Using a Database Query Language to Specify and Generate Spreadsheets

Jean-François Boulicaut[(1)] Patrick Marcel[(2)]
Christophe Rigotti[(1)]

(1) Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, Bâtiment 501, F-69621, Villeurbanne, France
{*jfboulic,crigotti*}*@lisi.insa-lyon.fr*

(2) Université François Rabelais
3 place Jean Jaurès, F-41000 Blois, France
{*marcel*}*@univ-tours.fr*

Abstract

Experimental studies have pointed out that operational spreadsheets contain a lot of errors, mostly due to the lack of abstraction offered by existing spreadsheet programs. In this paper, we consider the specification of a spreadsheet using a database query language and the generation of the corresponding running spreadsheet. We present a technique to perform this generation automatically, allowing to suppress the risk of human encoding errors. This technique has been implemented in a prototype that generates operational spreadsheets.

keywords: Query Languages, Spreadsheets, Specification, Automatic Generation

1 Introduction

Spreadsheet programs and their user-friendly interface have become a very popular tool for data presentation and analysis. They are used in a wide range of applications, from quick formulation of simple computation to sophisticated data manipulation in business decision support systems like On-Line Analytical Processing (OLAP) [6]. Unfortunately, experimental studies have pointed out a high rate of errors in operational spreadsheets and its increasing economical impact [4, 7, 15]. Using database terminology, it has been advocated [11, 10] that most errors come from the confusion of the spreadsheet physical level (i.e., the grid) with its logical level (i.e., the table definitions).

Important methodological elements have been proposed to design spreadsheets at a logical level [11, 10] and powerful dedicated environments have been built to support the execution of such abstract descriptions (e.g., [17]). In this paper, we show that simple spreadsheets can be specified at a logical

level using a database query language, and that the corresponding spreadsheets can be generated automatically and executed with a standard spreadsheet program (Excel [14] in our case). From a practical point of view, the main benefits are the following: (1) the developer of the spreadsheets can use a high level database query language, (2) the automatic generation avoids human encoding errors, and (3) the user of the spreadsheet can operate with it's favorite spreadsheet program and can perform ad-hoc customizations of the spreadsheet (e.g., changing the presentation, adding data and formulae).

The key idea presented in this paper is that a spreadsheet is simply a database query that has not been fully evaluated because some data are not yet available. The part of the query that has not been processed is encoded in the spreadsheet as formulae and the missing data will be entered later in the spreadsheet by the user.

As in the case of other non-standard query processing (e.g., [18]), we present the generation technique on queries written in an extension of Datalog [16] because it keeps the presentation more clear and concise. This is done without loss of generality, since various links have been exhibited between database query languages [1]. It allows to adapt processing of Datalog-like languages to any relational database query language that handles basic relational operators (i.e., selection , projection, join, union) and view definition (e.g., in particular SQL [12]).

A preliminary version of this work has been presented in [2]. In this short paper, we assume that the reader is familiar with basic Datalog syntax and semantics (see among others [16, 1]). The next section introduces informally the query language we use to specify the spreadsheets at a logical level. Section 3 presents on an example the spreadsheet generation technique, and we conclude in section 4.

2 Query Language Overview

In this section, we give a brief and informal presentation of the query language we use to specify the spreadsheets. This language is an extension of Datalog dedicated to multidimensional databases. A formal presentation of the language and its application to multidimensional databases can be found in [8, 9]. Furthermore, a survey of querying paradigms for multidimensional databases is [13].

2.1 Data Model

We use a very simple data model in which data are organized in *cells*. Cells can be seen as a logical abstraction of the physical squares of the grid of the spreadsheet. A cell is identified by a *cell reference*, and is associated with a unique *cell content*. A cell reference is of the form $T(R, C)$, where T, R and C are constants, respectively the name of the table, of the row and of the column. The associations of cell contents with cell references are represented by *ground atoms* (i.e., facts) of the form $T(R, C)$:N, where N (the cell contents) is a constant.

