# TQL. A textual query language

François Bancilhon, Patrick Richard

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 145

# TQL
# A TEXTUAL QUERY LANGUAGE

François BANCILHON
Philippe RICHARD

Juillet 1982

# T Q L, A TEXTUAL QUERY LANGUAGE

François BANCILHON, Philippe RICHARD

ABSTRACT

We suggest a new approach to the design of data management systems that handle both textual and relational data. In this approach the user sees two distinct data bases that (s)he can declare and query. Text interpretation and Text generation mappings allow the user to view his (her) data as (s)he wishes.

We define a Text Definition Language (TDL) which is simply based on regular expressions and a Text Query Language (TQL) that is a calculus oriented Text language based on the notion of cursors. The language is both defined informally through simple examples and formally through painfull mathematics.

RESUME

Nous proposons une nouvelle approche pour la conception de systèmes de gestion de bases de données factuelles et textuelles. Dans ce cadre, l'utilisateur voit deux bases distinctes qu'il déclare et interroge indépendamment. Les liens entre les deux bases sont fournis par les fonctions d'interprétation de textes (en termes de relations) et de génération de texte. Nous définissons un langage de déclaration de textes basé sur les expressions régulières et un langage de requêtes de type calcul basé sur la notion de curseur. Nous l'exposons à travers de nombreux exemples puis nous le définissons formellement.

## 1. INTRODUCTION

Relational systems are now available as software products. Even though they do offer the best degree of data independance they are essentially meant for management type applications. However new types of applications are emerging that require large amounts of data to be stored, fetched and updated : Office Automation, Information systems, Computer Aided Teaching, Software engineering etc... These new systems differ in the type of data they want to store and manipulate : not only factual data but also pictorial, vocal and textual. In this paper we address as a first step the problem of managing in the same system textual and factual data.

To define such a system several approaches can be considered :

(i) a Relational Based Approach : Starting from a relational system , extend it to include some Textual features [HASK.82]. This would have the advantage of building short term prototypes. However such an approach could generate complex and patched systems.

(ii) The symetric approach:the Textual Based Approach could go the other way , from a text based system with the same pros and cons.

(iii) A third approach could try defining a new data model including facts and texts as elementary objects. One of the dangers of this method might be the generation of "yet another data model".

However, we believe these three approaches both have advantages and drawbacks and should be experimented. We suggest here a fourth method which consists in the co-existence of a relational data base and a textual data base.

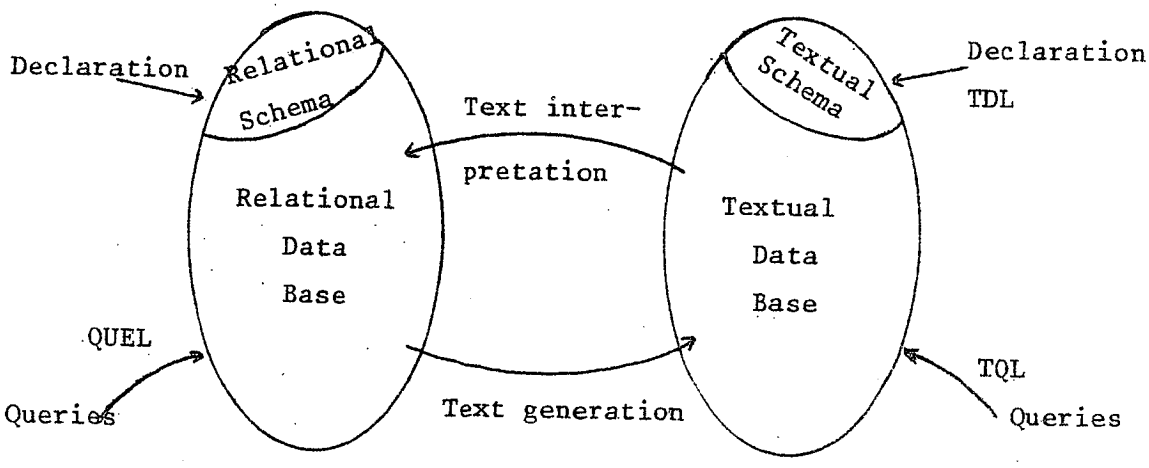



Fig 1. Co-existing Data Bases

As shown in Fig 1, two distinct data bases can be queried. A classical relational data base can be declared and interrogated through, say, QUEL while a "Textual Data Base" is declared by a Text Declaration Language (TDL) and queried by a Text Query Language (TQL). Two other arrows indicate the possibility of generating text from relations and of interpreting text by relations.

