another important consequence of FDs algorithms comparisons, is the need for test suites that allow a uniform and heterogeneous comparison. We claim for a set of test which recreate different models to stress the algorithms in different directions. Nowadays, there isn't public repositories that provide clas-

Enciso Garcia-Oliveros M., Mora Bonilla A., Cordero P. and Baena R..
A CLAIM TO INCORPORATE FUNCTIONAL DEPENDENCIES IN DEVELOPMENT TOOLS - Benchmarking and Checking Functional Dependencies Algorithms.
DOI: 10.5220/0003608803130316
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 313-316
ISBN: 978-989-8425-76-8
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

sic benchmarks of FDs to verify both the efficiency and behavior of the algorithms.

Another important concern is the design of inference systems for the management of FDs that may serve as a basis for the development of automated management of FDs. Armstrong's axiomatic system is the core for many logics of FDs. All these logics have a common characteristic: they use the transitivity rule which limits their direct application and the further development of methods with efficient deduction capabilities. As a conclusion, these logics are not used in the definition of FDs management algorithms.

There exists another logic approach that substitutes the transitivity rule by a new rule and allows the creation of automatic algorithms for the management of FDs. The authors (Ángel Mora et al., 2004) presented the Simplification Logic for FDs ($\mathbf{SL}_{FD}$), which is equivalent to Armstrong's axiomatic system. The main core of $\mathbf{SL}_{FD}$ is a new rule of simplification, that allow the elimination of redundant attributes efficiently, turning it into an executable logic which opens the door to the creation of efficient algorithms.

The authors have also developed a set of algorithms to solve the most important FDs problems: the closure of a set of atoms, the redundancy removal to get FDs basis and the calculation of minimal keys.

3 A DISCUSSION AROUND IMPLEMENTATION ISSUES

All the algorithms for FDs deal with a simple data structure: FDs are represented using two associated sets of attributes. Their flow are mainly based on primitives set operators: union, intersection, difference, etc.

As (Wirth, 1978) points out, programs are not only algorithms, they depend strongly in data structures. We have made two version of the implementation of set of attributes. The two implementations are based on the representation of the set of attributes as a set of bits. So we have two versions of each DF algorithm:

Fixed. The size of the set of atoms of is fixed and each bit represents an attribute. The cost of set operators depends on the maximum number of attributes of all the FDs in the set.

Sparse. The size of the set is variable, having the same cardinality than the number of attributes on each side of the FD. The cost of set operators depend on the size of the FD involved in the concrete execution.

Sparse implementations are recommended when the number of attributes which form FD, is very low compared to the cardinality of the set of atoms. This situation may be found on those models with a significant number of "small" FDs. In models where all the attributes appears in only a few number of FDs is better tackled using the fixed approach. A degenerated version of this situation is the start models, used in data warehousing, with a central table containing the data analysis, surrounded by other smaller tables called dimension tables.

The standard algorithm (Maier, 1983) is first presented in the literature that calculates the closure in a nonlinear time: $O(\|U\|\|\Gamma\|^2)$ [1], where Γ is the set of FDs and U the set of all attributes in Γ.

In this work we have consider five different closure algorithms which appears in the literature: (Beeri and Bernstein, 1979), (Diederich and Milton, 1988), (Paredaens et al., 1989) , (Maier, 1983) and (Mora et al., 2006). Each algorithm, as we have mention above, has been development in two different versions: fixed and sparse. To compare the efficiency of these algorithms, we have also developed a method to generate random sets of FDs with different characteristics, provided by a set of different parameters detalied above.

3.1 Benchmarking of Functional Dependencies

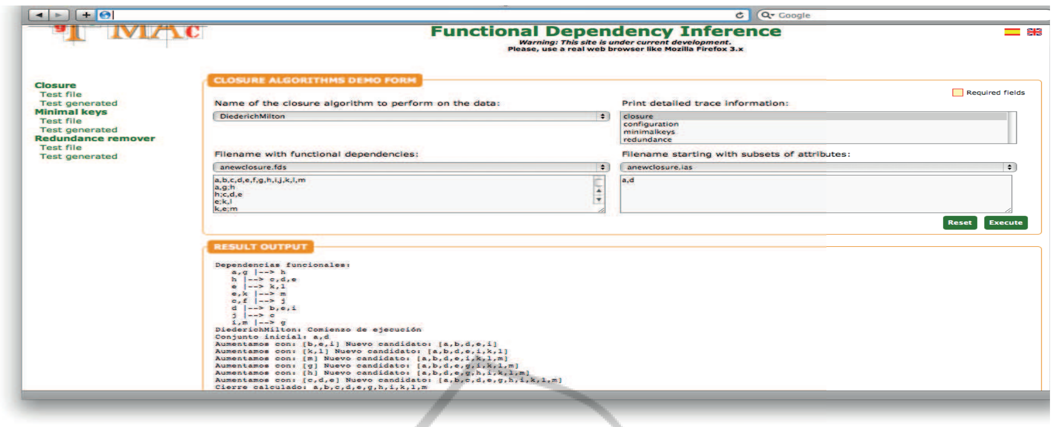
As we have mentioned, the lack of benchmarks for FDs limits the unified comparison of proofs for the efficiency and behavior of the FDs algorithms. Unless it is possible to generate random FDs sets, a more depurated method to produce set of FDs which represent different models is demanded. He have developed strategies to increase the control in the generation of FDs sets. In our approach the user can parametrize the random generation which determines the selection of an strategy. This strategies are characterized by a combination of the cardinality of the set of attributes and the size [2] of the right and left hand sides in the FDs:

Size. This is the first strategy used in (Mora et al., 2006). In this strategy, we provide a maximum level to the size of FDs and the left size is limited to $\frac{1}{4}$ of this threshold and $\frac{1}{3}$ on the right.