A *table* is a set of ground atoms having a common table name, in which the same reference does not appear more than once to ensure cell monovaluation.

Example 2.1 A table named *school* describing the results of students in mathematics and physics, together with their group membership can be represented by the following set of cells.

$\{school(kate, math)$:15, $school(kate, phys)$:12, $school(kate, group)$:$gr1$, $school(mike, math)$:14, $school(mike, phys)$:12, $school(mike, group)$:$gr2$, $school(john, math)$:10, $school(john, group)$:$gr1$, $school(alan, math)$:15, $school(alan, phys)$:16, $school(alan, group)$:$gr2$, $school(suzy, math)$:12, $school(suzy, phys)$:11, $school(suzy, group)$:$gr1\}$

A graphical representation of this table is given as figure 1.

school	*math*	*phys*	*group*
kate	15	12	gr1
mike	14	12	gr2
john	10		gr1
alan	15	16	gr2
suzy	12	11	gr1

Figure 1: The table *school*

2.2 A Rule-based Language

Rules à la Datalog are used to define new cell references and their associated contents from existing cells. We only provide here their intuitive meaning. We adopt the following conventions: symbols beginning with an upper-case letter denote variables, and symbols beginning with a lower-case letter or a digit denote constants. Consider the rule $p(X, Z) \longleftarrow q(X, Y), r(Z, Y)$. The standard (Datalog) informal meaning of this rule is: *if q(X,Y) holds and r(Z,Y) holds, then p(X,Z) holds.* The basic intuition of our extension is to read such a rule in the following way: *if there are two cells of references q(X,Y) and r(Z,Y), then there is a cell of reference p(X,Z).* We add also the handling of cell contents, and then a typical rule will be: $p(X, Z)$:$W \longleftarrow q(X, Y)$:$W, r(Z, Y)$:X. This rule means that *if it exists a cell of reference q(X,Y) containing W, and it exists a cell of reference r(Z,Y) containing X, then it exists a cell of reference p(X,Z) containing W.*

The language is also extended to specify *difference* and basic arithmetic computations by means of built-in predicates (is, $\neq$, $+$, $/$, ...) having their standard meaning.

The next example shows how these rules can be used to specify table consolidation.

Example 2.2 Definition of a column in table *school* giving the average for each student:
$school(N, avg)$:$Z \longleftarrow school(N, math)$:$X, school(N, phys)$:$Y, Z\ is\ (X + Y)/2$.
A representation of the corresponding table is given in Figure 2.

The rules can also be used to define various changes of the structure of the tables. A higher-order syntax stemming from Hilog [5], allows variables to range over every constant used in cell references (i.e., table name, row name and column name) or cell contents. The following example illustrates this kind of data manipulation.

Example 2.3 Splitting the table *school* into two tables according to the student groups:
$G(N, M)$:$X \longleftarrow school(N, M)$:$X, school(N, group)$:$G, M \neq group$.
A representation of the corresponding tables is given as Figure 3.

school	*math*	*phys*	*avg*	*group*
kate	15	12	13.5	gr1
mike	14	12	13	gr2
john	10			gr1
alan	15	16	15.5	gr2
suzy	12	11	11.5	gr1

Figure 2: Computing averages

$gr1$	*math*	*phys*	*avg*
kate	15	12	13.5
john	10		
suzy	12	11	11.5

$gr2$	*math*	*phys*	*avg*
mike	14	12	13
alan	15	16	15.5

Figure 3: A table for each group

3 Spreadsheet generation

In this section, we present how to generate a spreadsheet starting from its specification written in the query language introduced in section 2. This technique has been implemented in SWI-Prolog [19] to produce Excel [14] spreadsheets.

3.1 Principle

During the development phase, the spreadsheet is first logically modeled with the query language. Then the data available at this stage (called *initial input*) are used to generate the physical grid of the spreadsheet. When the spreadsheet has been generated, it can be opened with a spreadsheet program, and then operates on additional user-supplied data (called *deferred input*).