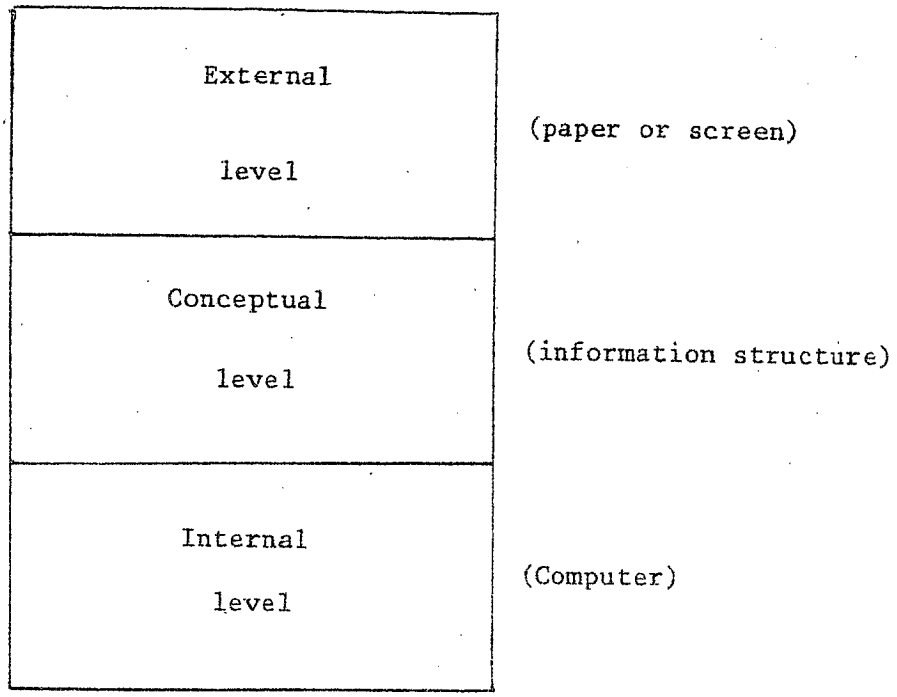
Languages to define generations and interpretations are necessary for the system to operate. No such language will be presented here as we shall concentrate on the definition of TDL and TQL as a first step to the co-existing systems approach.

Examples of text generations are : mailing letters from a set of addresses stored into a relation or preparing dictionaries from Base Information. Examples of text interpretation range from simple cases such as extracting a mailing list from a letter Log or constructing a dictionary, to complex operations such as text understanding. We do not suggest that all texts in the TDB will be interpreted by relations in the RDB nor that all relations in the RDB will generate texts in the TDB but merely that each user will have the possibility of having both a textual and factual view of each type of data.

The first step to the definition of the co-existing system is a sound definition of a textual data base.