Vanilla. The user provides two values which represent the maximum percentage of attributes on the left and on the right. The percentages determines two separate lengths in both sides of the FDs and

[1] $\|X\|$ denotes the cardinality of X

[2] The size is defined in the literature as the sum of the lengths of the left-hand side and the right-hand side.


Functional Dependency Inference
Warning: This site is under current development.
Please, use a real web browser like Mozilla Firefox 3.x
Closure
Test file
Test generated
Minimal keys
Test file
Test generated
Redundance remover
Test file
Test generated
Reset
Execute

CLOSURE ALGORITHMS DEMO FORM

Required fields

Name of the closure algorithm to perform on the data: DiederichMilton

Print detailed trace information: closure, configuration, minimalkeys, redundance

Filename with functional dependencies: anewclosure.fds

```
a,b,c,d,e,f,g,h,i,j,k,l,m
a,g;h
h;c,d,e
e;k,l
k,e;m
```

Filename starting with subsets of attributes: anewclosure.ias

```
a,d
```

RESULT OUTPUT

```
Dependencias funcionales:
    a,g |--> h
    h |--> c,d,e
    e |--> k,l
    e,k |--> m
    c,f |--> j
    d |--> b,e,i
    j |--> c
    i,m |--> g
DiederichMilton: Comienzo de ejecución
Conjunto inicial: a,d
Aumentamos con: [b,e,i] Nuevo candidato: [a,b,d,e,i]
Aumentamos con: [k,l] Nuevo candidato: [a,b,d,e,i,k,l]
Aumentamos con: [m] Nuevo candidato: [a,b,d,e,i,k,l,m]
Aumentamos con: [g] Nuevo candidato: [a,b,d,e,g,i,k,l,m]
Aumentamos con: [h] Nuevo candidato: [a,b,d,e,g,h,i,k,l,m]
Aumentamos con: [c,d,e] Nuevo candidato: [a,b,c,d,e,g,h,i,k,l,m]
Cierre calculado: a,b,c,d,e,g,h,i,k,l,m
```

Figure 1: FDs algorithm in action.

its size is always proportional to the cardinal of the set of attributes.

Groups. It allows us a more precise generation of the FDs sets and it obtains more realistic sets, closer to real database schemas. The attributes are classified in three groups: very frequents, medium frequents, lower frequents. For each group we provide the probabilities for left and right hand sides. The methods to built both sides of the FD are different: the attributes more frequents has a bigger probability of being in the left-hand side that the others two groups. In the right-hand side the attributes with medium and lower frequency are more frequently used.

3.2 A Tool to Motivate the Use of Functional Dependencies

The proposal of this paper is to re-activate the use of FDs in real software engineering tools. To illustrate the successful management of FDs, a web application has been developed (http://sicuma02.lcc.uma.es:8080/WebTin2007). This web application provides a simple and straightforward interface for the interactive execution of closure algorithms over tree kind of input data:

1. The user supplies a concrete set of FDs. We provide a simple language to be used as a specification of FD sets which allows to test a single problem with one algorithm or with all the algorithms.
2. Examples from the literature: the user may select a FDs set and an attributes set specified in the input language.
3. Random generation of big FDs sets: an advanced FDs set generator can be used to automatically generate a big and complex set of FDs.

4 CONCLUSIONS

This work claims for the practical use of Functional Dependencies and try to promote the integration of the FDs algorithms into the software engineering tools. In this work we have concluded:

- The theoretical efficiency of the algorithms in the literature does not match with the empirical results obtained after a rigorous implementation of them.
- The theoretical studies do not pay attention to data structures.
- It is not possible to establish a uniform comparison among algorithms without the source code.
- In others areas the use of benchmarks for the comparison of hard problems is a crucial issue.

These evidences reinforce the proposal oft his work: the need to have a common framework for algorithms tests, including the definition of benchmarks, as a preliminary step to promote the use of FDs in software Engineering tools for the design and development of database schemas.

REFERENCES

Antony, S. R. and Batra, D. (2002). CODASYS: a consulting tool for novice database designers. *ACM SIGMIS Database*, 33:54–68.

Beeri, C. and Bernstein, P. A. (1979). Computational problems related to the design of normal form relational schemas. *ACM TDS*, 4(1):30–59.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

Concepcion, A. I. and Villafuerte, R. M. (1990). Expert db: an assistant database design system. In *Proceedings* *of the 3rd Int. Conf*, volume 1 of *IEA/AIE '90*, pages 333–340. ACM.

Diederich, J. and Milton, J. (1988). New methods and fast algorithms for database normalization. *ACM TODS*, 13:339–365.

Maier, D. (1983). *The theory of relational database*. Computer Science Press.

Mora, A., Aguilera, G., Enciso, M., Cordero, P., and de Guzmán, I. P. (2006). A new closure algorithm based in logic: Slfd-closure versus classical closures. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 10(31):31–40.

Ángel Mora, Enciso, M., Cordero, P., and de Guzmán, I. P. (2004). An efficient preprocessing transformation for functional dependencies sets based on the substitution paradigm. In *Current Topics in Artificial Intelligence*, volume 3040 of *LNCS*, pages 136–146. Springer.

Paredaens, J., Bra, P. D., Gyssens, M., and Gucht, D. V. (1989). *The structure of the relational database model*. Springer.

Wirth, N. (1978). *Algorithms + Data Structures = Programs*. Prentice Hall PTR.