At the physical level in the current spreadsheet programs, a spreadsheet is simply a grid of squares that can be filled with constants or formulae making references to other squares. We do not take into account macro-command abilities, as this possibility is too much software-dependent. Starting from the logical specification of a spreadsheet, the generation process consists in three steps. Firstly, the logical specification is used to compute the set of useful cell references (i.e., to find the references of cells that will contain an initial input, a formula or a deferred input). Secondly, the logical specification is instantiated with these cell references in order to obtain the formulae defining the cell contents. And finally, the spreadsheet is implemented using these formulae, and a binding schema that maps cell references to physical squares in the grid.

3.2 Generation technique

The whole generation process is presented on the following example. Let the table *school* of example 2.1 be given and described by a set of ground atoms. We want to implement a spreadsheet to handle the marks of the students in group $gr2$. Suppose we must take into account that these students will

take one more course in computer science (*cs* for short), and that they can also take an optional course in chemistry. We want to produce the spreadsheet depicted in figure 4, to compute the final average of the students when the user will provide their marks in *cs* and chemistry.

$gr2$	$math$	$phys$	cs	$chemistry$	$finalAvg$
$mike$	14	12			
$alan$	15	16			

Figure 4: A view of the final spreadsheet

The final average $finalAvg$ is computed by applying a weight of 2 to the $math$ and $phys$ marks, and a weight of 1 to both cs and $chemistry$. If a student does not take the chemistry course, then the string nc (for *not chosen*) will be input instead of a mark, and his final average is computed by applying the weight of 2 to the $math$, $phys$ and cs marks. This spreadsheet is now defined with the query language. The construction of table $gr2$ using the table *school* of example 2.1 is specified by a single rule:

(R_1) $\quad gr2(N, M){:}X \longleftarrow school(N, M){:}X, school(N, group){:}gr2, M \neq group.$

The two following rules define the computation of the final average[1]:

(R_2) $\quad gr2(N, finalAvg){:}A \longleftarrow gr2(N, math){:}M1, gr2(N, phys){:}M2, gr2(N, cs){:}M3,$
$gr2(N, chemistry){:}M4, number(M4), A\ is\ (2 \times M1 + 2 \times M2 + M3 + M4)/6$

(R_3) $\quad gr2(N, finalAvg){:}A \longleftarrow gr2(N, math){:}M1, gr2(N, phys){:}M2, gr2(N, cs){:}M3,$
$gr2(N, chemistry){:}nc, A\ is\ (2 \times M1 + 2 \times M2 + 2 \times M3)/6$

The set of rules $\{R_1, R_2, R_3\}$ is called the *Spreadsheet Relation Specification* and is noted SRS. This specification is completed with the definition of the references of the cells that will contain a deferred input. In our example, these cells are those containing the marks that will be entered by the user of the spreadsheet (i.e., the marks in *cs* and *chemistry* in table $gr2$). This definition is made by the following rules using a particular symbol noted @ to fill the cells that contain a differed input:

(RD_1) $\quad gr2(N, cs){:}@ \longleftarrow school(N, group){:}gr2.$

(RD_2) $\quad gr2(N, chemistry){:}@ \longleftarrow school(N, group){:}gr2.$

Informally, the rule RD_1 (resp. RD_2) says that if it exists a cell $school(N, group)$ containing $gr2$, then it exists a cell $gr2(N, cs)$ (resp. $gr2(N, chemistry)$) whose contents will be supplied later by the user of the spreadsheet. The set of rules $\{RD_1, RD_2\}$ is called the *Differed Input Specification* and is noted DIS.

We now illustrate the three steps of the spreadsheet generation using SRS and DIS.

Step 1 This first generation step computes the references of the cells that will contain an initial input, a formula or a deferred input.

Starting from the set of facts corresponding to the initial input, the rules in $SRS \cup DIS$ are applied iteratively as if they were production rules to generate new facts. The generation terminates when no new fact can be obtained by any application of the rules. More precisely, this process is a naive

[1] In the first rule $number(M4)$ succeeds if $M4$ is bound to a number.

bottom-up query evaluation procedure that is well known in the database domain[1]. This form of evaluation for the kind of query language we use has been considered in [8]. This paper points out that simple syntactic restrictions can guarantee the termination of the procedure.