## 2. THE THREE LEVELS OF TEXTUAL DATA BASES.

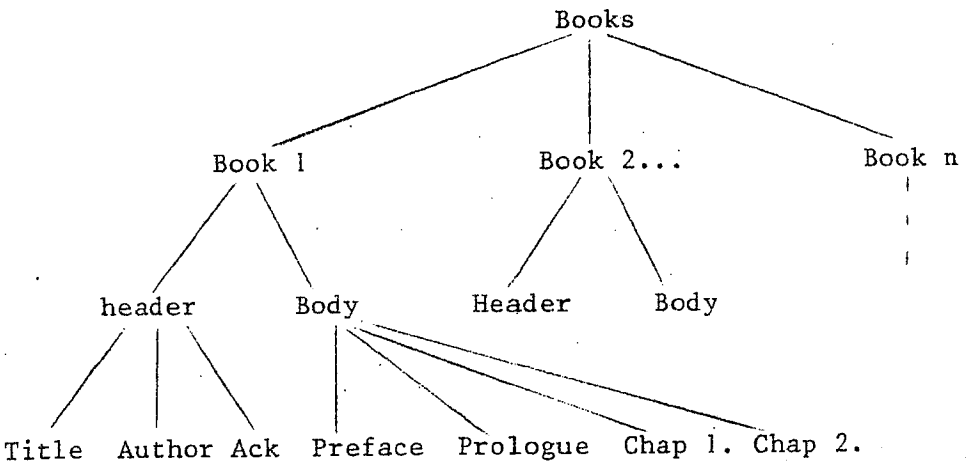
Two levels of description of textual data bases are fairly easy to define ; it is worth noting that both levels are physical : one represents the way text is being stored in a computer system (what code for characters, how is the text decomposed into blocks, how are the blocks chained etc...) the other one represents the way text is being stored on sheets of paper (or may be screens). At this level we say for instance that in a letter the address of sender is on the left handside, the address of sendee on the right handside, the date on the upright corner etc... At this level we worry about fitting sentences into lines and leaving blanks between paragraphs etc...




Both levels, because they worry about physical "implementation" problems contain unnecessary information. It seems therefore reasonnable to assume that between these two, a logical level can be defined which merely represents the information content and structure of the text. We shall call this level conceptual. Interpretation and generation mapping must be defined at the conceptual level. The conceptual level will be defined by a Text Definition Language (TDL) and a Text Query Language (TQL), these languages do not worry about the physical representation of the text (internal level) nor about the final manipulation of the text (text editor functions of the external level). We shall first define briefly TDL then move to an informal presentation of TQL which is a "calculus based" text manipulation language. A formal definition of the syntax and semantics of TQL is given in the appendix.

## 3. TDL : A TEXT DEFINITION LANGUAGE

We have choosen here the simplest view to structure a text : a hierarchical approach. Consider for example a text data base containing books. The text is therefore a sequence of books, each book consists of a header and a body. The header consists of the title, the author's name and the acknowledgement . The body consists of a preface, a prologue, and a sequence of chapters. Each chapter in turn is a set of paragraphs which are sets of words etc... There are many ways to represent such a hierarchical structure of text : grammars trees, expressions etc... For instance the syntax tree associated with this structure is :




In TDL we have choosen to describe such a structure by simplified grammars that generate regular languages. The TDL grammar associated with this tree is

Books = (Book)*

Book = Header . Body

Header = Title . Author. [Acknowledgement]