During the evaluation, the symbol @ is a place holder for values that are not known at that time. @ is handled as any other constant, except in the following cases:

1. If @ occurs in the right hand side of an *is* expression, then this right hand side is reduced to @. For example, X *is* $(2 \times 14 + 2 \times 12 + @ + @)/6$ will be transformed into X *is* @. Intuitively this expresses that if X depends on an unknown value then the value of X is also unknown.

2. If @ occurs in a *test* expression (e.g., $@ + 1 > 4$, or $number(@)$) then the test succeeds. This optimistic strategy is needed in order to compute *all* cell references that can be used in the final spreadsheet, whichever value will be given to @ when the user will execute the spreadsheet.

3. If the fact $T(R, C)$:@ (where T, R and C are constants) is produced then for any new rule application we consider that $T(R, C)$:α holds (i.e., is *true*) for any value α. For example, if $gr2(mike, chemistry)$:@ is produced then we consider that $gr2(mike, chemistry)$:nc also holds. As in the previous case, this strategy is needed to produce all possible final cell references, even though the value of @ is not yet known.

We illustrate the computation of the set of facts containing the references of the cells that will be filled with an initial input, a formula or a deferred input. Let $\mathcal{I}$ be the initial input. In the example, it corresponds to the table *school* given in figure 1 and we have:
$\mathcal{I} = \{school(kate, math)$:15, $school(kate, phys)$:12,$school(kate, group)$:$gr1$, $school(mike, math)$:14, $school(mike, phys)$:12, $school(mike, group)$:$gr2$, $school(john, math)$:10, $school(john, group)$:$gr1$, $school(alan, math)$:15, $school(alan, phys)$:16, $school(alan, group)$:$gr2$, $school(suzy, math)$:12, $school(suzy, phys)$:11, $school(suzy, group)$:$gr1\}$

A first application of all rules in $SRS \cup DIS$ on $\mathcal{I}$ leads to the set of new facts $\mathcal{I}_1 = \{gr2(mike, math)$:14, $gr2(mike, phys)$:12, $gr2(alan, math)$:15, $gr2(alan, phys)$:16, $gr2(mike, cs)$:@, $gr2(mike, chemistry)$:@, $gr2(alan, cs)$:@, $gr2(alan, chemistry)$:@$\}$.

Then a second application of the rules now on $\mathcal{I} \cup \mathcal{I}_1$ produces a superset of $\mathcal{I}_1$, the set $\mathcal{I}_2 = \{gr2(mike, math)$:14, $gr2(mike, phys)$:12, $gr2(alan, math)$:15, $gr2(alan, phys)$:16, $gr2(mike, cs)$:@, $gr2(mike, chemistry)$:@, $gr2(alan, cs)$:@, $gr2(alan, chemistry)$:@, $gr2(mike, finalAvg)$:@, $gr2(alan, finalAvg)$:@$\}$.

A third application of the rules on $\mathcal{I} \cup \mathcal{I}_1 \cup \mathcal{I}_2$ produces no new fact. It means that the set of facts $\mathcal{A}$ produced by the evaluation of the rules in $SRS \cup DIS$ on $\mathcal{I}$ is $\mathcal{A} = \mathcal{I}_2$.

It should be noticed that when the user operates on the final spreadsheet, the formulae produce only new cell contents and no new cell reference. Thus every cell reference has to be determined during the generation process. If the constant @ occurs within a cell reference (i.e., as the name of a table, a row or a column) in the set $\mathcal{A}$, it means that this cell reference is not completely determined, and that the generation must be aborted since no spreadsheet corresponding to the specification can be derived.

Step 2 This step produces for each cell the relation that defines its content. These relations are obtained by the specialization of the spreadsheet specification with respect to the references of the cells (and eventually their contents if available) that have been computed in Step 1.

All cells containing a deferred input have been associated with the same contents denoted @. In this step, their contents are still unknown, but they will be differentiated explicitly by associating a different constant noted $@_i$ with $i \in \mathbb{N}$ to each different cell reference. Let $\mathcal{A}'$ be the set of facts in $\mathcal{A}$ (generated in Step 1) transformed in this way.

$\mathcal{A}' = \{gr2(mike, math){:}14, gr2(mike, phys){:}12, gr2(alan, math){:}15, gr2(alan, phys){:}16,$
$gr2(mike, cs){:}@_1, gr2(mike, chemistry){:}@_2, gr2(alan, cs){:}@_3, gr2(alan, chemistry){:}@_4,$
$gr2(mike, finalAvg){:}@_5, gr2(alan, finalAvg){:}@_6\}$

The specialization of the specification is then made by instantiating the rules in SRS in all possible ways with the facts contained in $\mathcal{I} \cup \mathcal{A}'$.

On our running example, the instantiation of (R_1) gives the four following rules:

(R'_1) $gr2(mike, math){:}14 \longleftarrow school(mike, math){:}14, school(mike, group){:}gr2, math \neq group.$

(R'_2) $gr2(mike, phys){:}12 \longleftarrow school(mike, phys){:}12, school(mike, group){:}gr2, phys \neq group.$

(R'_3) $gr2(alan, math){:}15 \longleftarrow school(alan, math){:}15, school(alan, group){:}gr2, math \neq group.$

(R'_4) $gr2(alan, phys){:}16 \longleftarrow school(alan, phys){:}16, school(alan, group){:}gr2, phys \neq group.$

The rule (R_2) can be instantiated in two ways:

(R'_5) $gr2(mike, finalAvg){:}@_5 \longleftarrow gr2(mike, math){:}14, gr2(mike, phys){:}12, gr2(mike, cs){:}@_1,$
$gr2(mike, chemistry){:}@_2, number(@_2),$
$@_5\ is\ (2 \times 14 + 2 \times 12 + @_1 + @_2)/6.$

(R'_6) $gr2(alan, finalAvg){:}@_6 \longleftarrow gr2(alan, math){:}15, gr2(alan, phys){:}16, gr2(alan, cs){:}@_3,$
$gr2(alan, chemistry){:}@_4, number(@_4),$
$@_6\ is\ (2 \times 15 + 2 \times 16 + @_3 + @_4)/6.$

And finally, the rule (R_3) is instantiated as two rules:

(R'_7) $gr2(mike, finalAvg){:}@_5 \longleftarrow gr2(mike, math){:}14, gr2(mike, phys){:}12, gr2(mike, cs){:}@_1,$
$gr2(mike, chemistry){:}nc, @_5\ is\ (2 \times 14 + 2 \times 12 + 2 \times @_1)/6.$

(R'_8) $gr2(alan, finalAvg){:}@_6 \longleftarrow gr2(alan, math){:}15, gr2(alan, phys){:}16, gr2(alan, cs){:}@_3,$
$gr2(alan, chemistry){:}nc, @_6\ is\ (2 \times 15 + 2 \times 16 + 2 \times @_3)/6.$

Let SRS_{inst} be the set of rules obtained by the instantiation of the rules in SRS. SRS_{inst} is called the *instantiation of the spreadsheet relation specification.* In the example, $SRS_{inst} = \{R'_1, R'_2, R'_3, R'_4, R'_5, R'_6, R'_7, R'_8\}$.

The most interesting aspect of these rules is that they describe how to compute, from user-supplied values, the intentionally defined content of cells. This is used in the next step to generate the formulae to be placed in the grid of the spreadsheet. For example, the rules (R'_5) and (R'_7) define the contents of the cell reference $gr2(mike, finalAvg)$, and correspond to the following (not optimized) pseudo-code:

IF $gr2(mike, chemistry)$ contains a number $THEN$

$gr2(mike, finalAvg)$ contains

$((2 \times 14 + 2 \times 12 + valueInCell(gr2(mike, cs)) + valueInCell(gr2(mike, chemistry)))/6)$

$ELSE$

IF $gr2(mike, chemistry)$ contains the string "nc" $THEN$

$gr2(mike, finalAvg)$ contains $((2 \times 14 + 2 \times 12 + 2 \times valueInCell(gr2(mike, cs)))/6)$

$ELSE$

the contents of $gr2(mike, finalAvg)$ is not defined

$ENDIF$

$ENDIF$

Step 3 The grid at the physical level is now filled with the appropriated content. This is performed as follows:

1. Each cell reference is associated with a physical location, and the names of the rows (e.g., $mike$) and columns (e.g., $finalAvg$) are placed in the grid.
2. The squares in the grid that contain a deferred input are highlighted (e.g., with a particular border) to indicate that they must be filled by the user.
3. The cell contents computed in Step 1 are placed in the grid.
4. Finally, the formulae are generated and placed in the appropriated squares.

The first operation is performed using a default binding of the cell references in the grid (e.g., a binding based on a lexicographical ordering of the names of the rows and of the columns). For the second operation, the squares that will be used to enter the deferred input can be determined automatically, since they correspond to cell references associated with a content that is not known and not defined by a rule. These cells are the cells in $\mathcal{A}$ having a content equal to @, and such that their references do not appear in the head of a rule in SRS_{inst}. In our example, the references of these cells are: $\{gr2(mike, cs), gr2(mike, chemistery), gr2(alan, cs), gr2(alan, chemistry)\}$

Concerning the third operation, the set of cells containing definitively known constants corresponds to: $\{gr2(mike, math) : 14, gr2(mike, phys) : 12, gr2(alan, math) : 15, gr2(alan, phys) : 16\}$. Given the physical binding, the corresponding squares can be filled.

Finally, for the last operation, the set of squares containing a formula corresponds to the references of cells in $\mathcal{A}$ that are not associated with a deferred input or a known constant (determined during operations 2 and 3). In our example, the references of cells containing a formula are $gr2(mike, finalAvg)$ and $gr2(alan, finalAvg)$.

For each such cell reference, we consider the rules in SRS_{inst} defining its content (i.e., the set of rules in SRS_{inst} such that this cell reference appears in the rule head). As mentioned in Step 2, this set of rules corresponds to a pseudo-code that can be used to compute the cell content. At the physical level, the IF statements in the pseudo-code are encoded in formulae using the IF function available in most of the spreadsheet programs. This function uses three parameters: $IF(Exp1; Exp2; Exp3)$, where $Exp1$ is a boolean expression and where both $Exp2$ and $Exp3$ are unrestricted expressions

(e.g., a numeric or a string constant, another IF statement). From an operational point of view, when $Exp1$ evaluates to $true$ then $IF(Exp1; Exp2; Exp3)$ evaluates to $Exp2$, otherwise it evaluates to $Exp3$.

Consider a physical binding corresponding to the spreadsheet depicted in figure 4, where the cell reference $gr2(mike, cs)$ is associated in the grid to square $L2C4$ (line 2, column 4), while $gr2(mike, chemistry)$ and $gr2(mike, finalAvg)$ are associated respectively to squares $L2C5$ and $L2C6$. Then the rules defining the contents of $gr2(mike, finalAvg)$ (corresponding to the pseudo-code given in Step 2) are encoded in square $L2C6$ by the following formula (using Excel syntax [14]):

$= IF(ISNUM(L2C5);$
$\quad (2 \times 14 + 2 \times 12 + L2C4 + L2C5)/6;$
$\quad IF(L2C5 = "nc"; (2 \times 14 + 2 \times 12 + 2 \times L2C4)/6; "undefined"))$

where the function $ISNUM(X)$ evaluates to $true$ if X turns to be a number.

4 Conclusion

We presented a technique to generate automatically a spreadsheet from its specification written in a database query language. The key intuition of the generation process is that a spreadsheet is a query that has not been fully evaluated because some data were missing. The computation that can not be performed is delayed and encoded as formulae in the spreadsheet. The implementation of the technique in SWI-Prolog [19] to produce Excel [14] spreadsheets from their query-based definitions shows that the technique is effective. This prototype has been used to generate more complex spreadsheets than the one used as a running example in this paper and in particular it handles textual constraints to specify the ordering of the rows and columns in the grid. Interesting extensions to this work concern the use of a single query language to specify data manipulation within database management systems, data mining tools and spreadsheet programs (see [3] for a preliminary study).

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, Reading, MA, 1995.

[2] J.-F. Boulicaut, P. Marcel, F. Pinet, and C. Rigotti. Spreadsheet generation from rule-based specifications. In P. Fraternali, U. Geske, C. Ruiz, and D. Seipel, editors, *Proceedings of the 6th International Workshop on Deductive Database and Logic Programming (DDLP'98)*, volume 22 of *GMD-Studien*, pages 59–70, Manchester, UK, June 1998.

[3] J-F. Boulicaut, P. Marcel, and C. Rigotti. Knowledge discovery in multidimensional data. In *Proceedings of the ACM Second International Workshop on Data Warehousing and On-Line Analytical Processing DOLAP'99*, pages 87 – 93, Kansas City, Missouri, USA, November 1999. ACM Press.

[4] P. Brown and P. Gould. An experimental study of people creating spreadsheets. *ACM Trans. on Information Systems*, 5(3):258–272, July 1987.

[5] W. Chen, M. Kifer, and D.S. Warren. HiLog: a foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.

[6] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-Line Analytical Processing) to user-analysts: An IT mandate. White paper - http://www.hyperion.com/whitepapers.cfm, 1993.

[7] P. B. Cragg and M. King. Spreadsheet modeling abuse: an opportunity for OR? *Journal of the Operational Research Society*, 44(8):743–752, August 1993.

[8] M. S. Hacid, P. Marcel, and C. Rigotti. A rule-based CQL for 2-dimensional tables. In *Proceedings of the 2nd International Workshop on Constraint Database Systems (CDB'97)*, volume 1191 of *LNCS*, pages 92–104, Delphi, Greece, Jan. 1997. Springer-Verlag.

[9] M. S. Hacid, P. Marcel, and C. Rigotti. A rule-based data manipulation language for OLAP systems (short paper). In *Proc. of the 5th International Conference on Deductive and Object-oriented Databases DOOD'97*, volume 1341 of *LNCS*, pages 417–418, Montreux, Switzerland, December 1997. Springer-Verlag.

[10] J.-L. Hainaut. *Bases de données et modèles de calcul : outils et méthodes pour l'utilisateur*. InterEditions, Paris, 1994.

[11] T. Isakowitz, S. Schocken, and H. C. Lucas Jr. Towards a logical/physical theory of spreadsheet modeling. *ACM Trans. on Information Systems*, 13(1):1–37, Jan. 1995.

[12] ISO/IEC JTC1/SC21 N5739. *Database language SQL*, 1991.

[13] P. Marcel. Modeling and querying multidimensional databases. *Networking and Information System journal*, 2(5/6):515–548, 1999.

[14] Microsoft. *EXCEL V5.0 user's guide*, 1993.

[15] R. R. Panko and R. P. Halverson. Spreadsheets on trial: a survey of research on spreadsheet risks. In *Proceedings of the Twenty-Ninth Hawa International Conference on System Sciences*, Maui, Hawai, January 1996.

[16] G. Gottlob S. Ceri and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, March 1989.

[17] M. Stadelmann. A spreadsheet based on constraints. In *Proceedings of the 6th ACM symposium on User interface software and technology*, pages 217–224, Atlanta, GA, USA, Nov. 1993.

[18] S. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a generalization of association-rule mining. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, June 2-4, 1998*, pages 1–12. ACM Press, 1998.

[19] J. Wielemaker. *SWI-Prolog 2.9.6*. Dept. of Social Science Informatics (SWI). University of Amsterdam, Roeterstraat 15, 1018 WB Amsterdam, The Netherlands, 1997.