Body = [Preface].[Prologue] Chapters

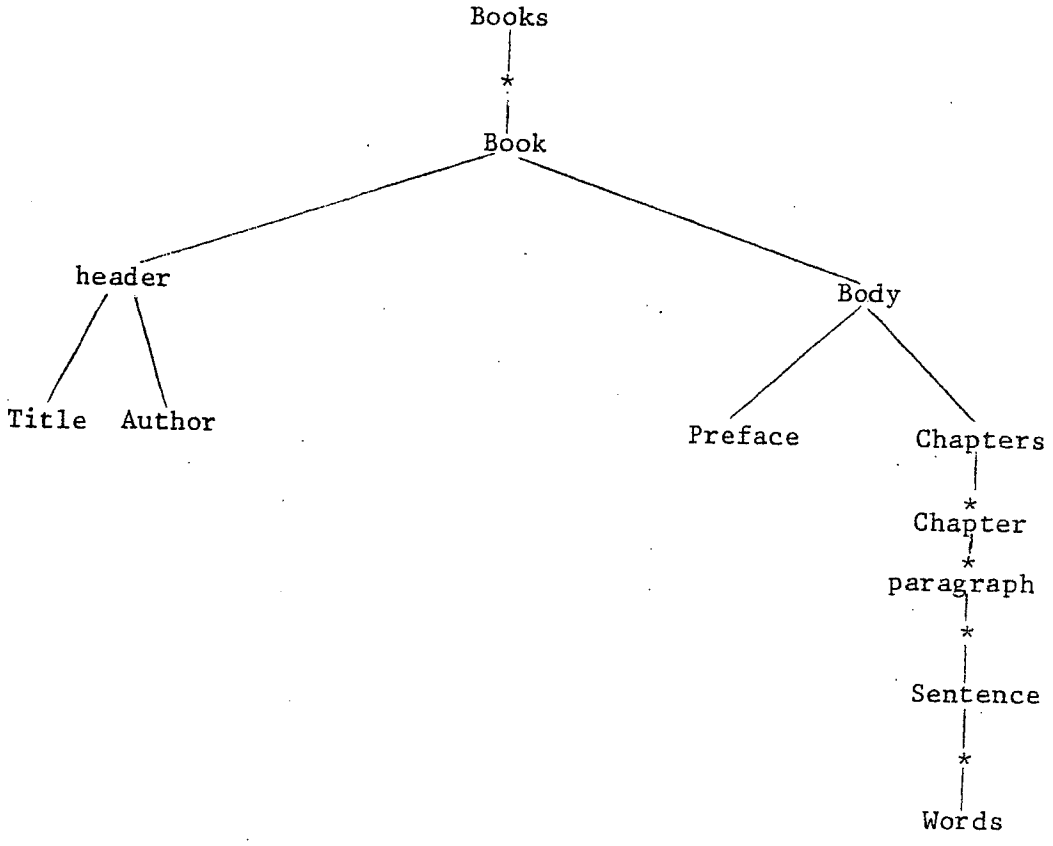
Chapters = (chapter)*

Chapter = (paragraph)*

Paragraph = (sentence)*

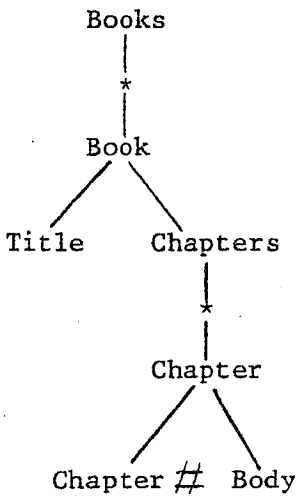
Sentence = (word)*

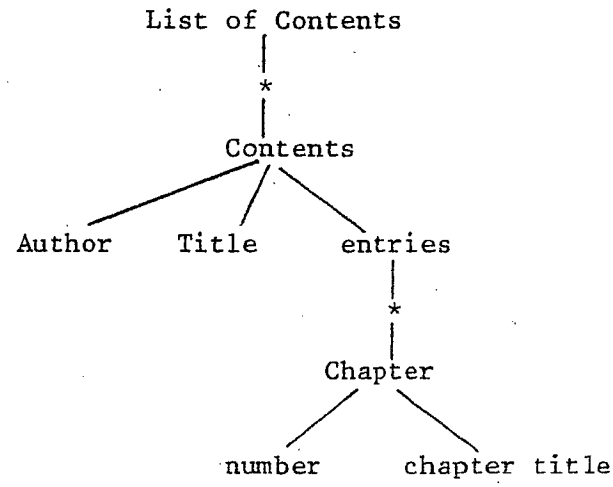
In this example one can easily see the rules that a TDL grammar follows : * is the iteration operator, [ ]denotes optional non terminals and no non terminal is recursive. Note that limiting ourselves to regular grammars is indeed a restriction but it seems to be a necessary assumption for any practical implementation. We shall sometimes prefer an abstract tree to a grammar. Such an abstract tree for the following example is :



## 4. AN INFORMAL PRESENTATION OF TQL

This description shall use the two following example text structures :




The basic notion in TQL is that of a cursor : a cursor is essentially a pointer that designates a structure in the text. For instance the query :

```
x cursor on Book in Books
select x. title
```

creates a cursor x on the non-terminal "Book" in the "Books" text. The select clause just asks to output the title part of that book. The result of that query is the sequence of titles of the Books text.

To print all the chapters of all the books we could use :

```
x cursor on Book in Books
select  x.chapter
```

These queries were mainly presented to introduce the use of cursors. Actual queries would have been :

```
select title
```

and `select chapters`

The in clause will be forgotten when no confusion is possible. Of course several elements can be printed by the same select clause

```
x cursor on Book
Select  x.title, x.chapter #
```

will print for all books its title followed by the list of chapter numbers. It is also possible to print constants within the text, thus :

```
x cursor on Book
select  'title', x.title
```

will insert the word "title" before each title. The next powerfull construct of TQL is the where clause that allows us to qualify cursors.
For instance

```
x cursor on Book
Select x
Where  x.title = 'Kidnapped'
```

will just print the entire book 'Kidnapped'

```
x cursor on Book
Select  x.title
where  x.chapter# = '15'
```

will print the title of all books having a chapter number 15. Note that "where x.chapter = '15'"is interpreted as "there exists a chapter # equal to 15". Of course more complex expressions can be constructed such as

```
x cursor on Book
Select  x.title
Where  x.chapter# = 15
and  x.title ⊃ 'man'
```

will give the title of all books having at least 15 chapters and having 'man' in their title. We shall not elaborate here on the necessary comparison operators.

A reasonable set might be :

$=, \neq, >, <, \geq, \leq, \supset, \subset$, near

Where near is a proximity operator defined by "equal up to a misspelling". At the level of detail of this paper it is only important to know that such a set could be defined.

When we want to compose one text from two texts it will be necessary to use two cursors. The first trivial example is that of concatenation :

x cursor on Books

y cursor on List-of-Contents

select x,y

will print the sequence of all books followed by the list of contents.

A more complex operation will insert the author's name into the book text :

x cursor on book

y cursor on contents

Select x.title, y.author, x.chapters

Where x.title = y.title

This operation is very similar to that of relational join. Note however that in the answer the set of books is ordered in the same way as in the book text. This is because in TQL we order terms lexicographically as in the original texts. Semi-join operations could also be defined by printing all books that are mentionned in the list of contents.

x cursor on Book

y cursor on Contents

Select x

Where x.title = y.title

The main difference in TQL comes when we want to do "nested" joins.

For instance, list for all books the sequence of chapters with their titles and numbers

```
x cursor on Book

y cursor on Contents

x' cursor on x.Chapter

y' cursor on y.Chapter

Select x'.chapter#, y'.chapter title, x'. body

Where x.title = y.title

and x'.chapter# = y'.number
```

In this query four cursors were created but they were "linked" by the fact that an occurence of cursor x' is generated by an occurence of cursor x and same for y and y'.

Finally the last feature of TQL is the ability to create cursors that represent queries. Consider for instance the query

```
Q(x,y) = x' cursor on x.chapter

         y' cursor on y.chapter

         Select x'.chapter# ,y'.chapter title, x'.Body

         Where x'.chapter# = y'.number
```

it is a query having x and y as "free cursor variables", we can define a new cursor say z and attach it to Q(x,y).
Thus giving :

```
x cursor on Book

y cursor on Contents

z cursor on Q(x,y)

Select x.title, y.author, z

Where x.title = y.title.
```

which represents the complete "join" of the two structures.

## 5. CONCLUSION

Even though the language might seem a bit complicated at first, experiments on a number of examples have shown it is a fairly powerfull and simple tool. It is now necessary to define some notion of completeness (weaker than recursive functions) to establish the power of such a language. Then implementation of such a system could be considered. This language was designed with the idea of being used in an environment where a hardware filter can process large amounts of data sequentially. We believe that most of the operations considered could be done in linear time.

[ADIB 82] . M.Adiba, C.Delobel, V.Joloboff. Private communication

[BRUA 81] . M.F.Bruandet
"Notion de concept pour la construction automatique d'un thesaurus evolutif," Congrés AFCET, Nov 81.

[CHRI 82] .C.Y.Chrisment, J.B.Crampes, M.Zina
"Les bases d'information généralisées" Projet BIG Lab. CERFIA. Université Paul Sabatier - Toulouse.

[HASK 82] R.L. Haskin, R.A. Lorie
"On Extending the Functions of a Relational Database System", Proc. of ACM-SIGMOD 82, June 2-4, Orlando, Florida.

[MIRA 82] .S.Miranda, M.Rothenburger. Private communication.

[RICH 82] .G.Richter
"IBM inscribed nets for modeling text processing and data (base) management systems".
VLDB 81. Cannes Sept. 9-11 France.

## Appendix 1. BNF description of TQL

Query = Text-definition

Text-definition = Declare-clause Selection-clause Condition-clause

Declare-clause = Declare-clause Cursor-declaration / Cursor-declaration.

Cursor-declaration = Variable-name cursor-on non-terminal [in Text-type]
| Variable-name cursor-on Variable-name.
non-terminal [in Text-type]
| Variable-name cursor on Text-definition.

Select-clause = Select Term-list

Term-list = Term-list . Term
| Term

Term = Cursor
| Cursor . Non-terminal
| Non-terminal
| 'text'

Condition-clause = Where Condition

Condition = Condition or condition
| Condition and condition
| not (Condition)
| (Condition)
| Elementary-condition

Elementary condition = Term Comparator Term

Comparator = = | > | ≥ | < | ≤ | ⊂ | ⊃ | near

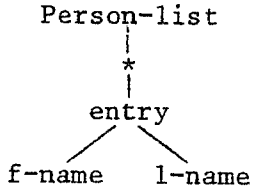
Besides the underlined keywords the terminals of this grammar are

| | |
|---|---|
| Variable-name | used to give name to variable |
| Non-terminal | defines a non-terminal of the text declaration |
| Text | sequence of characters |
| Text-type | The distinguished symbols of a text declaration |

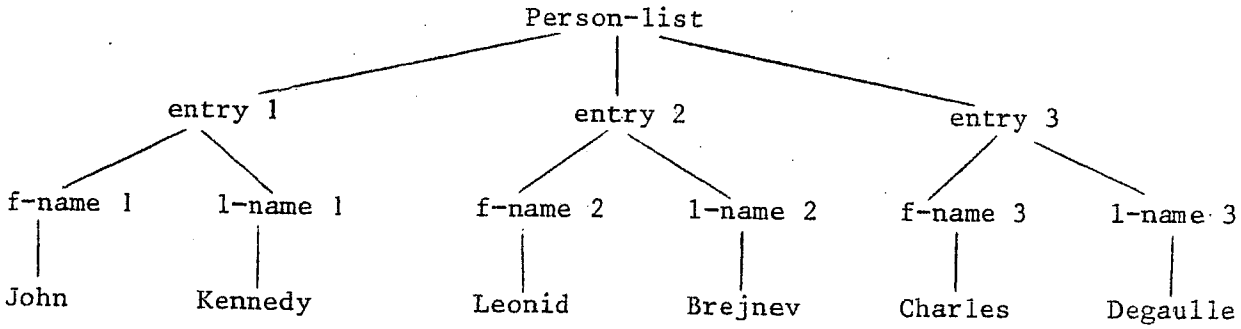
Appendix 2. Formal definition of the answer to a TQL Query :

Given a text (or a set of texts) and its structure, given a query on this (or these) text(s) the problem is to define formally the text that represents the answer to that query.

Recall that the text is structured by its grammar. Consider the following example description :




And the following text :
John Kennedy Leonid Brejnev Charles Degaulle.
Its associated syntax tree is :




Note that we have subscripted all nodes of the same type so that a unique name is given to each node of the syntax tree.

Recall that the general form of a query consists of three parts :

(1) cursor declarations

(2) selection of terms

(3) conditions.

Each cursor declaration is of the form

x cursor on A

or x cursor on $y^1$ . $y^2$. ... . $y^n$ . A

where x $y^1$ $y^2$ ... $y^n$ are cursor names and A is a non terminal.

or x cursor on Q

where Q is a query itself.

Each term of the select clause is of the form

x or x. A or A or 'text'

Where x is a cursor variable, A a non terminal and 'text' a constant text. Finally the condition is a boolean expression of elementary terms of the type

x . A $\Theta$ w

or x . A $\Theta$ y.B

Where x and y are cursor variables, $\Theta$ is a comparator, A and B are non terminals and w is a constant text.

Let us now give a set of definitions relative to a given text and its syntax tree

Let A be a non terminal : an occurence of A is simply a node of type A in the syntax tree. Occurences of A in the tree will be denoted $A_1$ $A_2$... $A_n$... Note that they form an ordered sequence.

For instance f-$name_2$ is an occurence of the non terminal f-name and the occurences of entry are {$entry_1$, $entry_2$, $entry_3$}.

Let now x be a cursor that has been declared by a statement of the type :

x cursor on $y^1$ . $y^2$ .... $y^n$ . A

where the sequence of $y^i$'s is possibly empty. Then an occurence of x is simply an occurence of A. Occurences of x will be denoted $x_1$ $x_2$....

For instance f-name 3 is an occurence of the cursor x defined by

x cursor on f-name

Let now $\left(x^1\ x^2 \ldots\ x^n\right)$ be a set of cursors of the query. A set of cursor occurences

$$\left(x^1_{i_1}\ \ x^2_{i_2}\ \ \ldots\ x^n_{i_n}\right)$$

is said to be valid if it satisfies both the cursor declaration and the condition clause.

Satisfaction of the cursor declaration part is defined as follows :

If x is declared by

x cursor on $y^1$ . $y^2$ . ... . $y^n$ . A

then $\left(x_i, y^1_{i_1}, y^2_{i_2}, \ldots\ y^n_{i_n}\right)$ is a valid set of occurences iff

$y^1_{i_1} \rightarrow^* \ldots\ y^2_{i_2} \ldots$

$y^2_{i_2} \rightarrow^* \ldots\ y^3_{i_3} \ldots$

$y^n_{i_n} \rightarrow^* \ldots\ x_i \ldots$

For instance if

x cursor on person-list

y cursor on x.entry

z cursor on x. y. l-name

then (person-list, $entry_2$, $l\text{-}name_2$) is a valid set of occurences of (xyz) while (person-list, $entry_1$, $l\text{-}name_3$) is not.

Satisfaction of the condition clause part is defined as follows :

Occurence $x_i$ of cursor n satisfies the elementary condition

x . A $\theta$ w

If there exists an occurence $A_i$ of A
such that :

$x_i \rightarrow^* \ldots A_i \ldots$

$A_i \rightarrow^* w'$

and $w' \ \theta \ w$

For instance $entry_2$ satisfies the condition

x cursor on entry

where x. f-name = Leonid

and Person-list satisfies the condition

x cursor on person-list

where x.F.name ≤ Xantipes

Finally, occurences $x_i$ and $y_i$ of x and y satisfy the elementary conditions

$x \ . \ A \ \theta \ y \ . \ B$

If $\exists$ occurences $A_1$ and $A_k$ such that

$x_i \rightarrow^* \ldots A_1 \ldots$

$y_j \rightarrow^* \ldots A_k \ldots$

$A_k \rightarrow^* w_1$

$A_k \rightarrow^* w_2$

and $w_1 \ \theta \ w_2$

For instance entry 1 and entry 2 satisfy the conditions

x cursor on entry

y cursor on entry

where x . f-name ≥ y . f-name

Let us now turn to the select part of the query. It consists of a set of terms depending on a set of cursor variables and of non terminals. Consider a set of occurences for these variables and non terminals and define the value of each term with respect to this set of occurences as follows :

-The value of term x is w where

$x_i \rightarrow^* w$

and $x_i$ is the occurence of x

-The value of term x . A is $w_1\ w_2 \ldots w_n$ where

$x_i \rightarrow^* \ldots A_1 \ldots A_2 \ldots A_n \ldots$

$A_j \rightarrow^* w_j \quad \forall j = 1, 2, \ldots, n$

and the $A_j$'s are all the occurences of A generated by $x_i$.

-The value of term 'w' is w

-The value of term A is w where

$A_i$ is the occurence of A

and $A_i \rightarrow^* w$.

We are now ready to define formally the answer to a query Q :

Let $x^1, x^2 \ldots x^n$ be the cursors of the query. Consider a set of valid cursor occurences $x^1_{i_1}\ x^2_{i_2} \ldots x^n_{i_n}$.

Let $t_1\ t_2 \ldots t_n$ be the terms of the select clause and let val $(t_i)$ be the associated value of term $t_i$. Define an element of the query as the concatenation :

val $(t_1)$ val$(t_2)$ ... val $(t_n)$.

Of course there are many elements in the query (as many as valid sets of occurences). The answer to the query is obtained by concatenation of all these elements in lexicographic order (remember that all occurences are ordered in their syntax tree).

This concludes the formal semantics of TQL . To be complete this definition would have to include the case of a cursor defined by a query, which can be defined recursively (note however that the cursor defining query might contain some free cursor variables, but the definition extends easily).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique