IMPLEMENTATIONS OF QT-SELECTORS IN RELIX

LINA JABBOUR

School of Computer Science
McGill University
Montreal, Quebec
May 1992

Acknowledgements

I would like to express my gratitude to Professor T.H. Merrett, my supervisor, for his guidance and support to do this project.

I would like also to thank Mr H. Shang for his help and continuous availability throughout my work.

OUTLINE

1. Introduction

the object of this project was to implement QT_selectors, a relational Algebra operation on **Relix** the new implementation of **Aldat**, the **Al**gebraic **da**ta language.

1.1 QT_selectors

Merrett, in 1977, introduced a new unary operator into the relational algebra, the QT-selectors.[2]

QT_selectors supply a quantifier query facility. The result of a QT_selector is a new relation which contains the tuples that satisfy the given quantifier expression.

A QT_selector is a facility used to count the number of different values of an attribute in a given relation or in a given subset of tuples in a relation. Hence, a QT_selector is a T_selector, a combination of the project and select operations, which has been extended to include quantifiers.

For example, find colours of cars owned by more than 60 persons in the following relation CARS (REG, NAME, SUBM, YEAR, COLOUR)

Answer ← [COLOUR] where { (# > 60) NAME } in CARS;

The *quantifiers* used in the QT-selectors are denoted by "#" and "." and symbolize the "number of" and the "proportion of" respectively. They allow the expression of such conditions as "more than 60 of" ("# > 60"), " at least 25% of (". >= .25 ") , "most of" (". > .5"), "an odd number of" ("# mod 2 = 1"). Similarly, the classical quantifiers "for some" and "for all" will be expressed as "# >=1" and " . = 1" respectively.

The "proportion of" quantifier suffers some ambiguities since it must be taken with respect to some total quantity. This total quantity could be taken to be the total number of different values in the domain corresponding to the attribute, or it could be the number of different values of the attribute in the given relation specified after **in** in the QT_selector. In this implementation it is the latter.

While QT_selectors have relations as values, *QT_predicates* and *QT_counters* have scalars as their results.

QT_predicate returns the percentage of attributes that satisfy the relation out of the total quantity . The total quantity has just been specified above.

For instance, **Answer2** ← .[COLOUR] where { (# > 60) REG } in CARS;

QT_counters returns the integer count of the attributes that satisfy the query

For instance, **Answer3** ← #[COLOUR] where { (# > 60) REG} in CARS;

1.2 QT-selectors Implementation in Relix

The QT_selectors are implemented on the new version of **Relix**, the

Relational database on Unix .Tsakalis in 1987 [3], already implemented the QT_selectors but in a previous version of Relix. In the older version, the whole relation being processed had to be in main memory. And that was one big disadvantage because it did not allow for the processing of large relations. But, this is not the case in the new version. Large relations are processed with no problem at all.
Other main differences between the two versions are the following :

- The new Relix has a multiuser environment. It has a concurrency control.
- It has a process concept. Several processes can run simultaneously
- It supports scoping.

This implementation of QT_selectors is based on Algorithm QT by Merrett[1]. Unlike other algorithms that use several passes because of projections and equivalence reductions, the QT algorithm is an implementation that uses a single pass.

1.3 Overview of Report

In the following, section two deals with the user manual. There the reader finds an overview of the QT_selectors syntax, and examples. Among the examples, query 3 is worked out in detail. The third section is about implementation. There is an actual copy of QT_selectors statements in the lexical and parsing files, as well as a pseudocode of the actual programming code. Moreover there is a copy of the QT Algorithm of which the pseudocode is an expansion.

[1] Merrett T. H., "Relational Information Systems", Reston 1984, pp. 255.

2. User Manual

2.1 Syntax

The syntax of the QT_selector in Relix :

```
< relational_expression >
        no_if_relational_expression


<no_if_relational_expression>
        '#' < project_list > < selection > IN < relational_expression >
    |
        '.' < project_list > < selection > IN < relational_expression >
    |
        < project_list > < selection > IN < relational_expression >


< project_list >
        empty
    |
        '['     <domain_id_option > ']'


< selection >
        WHERE '{' < quantifier_list > '}'
    |
        WHERE '{' < quantifier_list > '}' < domain_expression >


< quantifier_list>
        '( <domain_expression > ')' < domain_expression >
    |
        '(' < domain_expression > ')' < domain_expression >
        ',' <quantifier_list>


<domain_expression >
        '#' < constant_type_option >
    |
        '.' <constant_type_option >
    |
        < domain_expression > EQ < domain_expression >
    |
```

- no_if_relational_expression: is one of the two options of a relational expression. It is as the name says the relational expression without if_then_else in it.
- the domain_id_option: is the list of domains seperated by a comma.
- and domain_expression could be "#" or "." sign, an identifier or an expression of the form opnd1 op opnd2.

More details about the above is found in Tsakalis thesis[3] or in the file lgrammar.y

2.2 Quantifiers

The quantifier symbols "#", "." are of type integer and real respectively. A *quantifier expression* is a boolean expression involving any number of occurrences of the following elements:

- quantifier "#" , "."
- domains or constants of the same type as "#" or "."
- logical operators : **and** , **or** , **not**
- comparison operators : = , ≠, <=, >=, <, >

A *quantifier expression* always appears in parentheses before an attribute list in QT-selectors and QT-counters. It is a special case of domain expression which appears in the syntax of section **2.1**. For example, to express "more than two" departments we would write (# > 2) DEPT. A parenthesized quantifier expression is a *quantifier predicate* (e.g. "(#>2)") and an attribute preceeded by a quantifier is a *QT_domain* (e.g. DEPT). A quantified expression is a quantifier predicate followed by an attribute or an attribute list.[2]

2.3 Examples

The following are QT_selector examples. The third example, query 3, is worked out in detail.

Consider the following relation :

SUPPLY (	COMP	DEPT	ITEM	VOL)
	Domtex	Rug	Yarn	10
	Playsew	Rug	Yarn	17
	Playsew	Toy	Yarn	20
	Oddball	Toy	Yarn	13
	Domtex	Rug	String	5
	Domtex	Toy	String	2
	Playsew	Toy	String	10
	Playsew	Shoe	String	5
	Shoeco	Shoe	String	15
	Playsew	Toy	Ball	2
	Oddball	Toy	Ball	2

Query 1 : Find items supplied to more than two departments.

Syntax : QT1 ← [ITEM] **where** { (# > 2)DEPT } **in** SUPPLY;

Result :

relation QT1 (ITEM)
String

Query 2 : Find items except those supplied in volumes of less than 10.

Syntax : QT2 ← [ITEM] **where** { (# = 0) VOL } VOL < 10 **in** SUPPLY;

Result

relation QT3 (ITEM)
Yarn

NOTE : this example shows that conditions are evaluated from right to left if there are more than one. This also shows how to negate a condition (e.g., VOL < 10) by selecting values (e.g., ITEM) for which it is not true.

Query 3 : Find items supplied by at least two companies to more than one department.

Syntax : QT3 ← [ITEM] **where** {(# > = 2) COMP, (# > 1) DEPT} **in** SUPPLY.

The following are stages in evaluating query 3 :

sorted_file

Line	*CB*	(ITEM	COMP	DEPT)	Q_3 true	S_2	Q_2 (S_2 > 1)?	S_1	Q_1 (S_1 >= 2)?	S_0
1	3	Ball	Domtex	Rug	t	1\0	f	0		
2	1	Ball	Oddball	Rug	t	1				
3	2	Ball	oddball	Toy	t	2\0	t	1		
4	1	Ball	Playsew	Toy	t	1\0	f	1\0	f	o
5	0	String	Domtex	Rug	t	1				
6	2	String	Domtex	Toy	t	2\0	t	1		
7	1	String	Oddball	Toy	t	1				
8	2	String	Oddball	Shoe	t	2\0	t	2		
9	1	String	Shoeco	Shoe	t	1\0	f	2\0	t	1 ->**String**
10	0	Yarnl	Oddball	Toy	t	1\0	f	0		
11	1	Yarn	Playsew	Toy	t	1	f	0	f	0
12	-1									

Where ***CB*** stands for **C**ontrol **B**reak.
1/0 means at first the value was 1 and then it became 0.
Si i:0 .. 2 : Counts the number of different values of (ITEM, COMP,..) for which Q_{j+1} is true.
Qi i:1 .. 3 : is the boolean-valued function on one or both of "#" and ".". When evaluating Q_j, the symbol "#" is replaced by S_j. And the symbol "." by S_j/D_j. This is the meaning of $Q_j \leftarrow Q_{j+1}(S_{j+1}, S_{j+1}/D_{j+1})$

Explanation : To understand more this example follow the QT Algorithm, section 3.2, or the pseudocode, section 3.3.
line 1: The first tuple is assumed to be identical to its predecessor. The first CB is 3, because this is the first attribute at which the two identical tuples differ. ITEM is the attribute at position 0, COMP is at position 1 and DEPT at position 2.
k =2 as we have two QT_expressions in the query, (# > = 2)COMP and (# > 1) DEPT.
As the control Break > k then

$$Q_{k+1} \leftarrow (\ Q_{k+1}\ \textbf{or}\ T(A, B_1, .., B_k, C))$$

and there is no T_selection, hence Q_3 at line 1 is true.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
1	3	Yarn	Domtex	Rug	*t*	0	NA	0		0

line 2: control Break is 1 because COMP is the first attribute at which tuples 1 and 2 differ. Now Control Break is not > k so we take the
else option

$$S_k \leftarrow (S_k\ +\ \textbf{if}\ (Q_{k+1}\ \textbf{then}\ 1\ \textbf{else}\ 0))$$

So S_2 at line 1 gets 1 because Q_3 is true.
As Control Break is not equal to k (CB = 1 and k = 2) then

$$Q_{j+1} \leftarrow Q_{j+1}\ (S_{j+1}, S_{j+1}/D_{j+1})$$
$$S_j \leftarrow S_j + (\textbf{if}\ Q_{j+1}\ \textbf{then}\ 1\ \textbf{else}\ 0)$$

Q_2 gets false (S_2 is not greater than 1) and
S_1 stays 0 because Q_2 is false.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
1	3	Yarn	Domtex	Rug	t	*1*	*f*	*0*		0
2	1	Yarn	Playsew	Rug						

As it is not the last tuple

$$Q_{k+1} \leftarrow T(A, B_1, .., B_k, C) \quad \text{/*evaluate T if any */}$$
$$\textbf{for}\ j \leftarrow \max(0, \text{ControlBreak}) + 1\ \textbf{to}\ k\ \text{do}\ \ S_j \leftarrow 0$$

Q_3 gets true at line 2.
and S_2 and S_1 become zero.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
	3	Yarn	Domtex	Rug	t	1/*0*	f	*0*		0
2	1	Yarn	Playsew	Rug	*t*					

line 3: Now Control Break is 2.
S_2 <-+ 1 and becomes 1 at line 2; and Q_3 gets true at line 3.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
2	1	Yarn	Playsew	Rug	t	*1*				
3	2	Yarn	Playsew	Toy	*t*					

line 4: Control Break is 1. S_2 gets +1 and becomes 2,
Q_2 gets true (S_2 is greater than 1) and S_1 becomes 1 at line 3.
Q_3 gets true at line 4 and S_2 gets 0 at line 3.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
2	1	Yarn	Playsew	Rug	t	1				
3	2	Yarn	Playsew	Toy	t	*2/0*	*t*	*1*		
4	1	Yarn	Oddball	Toy	*t*					

line 5: Control Break is 0.
S_2 <- 1 at line 4 and
Q_2 gets false and S_1 stays 1 at line 4,
Q_1 gets false and so S_0 stays 0 at line 4.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
3	2	Yarn	Playsew	Toy	t	2\0	t	1		
4	1	Yarn	Oddball	Toy	t	*1*	*f*	*1*	*f*	*0*
5	0	String	Domtex	Rug						

Control Break is zero but as S_0 is also zero , the attribute " Yarn" is not added to the result.
Moreover, Q_3 gets true at line 5, and S_2 and S_1 gets 0 at line 4.

Line	*CB*	(ITEM	COMP	DEPT)	Q_3	S_2	Q_2	S_1	Q_1	S_0
3	2	Yarn	Playsew	Toy	t	2\0	t	1		
4	1	Yarn	Oddball	Toy	t	1*0*	f	1*0*	f	0
5	0	String	Domtex	Rug	*t*					

And this goes on until...

Result :

relation QT3 (ITEM)
String

Query 4 : Find items supplied to more than one department by at least two companies.

Syntax : QT4 <- [ITEM] where { (# > 1)DEPT, (# >= 2)COMP } in SUPPLY;

Result :

relation QT4 (ITEM)
String
Yarn

NOTE : Interchanging the quantifier_expressions in the last two examples gives different results.

Query 5 : Find items supplied by an even number of companies

Syntax : QT5 <- [ITEM] where {(# mod 2 = 0)COMP} in SUPPLY;

Result :

relation QT5(ITEM)
Ball

Query 6: Find items that are supplied to most of the departments

Syntax : QT6 <- [ITEM] where {(. > .5)DEPT} in SUPPLY;

Result:

relation QT6 (ITEM)
String
Yarn

Query 7 : Find items whose volume is greater than one and are supplied to exactly half the companies

Syntax : QT7 <- [ITEM] where { (. = 1/2)COMP } VOL > 1 in SUPPLY;

Result :

relation QT7 (ITEM)
Ball

Query 8: Find the number of items that are supplied to exactly two departments

Syntax: QT8 <-#[ITEM] where {(#=2)DEPT} in SUPPLY;

Result

relation QT8 (SHORT)
1

Query 9: Find the percentage of items whose volume is greater than one and are supplied to exactly half the companies.

Syntax: QT9 <- .[ITEM] where {(. = 1/2)COMP } VOL > 1 in SUPPLY;

Result

relation QT9 (REAL)
0.333333

3. Implementation

The object of this section is to present in detail the techniques and algorithms used to implement the QT_selectors..

The code is actually an implementation of the **QT Algorithm** by Merrett.[1]

3.1 Adding the QT Features to Relix

the following are QT_selector statements as they appear in files llex_ana.l, lgrammar.y, ltranslator.c and interpreter.c. And /* COMMENT */ stands for comments in the following files.

llex_ana.l

```
"#"                     { list_token;  return ("#"); }
"."                     { list_token;  return ("."); }
"{"                     { list_token;  return ("{"); }
"}"                     { list_token;  return ("}"); }
```

lgrammar.y

```
no_if_relational_expression :
            '#' < project_list > < selection > IN < relational_expression >
                { translator( QT_COUNT_NUM); }
      |
            '.' < project_list > < selection > IN < relational_expression >
                { translator( QT_COUNT_DOT); }

      |
            < project_list > < selection > IN < relational_expression >
                { translator( PROJECT); }


selection :
            WHERE '{' < quantifier_list > '}'
                { translator( QT_WHERE ); }
      |
            WHERE '{' < quantifier_list > '}' < domain_expression >
                { translator( QT_SELECT ); }
```

```
quantifier_list :
        '(  <domain_expression > ')'  < domain_expression >
                { translator( QT_DOMAIN );}

    |
        '(' < domain_expression > ')' < domain_expression >
                { translator( QT_DOMAIN );}
        ',' <quantifier_list>
    ;


domain_expression :
        '#'  { translator(NUM); } < const_type_option >
                {translator( SHORT); }
    |
        '.'  { translator(DOT); }   <const_type_option >
                { translator( SHORT); }
```

ltranslator.c

```
switch( code )
    .
    .
    case NUM :
    case DOT :
        if (code == NUM)
            FORMSHORT( scalar_template. name, NUM);
        else
            FORMSHORT( scalar_template. name, DOT);
        push_scalar_stack( NAME_ITEM, &scalar_template);
        push_short_stack( code, &select_stack);
        break;

    case QT_COUNT_NUM :
        fetch_RD_id();                  /* generate PUSH_NAME of
                                           pop_scalar_stack()->name */
        option = pop_short_stack( &select_stack);
        if (option == NULL_SELECT)          /* if no T_ (or QT_) selections */
            code = QT_COUNT_NUM;
        else if ((option == WHERE) || ( option == DOT) || (option == NUM))
            code = QT_COUNT_NUM_QT;

        push_scalar_stack(NAME_ITEM, &empty_template);
        list_generate( pop_list_stack(), /*NOT UPDATE*/ FALSE);
```

```
        operator_generate_code( code );
        break;

case QT_COUNT_DOT :
        fetch_RD_id();
        option = pop_short_stack( &select_stack);
        if (option == NULL_SELECT)          /* if no T_ (or QT_) selections */
                code = QT_COUNT_DOT
        else if ((option == WHERE) || ( option == DOT) || (option == NUM))
                code = QT_COUNT_DOT_QT;

        push_scalar_stack(NAME_ITEM, &empty_template);
        list_generate( pop_list_stack(), /*NOT UPDATE*/ FALSE);
        operator_generate_code( code );
        break;

case QT_WHERE :
        operator_generate( PUSH_COUNT);
        /* to get count= 2*m if QT_expressions are (QT1)B1, .. (QTm)Bm*/
        count_opnd_generate(2 * (1+pop_short_stack( &count_stack)));
        push_short_stack(0, &count_stack);
        reset_short_stack( &count_stack);
        break;

case QT_SELECT :
        operator_generate( PUSH_COUNT);         /* to get count= 2*m + 1
        if QT_expressions are (QT1)B1, .. (QTm)Bmand there is a T_selection */
        count_opnd_generate((2 * (1+pop_short_stack( &count_stack))) + 1);
        push_short_stack(0, &count_stack);
        reset_short_stack( &count_stack);
        break;

case QT_DOMAIN :
        fetch_RD_id();
        increment_top_short_stack( &count_stack);
        break;

case PROJECT :
        fetch_RD_id()
        option = pop_short_stack(&select_stack);
        if (option == WHERE)
                operator_generate(SELECT);
        else
                if ((option == NUM) || (option == DOT))
                        code = QT_WHERE;
        push_scalar_stack( NAME_ITEM, &empty_template);
```

```
        list_generate( pop_list_stack(), /*NO UPDATE*/ FALSE);
        operator_generate( code);
        break;
        .
        .
}
```

Intermediate Code

The intermediate code is a copy of the stack produced after the translator is called and is read by the interpreter by popping the top elements one after the other.

The translator procedure is called with the short argument "code". The translator is made of many cases. Each case depends on the value of "code" and either pushs, pops the tokens in stacks or print to the intermediate code file.
I would like to show the intermediate code for the following two examples:

```
answer1 ← [DEPT] where { (# > 2)COMP } in SUPPLY;
```

```
statements
push-name
answer1              /* relation name */
name
constant-relation
push-name
519                  /* short equivalent to "#" */
short                /* it is of type short */
push-name
-00002               /* 2 */
short
d->                  /* symbol for greater than domain */
push_name
COMP                 /* QT_domain */
push-count
2                    /* number 2 used in the interpreter to pop up both the */
push-name            /* quantifier predicate and the QT_domain */
SUPPLY
push-name
DEPT                 /* projection attribute */
push-count
1                    /* there is only one attribute */
qt-where             /* qt_where == it is a qt_selection with no T_selections */
assign-scalar        /* were there a T_selection we would have had qt_select */
halt
```

Another example,

```
answer2 <- [ITEM] where {(# >=2)COMP,(. > .01)DEPT} in SUPPLY;
```

```
statements
push-name
answer2
name
constant-relation
push-name
519                     /* short equivalent to '#" */
short
push-name
-00002                  /* 2 */
short
d->=                    /* greater or equal than domain */
push-name
COMP
push-name
437                     /* short equivalent to "." */
short
push-name
-+971000000             /* .01 in sortable ASCII code */
real
d->                     /* greater than domain */
push-name
DEPT                    /* the projection attribute */
push-count
4                       /* in the interpreter pop into the queue 4 elements */
push-name               /* 2 * (qt predicate and a qt_domain)  */
SUPPLY
push-name
ITEM
push-count
1
qt-where
assign-scalar
halt
```

linterpreter.c :

```
switch (code) {
        .
        .
        case QT_WHERE:
        case QT_COUNT_DOT_QT:
```

```
        case QT_COUNT_NUM_QT:
                pop_list_from_scalar_stack(FALSE, &list_R); < pop in list_Rlist of domains from scalar_stack>
                strcpy(RW_set[READ1], pop_scalar_stack()->name); < pop relation namein RW_set[READ1] >
                pop_list_from_scalar_stack(FALSE, &list_S);          < pop in list_S quantifier predicates
                (QTi) and QT_domains Bi and T_selctions if any. list_S <- QT1,B1,..QTm, Bm, (IT) >
                break;

        case QT_COUNT_DOT :             /* are the same as the PROJECT case */
        case QT_COUNT_NUM:

}

 switch (code) {
        .
        .
        case QT_WHERE:
        case QT_COUNT_DOT:
        case QT_COUNT_DOT_QT:
        case QT_COUNT_NUM:
        case QT_COUNT_NUM_QT:

                if (code == QT_COUNT_DOT_QT)
                        code  = QT_COUNT_DOT;
                else if (code == QT_COUNT_NUM_QT)
                        code  = QT_COUNT_NUM;
                strcpy(scalar_template.name, qt_where(list_R, list_S, RW_set[READ1]),code);
                break;
        .
        .
        }
```

example of each case:

QT_WHERE:	QT <- [ITEM] where {(# > 2)DEPT} in SUPPLY.
QT_COUNT_DOT:	QT <- .[ITEM] where DEPT = "Playsew" in SUPPLY.
QT_COUNT_NUM:	QT <- #[ITEM] where DEPT = "Playsew" in SUPPLY.
	/* with no QT_selection after where*/
QT_COUNT_DOT_QT:	QT <- .[ITEM] where {(# > 2)DEPT} in SUPPLY.
QT_COUNT_NUM_QT:	QT <- #[ITEM] where {(# > 2)DEPT} in SUPPLY

So the interpreter puts in list_R the projection domains, reads in RW_set[READ1] the relation name and copies to list_S the qt_predicates and the qt_domains $QT_1,B_1,..QT_m$, B_m plus the T_selection if any.
QT_i stands for quantifier predicate and B_i for its QT_Domain.
Then it calls the procedure qt_where with the three above variables as parameter plus code to tell whether the selection is a QT_selection a QT_count or a QT_num.

For the following example QT <- #[ITEM] where {(# > 2)DEPT, (# >1)COMP} in SUPPLY
list_R is (ITEM)
RW_set[READ1] is SUPPLY
list_R is the list ((#>2),DEPT, (#>1), COMP)

3.2 The QT Algorithm

Algorithm QT by Merrett[1] is a one-pass implementation of QT_expressions which are of either form:

QT-selectors	A where $(Q_1)B_1, \ldots . (Q_k)B_k$, T in R,
QT-predicates	.A where $(Q_1)B_1, \ldots . (Q_k)B_k$, T in R,
and QT-counters	#A where $(Q_1)B_1, \ldots . (Q_k)B_k$, T in R.

where A, $B_1, .. , B_k$ are attributes of a relation R(A, $B_1, .. , B_k$, C).
C is the attribute in the T_selection T. And, T is optional. When it does not exist T is considered to be true for all tuples.
Work is done on a sequential file sorted in lexicographical order on A, $B_1, .. , B_k$.
Control Break is j, the position of the first attribute B_j at which the current tuple differs from the preceeding tuple. attribute A is at position 0, B_1 at position 1 and so on so forth.

The first tuple is assumed identical to its predecessor and the last tuple is an imaginary successor to the actual final tuple, giving a control break of -1.

```
for each tuple
    if ( control Break > k) then
        Q(k+1) ← ( Q(k+1) or T(A, B1, .. , Bk , C))          /*evaluate T */
    else
        Sk ← (Sk + if (Q(k+1) then 1 else 0)

        if ( Control Break = k) then
            Q(k+1) ← T(A, B1, .. , Bk , C)                   /*evaluate T */
        else
            for each Bj, j ← k - 1 by -1 to max (0, ControlBreak)
                Q(j+1) ← Qj+1 (S(j+1), S(j+1)/D(j+1))        /* evaluate Q(j+1) */
                Sj ← Sj + (if Q(j+1) then 1 else 0)

            if ( Control Break <= 0 ) then
                if not last tuple and QT-selector then add tuple to result
                else if last tuple and QT-predicate then value is Q0
                else if last tuple and QT-counter then value is S0
```

if not last tuple **then**
$Q_{k+1} \leftarrow T(A, B_1, .. , B_k , C)$ /*evaluate T */
for j ← max(0, ControlBreak) + 1 **to** k do
$S_j \leftarrow 0$
end for each tuple

S_j : counts the number of different values of (A, B1, .. ,Bj-1) for which Qj+1 is true.
Q_j : is the boolean-valued function on one or both of "#" and "." defined on the query : $Q_j \leftarrow Q_j+1(S_{j+1}, S_{j+1}/D_{j+1})$, for j <= k.
When evaluating the symbol "#" in the quantifier-predicate Q_j is replaced by S_j, and the symbol "." is replaced by S_j/D_j
D_j : counts the number of different values of the attribute in the sorted_file relation.

3.3 The pseudocode

The following is the pseudocode of the actual implementation. The code is the procedure **qt_where**() in the file lalg_rel1.c. The code is for a relation of the form:
x A where $(Q_1)B_1$, ... $(Q_k)B_k$, T in R
where x is empty, "." or "#".
T is a T_selection involving an attribute C and T is optional.
R is a relation $R(A,B_1, ..,B_k,C)$.
Q_1 ,.. Q_k are QT_expressions

Four arguments are passed to qt_where() :
list_R contains the projection list attributes.(A)
r-name the relation name (R)
list_S is the selection list it has the form: (QT_1, B_1, .. , QT_k, B_k, T); T is optional.
qt_type has values QT_WHERE, QT_COUNT_DOT or QT_COUNT_NUM.
which means x is empty, "." or "#".

The code :

k ← count of elements in list_S /* number of quantifier predicates */

memory space is allocated for the arrays

Q[K+2] /* domains holding the boolean valued quantifier expressions */
s[k] /*short s_j counter. Holds the number of different values of $(A, B_1, .. ,B_{j-1})$ for which Q_{j+1} is true. */
q[k+2] /* q_j short boolean that holds value of actualization of Q_j on current tuple */
d[k+2] /* d_j integer that holds the number of different values of QT_domain j in case the quantifier symbol of expression j is "." */

```
qt[k+1]                 /* array of structure whose fields are QT , the quantifier predicate, and
                           B, the QT_domain  */
/* The sort_list (A, B1, .. Bk, C), where C is optional is created. And a sorted file, sort_file,
 is created. The structure qti with it fields QT and B are filled with the quantoifier expression
 and the  QT_domain respectively */

sort_list = list_copy ( list_R)
list_p == list_copy( list_S)
for each m, m ← 0 to m < k, by +1                    /*qt[1 .. k] */
        qt_m.QT = QT_m of list_p                          /* .QT = QT1, .. , QTk */
        qt_m.B  = B_m  of list_p->next                    /* .B  = B1, .. ,  Bk  */
        list = list_p ->next->next

sort_list = list_append ( (B1, . Bk) of list_S)
if T_selection, then
        sort_list = list_append (C )                 /* C is the attribute in the T-selection */
sort_file = sort( on  sort_list, r_name)             /* So, sort_list == (A, B1, .. Bk, C) , C optional */

if  T-selection
        Q_{k+1} ← T             /* the T_selection expression T*/
else
        Q_{k+1}← true.          /* otherwise Q_{k+1} is true no matter what the current tuple is */
        let domain "DOT" be (0,real)
        let domain "NUM" be (0,short)
for  each j , j ← k  to j > 0, by -1          /* for all the quantifier expressions in list_S */
        /* replace short NUM and short DOT in QT expressions by domains "NUM" and "DOT" */
        Find_Quantifier(qt_{j-1}.QT, "NUM", "DOT")
        Q_j ← qt_{j-1}.QT                                 /* Qj = QTj */


open sorted-file for reading.
open target-file for writing.
last-tuple = 0                                       /* it is not the last tuple */
relNtuples = the number of tuples in sorted file
read current tuple                          /* current tuple gets the first tuple of sorted_file */

for each tuple

        if first tuple then
                previous tuple = string_copy ( current tuple)
        else if ( ! read current tuple( sorted_file )) then   /* if there are no more tuples to read */
                if last-tuple = 0 then        /*if it is the last tuple then allow 1 more loop where last*/
                        last-tuple = 1        /*tuple is an imaginary successor to actual final tuple*/
                else
                        ntuples--
                        break                 /* get out of for loop */
```

```
        /* Get The Control Break */
if last tuple then
        controlBreak = -1                       /* special control Break value for last loop */
        current tuple = string_copy( previous tuple)
else    /* get the controlBreak */
        if current tuple <>  previous tuple  then
                controlBreak = 0
                while (attributes of current and previous tuples at the controlBreak level are
                different)
                        control Break ++
        else
                controlBreak = k + 1

if control Break > k  then
        if  q_{k+1} != 1    then
                qt_tuple_actualize ( Q_{k+1} on current tuple)  /* evaluate T_selection if  it exists*/
else
        if q_{k+1} = 1 then
                s_k ++
        if Control Break = k then
                qt_tuple_actualize(  Q_{k+1} on current tuple)
                q_{k+1} = actualized value of Q_{k+1}
        else
                for each j, j ← k-1, j >  max(0, controlBreak) by -1 do
                        let domain "NUM" be s_j
                        if  (d_{j+1} != 0) then
                                let domain "DOT" be (s_{j+1} /d_{j+1})
                                qt_tuple_actualize( Q_{j+1} on current tuple)
                                q_{j+1} = result of actualization of Q_{j+1}
                        if q_{j+1} = 1 then
                                s_j++

        if  (control Break <= 0) then
                if (ntuples != relNtuples + 1  & S_0 > 0 & qt_type = QT_WHERE) then
                        print attribute A of previous tuple, to target file
                else if (ntuples = relNtuples & qt_type = QT_COUNT_DOT) then
                        convert s_0/d_0  by real_to_ASCII_sortable
                        print it to target file
                 else if (ntuples = relNtuples & qt_type = QT_COUNT_NUM) then
                        convert s[0] by int_to_ASCII_sortable
                        print it to target file

        if (ntuples != relNtuples) then
                        qt_tuple_actualize ( Q_{k+1} on current tuple)
                        q_{k+1} = actualized value of Q_{k+1}
```

```
                for each j, j ← m+1, j <= k, by +1
                        s_j = 0

        i = 1 -i                /* swap last and current tuple by changing i */
end for each tuple
```

In implementation enough storage had to be available to contain the current relation. And if that was not possibe then the query could not be run.[3]
In this implementation we need not worry about memory storage at all.
Moreover, in this version of Relix, there is a multiuser environment so more than one user can be using the same relation at the same time. And this does not affect our implementation. This is all taken care off in the Relix code.

Functions Called

many functions are called by the function qt_where().
Many of them have been already available in Relix.
But qt_tuple_actualize() , qt_diff_values_of_attr() and qt_Find_Quantifier() are new.

qt_tuple_actualize() :

When Relix had to evaluate a boolean expression it was calling a procedure to actualize_current_pass() which in turn called the procedure tuple_actualize() to evaluate and actualize the expression for every tuple. But in our case we only need to evaluate the quantifier expression of one specific tuple at a time. Hence a new function qt_tuple_actualize() was introduced which has two arguments more than in tuple_actualize(). Those two arguments are the *current tuple* and the *attribute list* of the relation. Then qt_tuple_actualize calls tuple_actualize. The fact is that tuple_actualize is called only in the same file lactualize.c and the two variables the current tuple and the attribute list are private variables to the file.

qt_diff_values_of_attr() :

It takes as argument an attribute and a relation name. And returns the number of different occurrences of the attribute in the relation. It is used with the "." symbol. It is the denominator by which S_0 is divided to get the percentage.

qt_Find_Quantifier():
This is a tree searching algorithm that uses recursion.Its arguments are "ptr" a pointer to a domain expression tree structure and the name of two domains "num" and "dot". The thing is to go down the tree as far as possible, trying to go down operand1, but going operand2 if necessary. the basic is when the tree is of the form op1 op op2 where op1 or op2 is a leaf node. If that is the case, then check if the leaf is "." and if so, replace it with "dot", the domain parameter.
And if it is a "#" replace it with "num".

Summary

In summary, The report dealt with the implementation of QT_selectors on the new version of Relix.The user manual has worked out examples and explanation about QT_selectors and there syntax. The report gave details about implementation. the implementation part has an actual copy of statements found in four files for parsing and the pseudocode of the main code of procedure qt_where(). Hence, for any future maintenance enough details exist in this document.

The given examples in this report have been successfully tested. Other tests are being made. And the program is behaving very well.

References

1. Merrett T. H., "Relational Information Systems", Reston 1984.

2. Merrett T. H., "Relations as Programming Language Elements", Information Processing Letters, Vol.6,No.1,Feb. 1977, pp.29-33.

3. Tsakalis M., "Implementing QT_Selectors and Updates for a Primary Memory Version of Aldat", Master's thesis, McGill University, March 1987.

```
/****************************** QT_WHERE ***********************************/


/************************* From lalg_rel1.c file ************************/


/*---------
/* function:  qt_where: string
/*
/* arguments: list-R: pointer to LIST-TYPE; the projection list(A)
/*            r_name: string; Relation name.
/*            list_S: pointer to LIST_TYPE;selection-list(QT1,B1,..QTm,Bm,|T)
/*            qt_type:  QT_WHERE,
/*                      QT_COUNT_DOT,
/*                      QT_COUNT_NUM,
/*
/* return:    name of relation obtained when selecting from r_name
/*            the tuples satisfying the QT_expressions in list-S, projected
/*            on list_R.
/*
/* method:this is the implementation of the QT algorithm for QT_expressions:
/*      QT_slectors:  A where  (Q1)B1,... , (Qk)Bk, T in R,
/*      QT-predicate  .A where (Q1)B1, ... ,(Qk)Bk, T in R,
/*      & QT_counter  #A where (Q1)B1, ... ,(Qk)Bk, T in R.
/*
/*      sorted_file is sorted on [A, B1, .. , Bk, C] of R.
/*      ControlBreak is the position of the first attribute Bj at
/*      which the current tuple differs from the preceeding tuple.
/*
/*      Algorith QT : one-pass QT-expression evaluation
/*      (T.H. Merrett, Relational Information Systems p.255)
/*      Comment: attr A is at position 0.
/*      the first tuple is assumed identical with its predecessor
/*               the last tuple is an imaginary successor to the actul
/*               final tuple, giving a control Break of -1.
/*
/*      for each tuple
/*          if ( control Break > k) then
/*              Qk+1 <- ( Qk+1 or T(A,B1, .. ,Bk, C))
/*          else
/*              Sk <-+ (Sk if (Qk+1 then 1 else 0))
/*
/*              if (ControlBreak == k) then
/*                  Qk+1 <- T(A,B1, .., Bk, C)
/*              else
/*                  for each Bj,j<-k-1 by -1 to max(0,ControlBreak)
/*                      Qj+1 <- Qj+1(Sj+1, Sj+1/Denomj+1)
/*                      Sj <- Sj + (if Qj+1 then 1 else 0)
/*
/*                  if (ControlBReak <= 0) then
/*                      if (not last tuple) & QT_selector then
/*                              add tuple to result.
/*                      else if last tuple & QT-predicate then
/*                              Q0
/*                              else if last tuple & QT_counter
/*                                      then S0
/*
/*              if not last tuple then
/*                  Qk+1 <- T(A, B1, .. ,Bk, C)
/*                  for j <- max(0,ControlBreak) +1 to k
/*                      Sj <- 0
/*
/*      end of for each tuple
/*
/*
```

```
/*---------*/
char *
qt_where( list_R, list_S, r_name, qt_type)
LIST_TYPE       *list_R, *list_S;
char            *r_name;
int             qt_type;
{
  static
  char       fname[]= "qt_where: ", R_name[ MAX_ID],
             target_file[ MAX_ID], sorted_file[ MAX_ID];
  register
  char       *p, *r, *number;
  register
  short      controlBreak, m, nr_of_sort_dom, selector_dom,last_tuple;
  short      i, j, k, l, T_selection, nProjAttr; /* T_selection: boolean */
  short      *s, *q;
  int        *d;
  char       zero[MAX_ID],one[MAX_ID],bool1[MAX_ID],tmp[MAX_ID],tmp2[MAX_ID];
  char       dot[MAX_ID], num[MAX_ID], c[2], *tuples[2];
  char       spp[ SMALL_BUFFER_SIZE], spp1[ SMALL_BUFFER_SIZE],
               spp2[ SMALL_BUFFER_SIZE];
  long       ntuples, relNtuples, qt_tuples, result_dom;
  float      f1;

  /* qt[i].B and qt[i].QT are (.. QTi,Bi,...) of selection_list */
  struct qt_selec {
               char    B[MAX_ID];
               char    QT[MAX_ID];
               } *qt;
  struct astring {      /* allows a runtime creation of arrays of strings */
               char n[MAX_ID];
               } *Q, *S;


  FILE          *fp_in, *fp_out;
  LIST_TYPE     *list_p, *attr_list, *sort_list;
  REL_TYPE      *R;
  DOM_TYPE      *D, *ptr_D ;
  RD_TYPE       *RD;
  ACT_VAL_TYPE  act_val;
  boolean       at_least_one_tuple=FALSE;


   k = LT_count(list_S);
   if ((k % 2) == 1)            /*if there exists a T_selection */
        T_selection = 1;
   else
        T_selection = 0;

   k = k/2;                     /* k = the number of QT_selections. */
   nProjAttr = LT_count(list_R);

  /* actualize list-R and list_S in R, error checking and sort R */
  if(!vir_dom_LT_check( list_R)||!vir_dom_LT_check( list_S)||!*r_name)
    return( "");

  strcpy( R_name, r_name);
  if( !( R= search_rel_table( R_name)) ){
    error_gecho( geal_string( CANNOT_FIND), fname, R_name);
    return( "");
  }

  attr_list = rel_attr_list(R);
  if(!rel_ntuples( R) ) /*If relation is empty;If it is a temp */
        return("");

  strcpy( R_name, actualize if any virtual( list_R, R_name));
```

```
    if( !*R_name )
      return( "");

    if( !( R= search_rel_table( R_name))){
      error_gecho( geal_string( CANNOT_FIND), fname, R_name);
      return( "");
    }

    /* establish number of domains to sort on and
    /* create a name for the output relation;
    /*-*/
    nr_of_sort_dom= LT_count(list_R);
    create_Rtemp_name( target_file);
    if( !(fp_out= fopen( target_file, "w"))){
      error_gecho( geal_string( CANNOT_OPEN), fname, target_file);
      if( fclose( fp_in) == EOF)
        print_errno( fname, GEAL_FCLOSE);
      return( "");
    }
    qt_tuples = 0L;        /* number of tuple that satisfy QT_selection */

    if (nr_of_sort_dom)
  {
     /* allocate space for the arrays S[k], Q[k + 1],... */
     Q  = (struct astring *) calloc(k+2,sizeof(struct astring));
     d  = (int *)  calloc(k+1,sizeof(int));
     s  = (short *)  calloc(k+1,sizeof(short));
     q = (short *)  calloc(k+2,sizeof(short));
     qt = (struct qt_selec *) calloc(k+1,sizeof(struct qt_selec));

    /* sort on list (A,B1,.. Bk) */
    sort_list = LT_copy(list_R);
    list_p = LT_copy(list_S);
    /* append Bk ... B1 to sort_list  i.e. append k elts == 0 .. k-1*/
    m = 0;
    while(m < k) /* (list_p) */
    {
          strcpy(qt[m].QT,list_p->name);
          strcpy(qt[m].B, list_p->next->name);
          /* If it is not a valid virtual dom; */
          if( !(D= search_dom_table(qt[m++].B)) ){
                  error_gecho( geal_string( CANNOT_FIND), fname, qt[m].B);
                  return("");
          }
          list_p = list_p->next->next;
    }

    if (m > k)     /* if something went wrong */
          return ("");

    for(m=0; m < k; m++)
          LT_append_nocheck(qt[m].B, sort_list);

    if (T_selection)       /* get the attribute of the T-selection */
    {
          if(!(ptr_D = search_dom_table(list_p->name)))
                  return("");

          if (LT_member(dom_name(dom_opnd1(ptr_D)), attr_list))
                  strcpy(tmp ,dom_name(dom_opnd1(ptr_D)));
          else
                  if (LT_member(dom_name(dom_opnd2(ptr_D)), attr_list))
                          strcpy(tmp, dom_name(dom_opnd2(ptr_D)));
                  else
                          return("r_name");  /* PRINT ERROR MESSAGE  */
          /* append attribute C to sort_list */
```

```
      LT_insert(tmp, sort_list);
}

strcpy( R_name, actualize_if_any_virtual( sort_list, R_name));
if( !*R_name )
  return( "");

if( !( R= search_rel_table( R_name))){
  error_gecho( geal_string( CANNOT_FIND), fname, R_name);
  return( "");
}

strcpy( sorted_file, geal_sort(sort_list, R, /*ascending*/ TRUE));
if( !sorted_file )
      return( "");

/* set tuples[] to tuple-R and tuple-S;
/*-*/
tuples[ 0]= tuple_R;
tuples[ 1]= tuple_S;


 strcpy(zero, constant_dom(SHORT, "-00000\000"));
 strcpy(one, constant_dom(SHORT, "-00001\000"));

 /* Qk+1 = (Qk+1 or T(A,B1, .. ,Bk)) currTuple */


 if (T_selection)
      strcpy(bool1,list_p->name);
 else /* let Qk+1 be true */
      strcpy(bool1, constant_dom(BOOLEAN,/*TRUE*/ "1\000"));

 create_Dtemp_name(Q[k+1].n);
 let_dom_be(Q[k+1].n, bool1, tmp2, TRUE, LET);
 if (qt_type == QT_COUNT_DOT)
      d[0] = qt_diff_values_of_attr(list_R->name,r_name);

 create_Dtemp_name(dot);
 create_Dtemp_name(num);
 let_dom_be(dot, constant_dom(REAL, "-00001\000"),tmp2,FALSE,LET);
 let_dom_be(num, zero, tmp2, FALSE, LET);

 for(j = k; j > 0; j--)
 {
        strcpy(tmp, qt[j-1].QT);  /***** Qj <- Qj(Sj, Sj/#j)   */
        if(!(ptr_D = search_dom_table(tmp)))
              return("");

        /*denominator = red max of (fun + of 1 order attr)*/
        d[j] = qt_diff_values_of_attr(qt[j-1].B,r_name);

        /* get operand num and dot & change it to Sj or to Sj/# */
        qt_Find_Quantifier(ptr_D,num,dot);
        create_Dtemp_name(Q[j].n);
        let_dom_be(Q[j].n, tmp, tmp2, FALSE, LET);
        q[j] = 0;
        s[j] = 0;
 }/*FOR*/
 s[0] = 0;
 q[k+1] = 0;

 /*---
 /* open input and output files; input file contains the sorted relation*/
 if( !(fp_in= fopen( sorted_file, "r"))){
   error_gecho( geal_string( CANNOT_OPEN), fname, sorted_file);
```

```
    return( "");
}

i= 0;
last_tuple = 0;
tuples[ 1][ 0]= BYTE_0;
tuples[ 1][ 1]= BYTE_0;

/*read 1st tuple */
fgets( tuples[1 - i], MAX_TUPLE_LEN, fp_in);

/* tests on act_val */
act_val. v_buffer= get_free_buffer( &act_val. v_string);
act_val. v_bool = FALSE;

result_dom = LT_count(list_R); /* domains to appear in the result */
relNtuples = rel_ntuples(R);

/************************  THE LOOP ********************************/
/* while there are more tuples in the input relation
  read the next tuple into current tuple                */

for( ntuples= 0L; ; ++ntuples)
{
   if (ntuples == 0L) /* first tuple let it be a copy of tuples[1] */
      strcpy(tuples[i], tuples[1-i]);
   else
      if( !fgets( tuples[ i], MAX_TUPLE_LEN, fp_in) )
      {
           /* if last tuple set last_tuple to true */
           if (last_tuple == 0) {
                last_tuple = 1;
           } else {   /*beyond the last tuple then break */
                ntuples--;
                break;
                }
      }

  selector_dom = 1;
  extract_attribute( tuples[i], spp, selector_dom);
  if( !strcmp( spp,"1") )
    at_least_one_tuple=TRUE;
  /*---
  /* compare last tuple and current tuple and get the      */
  /* control Break level if it differs from the last tuple */
  if (last_tuple) /* in fact it if it is the imaginary tuple */
  {               /* after the last actual tuple */
      controlBreak = -1;
      /* make the last 2 tuples identical to the actual last tuple */
      strcpy(tuples[i],tuples[1-i]);
  } else
      if( !EQUAL( tuples[ 0], tuples[ 1]))
      {       /* get the control break which is <= k */
              controlBreak = 0;
              strcpy(spp1,"");
              strcpy(spp2,"");
              for (j=0;j < nProjAttr;j++)
              {
                  extract_attribute( tuples[i], spp, j);
                  strcat(spp1,spp);
                  extract_attribute( tuples[1 - i], spp, j);
                  strcat(spp2,spp);
              }
              j = nProjAttr -1;
              while (strcmp(spp1, spp2) == 0 )
              {
```

```
                controlBreak++;
                extract_attribute( tuples[i], spp1, controlBreak+ j);
                extract_attribute( tuples[1 - i], spp2, controlBreak+ j);
            }
    } else
            controlBreak = k + 1;

/************              IF CONTROL BREAK        ***********/

if (controlBreak > k)       /* >= or just > ??????????*/
{
      /* ACTUALIZE  Qk+1 = (Qk+1 or T(A,B1, .. ,Bk,C))  */
      if (q[k+1] != 1)
      {
           if (!T_selection)
                   q[k+1] =1;
           else {
               ptr_D = search_dom_table(Q[k+1].n);
               qt_tuple_actualize( ptr_D, &act_val, tuples[i],sort_list);
               q[k+1] = act_val.v_bool;
           }
      }
}
else     /* controlBreak <= k */
{
     /* actualize S[k] <-+ if Q[k+1] then 1 else 0 prevTuple */
     if (q[k+1])
          s[k]++; /* s[k] counting for arithmetic purposes */

     if (controlBreak == k) /* ?????????? */
     {
          if (!T_selection)
                  q[k+1] =1;
          else {  /*ACTUALIZE  Qk+1 =  T(A,B1,..,Bk,C) curTuple*/
               ptr_D = search_dom_table(Q[k+1].n);
               qt_tuple_actualize( ptr_D, &act_val, tuples[i],sort_list);
               q[k+1] = act_val.v_bool;
          }
     }
     else  /* if (controlBreak < k) */
     {
          if (controlBreak > 0)
              m = controlBreak;
          else
              m = 0;
          for(j=k-1;j >= m; j--)
          {
              if(!(ptr_D = search_dom_table(Q[j+1].n)))
                  return("");

              FORMSHORT(tmp,s[j+1]);
              strcpy(tmp, int_to_ASCII_sortable(tmp, DT_SHORT));
              strcpy(tmp, constant_dom(SHORT, tmp));
              let_dom_be(num, tmp, tmp2, FALSE, LET);
              if (d[j+1] != 0)
              {
                    f1 = ((float) (long) s[j+1])/((float) (long) d[j+1]);
                    FORMREAL(tmp, f1);
                    strcpy(tmp, real_to_ASCII_sortable(tmp));
                    strcpy(tmp, constant_dom(REAL, tmp));
                    strcpy(tmp2,dot);
                    let_dom_be(tmp2, tmp, tmp2, FALSE, LET);
              } else
                        ;/* ????? TRY THE DONT CARE */

                qt_tuple_actualize(ptr_D,&act_val,tuples[1-i],sort_list);
```

```
            q[j+1] = act_val.v_bool;

        /* Sj<-Sj+(if Qj+1 then 1else 0) */
        if (q[j+1])
            s[j]++;

    } /*for*/

    if (controlBreak <= 0)
    {
        /*if !last tuple & QT_selector add tuple to result */
        if (/*(ntuples != relNtuples + 1) && */ (s[0] > 0) &&
            (qt_type == QT_WHERE)) {
            /*find the first byte after the domains to sort
            on in the current tuple and put \n followed by \0 */
            if(!(p= skip_attributes(tuples[1- i], result_dom)))
                    /* continue*/ ;
            *p= '\n';
            *++p= BYTE_0;
            if( fprintf( fp_out, "%s", tuples[1-i]) CMP 0)
                print_errno( fname, GEAL_FPRINTF);
            qt_tuples++;
            s[0] = 0;
        }
        else if((ntuples == relNtuples)&&(qt_type==QT_COUNT_DOT))
        {
            number = (char *) calloc(15,sizeof(char));
            f1=((float)(long)(int)s[0])/((float)(long)d[0]);
            FORMREAL(number, f1);
            strcpy(number, real_to_ASCII_sortable(number));
            number[11] = '\006';
            number[12] = '\n';
            number[13] = BYTE_0;
            if( fprintf( fp_out, "%s", number) CMP 0)
                print_errno( fname, GEAL_FPRINTF);
            qt_tuples = 1;
        }
        else if((ntuples==relNtuples)&&(qt_type==QT_COUNT_NUM))
        {
            number = (char *) calloc(10,sizeof(char));
            FORMSHORT(number, s[0]);
            strcpy(number,int_to_ASCII_sortable(number, DT_SHORT));
            number[6] = '\006';
            number[7] = '\n';
            number[8] = BYTE_0;
            if (fprintf(fp_out, "%s", number) CMP 0)
                    print_errno( fname, GEAL_FPRINTF);
            qt_tuples = 1;
        }
    }/*IF*/

    /*if not last tuple then evaluate Q[k+1] & free memory of S[]*/
    if (ntuples != (relNtuples))
    {
        /* Qk+1 <- T(A, B1, .. Bk, C)       */
        /* ACTUALIZE        Qk+1 currentTuple */
        if (!T_selection)
            q[k+1] =1;
        else {
            ptr_D = search_dom_table(Q[k+1].n);
            qt_tuple_actualize(ptr_D,&act_val,tuples[i],sort_list);
            q[k+1] = act_val. v_bool;
        }
        for(j = m+1; j <= k; j++)
            s[j] = 0;
    }
```

```
        }/*else*/
      } /*else */
      i = 1 - i;      /* swap last tuple and current tuple by changing i */
  } /*FOR*/

  LT_free(sort_list);
  /*---
  /* close the files
  /*-*/
  if( fclose( fp_in) == EOF)
    print_errno( fname, GEAL_FCLOSE);
}
  if( fclose( fp_out) == EOF)
    print_errno( fname, GEAL_FCLOSE);

  /*---
  /* create entries for the output relation in the system tables;
  /*-*/
  if( !(R= search_rel_table( target_file)) )
    R= insert_rel_table( target_file);

 if (qt_type == QT_COUNT_DOT) {
       strcpy(list_R->name, ".real");
       /*LT_free(list_R->next);*/
  }
  else if (qt_type == QT_COUNT_NUM)  {
       strcpy(list_R->name, ".short");
       /*LT_free(list_R->next);*/
  }

  change_ntuples( R, qt_tuples);
  change_attr_list( R, LT_copy(list_R));

  if( !qt_tuples ){
    change_sort_status( R, UNKNOWN);
    change_rank( R, 0);
  }
  else{
    change_sort_status( R, SORTED);
    change_rank( R, nr_of_sort_dom);
  }


  p = rel_name( R);
  for( list_p= list_R, j= 0; list_p; list_p= list_p-> next, j++) {
    D= search_dom_table( list_p-> name);
    if( !(RD= search_rd_table( p, r= dom_name( D))))
        RD= insert_rd_table( p, r);

    change_pos( RD, j);
    change_z_order( RD, NO_OPERAND);
  }
  if (nr_of_sort_dom)
  if( !EQUAL( sorted_file, R_name) )
    if( unlink( sorted_file) == -1)
      print_errno( fname, GEAL_UNLINK);
   /* delete temporary domain expression names */
   del_temp_dom_expr(num);
   del_temp_dom_expr(dot);
   for(i=1;i <k; i++)
      del_temp_dom_expr(Q[i].n);

   del_temp_dom_expr(Q[k+1].n);

  /* return the name of the output relation
  /*- */
```

```
    if (list_p)
          LT_free(list_p);

     /* LT_free(attr_list);  THE PROBLEM */

    return( target_file);

}/*end of qt_where */


/*---------
/* function:  qt_diff_values_of_attr: string
/*
/* arguments: attr:     string;
/*            r_name    string;
/*
/* return:    integer which is the number of different values of attribute /*
/*            attr in the relation r_name.
/*
/*---------*/
int
qt_diff_values_of_attr(attr,r_name)
char    attr[MAX_ID];
char    *r_name;
{
  LIST_TYPE     *ptr_L;
  char          fname[]= "qt_diff_values_of_attr: ", Denom[MAX_ID],
                countD[MAX_ID], tmp[MAX_ID], tmp2[MAX_ID],
                redMax[MAX_ID], one[MAX_ID];
  int           d;
  FILE          *fp_Den;
  DOM_TYPE      *D;

        ptr_L = LT_get();
        strcpy(ptr_L->name, attr);
        ptr_L->ordn = 1;

        strcpy(one, constant_dom(SHORT, "-00001\000"));

        strcpy(redMax, vertical_op_of_dom(RED_MAX,
                vertical_op_of_dom(FUN_PLUS,one, (LIST_TYPE *) 0 ,ptr_L),
                (LIST_TYPE *) 0,(LIST_TYPE *) 0));

        create_Dtemp_name(Denom);
        let_dom_be(Denom, redMax, tmp2, FALSE, LET);
        D = search_dom_table(Denom);
        strcpy(ptr_L->name, Denom);
        strcpy(countD, project(ptr_L, r_name));
        if( !(fp_Den = fopen(countD, "r"))){
                error_gecho(geal_string(CANNOT_OPEN),
                fname, countD);
                return(0);
        }
        fgets(tmp, MAX_ID, fp_Den);
        if (tmp[0] == '-')
                tmp[0] = '+';

        fclose(fp_Den);
        d = (int) atoi(tmp);
        /* del_temp_dom_expr(Denom);*/
        LT_free(&ptr_L);
        return(d);
}/*qt_diff_values_of_attr()*/


/*-----------
```

```
/* function:    qt_Find_Quantifier
/*
/*arguments:    ptr: pointer to domain type
/*              n and d: strings
/*
/*action:       calls qt_Find_Quantifier recursively, while there are
/*              still inner nodes in the ptr domain expression tree.
/*              Basis is when we reach a basic expression in the form
/*              op1 op op2 where at least one of op1 or op2 is a leaf
/*              if op1 or op2 is NUM (i.e. "#") replace it by domain name
/*              n, an argument. if either is a DOT (".") replace it by
/*              domain d, also an argument.
/*
/*-------------------*/
int
qt_Find_Quantifier(ptr,num,dot)
DOM_TYPE        *ptr;
char    num[MAX_ID], dot[MAX_ID];
{
        /* if it is an inside node recurse */
   if (ptr->opnd1)
        if ((ptr->opnd1->opnd1) || (ptr->opnd1->opnd2))
                qt_Find_Quantifier(ptr->opnd1,num,dot);

    if (ptr->opnd2)
        if ((ptr->opnd2->opnd1) || (ptr->opnd2->opnd2))
                qt_Find_Quantifier(ptr->opnd2,num,dot);


        /* if it is a leaf check if op1 or op2 is either NUM
        or DOT and replace it by either n or d */
        if (ptr->opnd1)
            if (dom_constant(dom_opnd1(ptr)))
                if (atoi(dom_constant(dom_opnd1(ptr))) == NUM)
                        change_opnd1(ptr, search_dom_table(num));
                else if (atoi(dom_constant(dom_opnd1(ptr))) == DOT )
                        change_opnd1(ptr, search_dom_table(dot));

        if (ptr->opnd2)
            if (dom_constant(dom_opnd2(ptr)))
                if (atoi(dom_constant(dom_opnd2(ptr))) == NUM )
                        change_opnd2(ptr, search_dom_table(num));

                else if (atoi(dom_constant(dom_opnd2(ptr))) == DOT)
                        change_opnd2(ptr, search_dom_table(dot));
}/* qt_Find_Quantifier() */



/********************** From lactual.c file  ******************************/

/*---------
/* function:  qt_tuple_actualize
/*
/* arguments:   D:        ptr to dom-cell;
/*              act-val:  ptr to ACT-VAL-TYPE;
/*              tuple :   current tuple ptr to char;
/*              ptr_L :   the global attribute list of current relation.
/*                        prt to list type
/*
/* action:   initializes the private global variables: current_tuple
/*           and glob_attr_S and calls tupl_actualize().
/*
/***********/

void
```

```
qt_tuple_actualize ( D, act_val, tuple, ptr_L)
register
DOM_TYPE        *D;
ACT_VAL_TYPE    *act_val;
char            *tuple;
LIST_TYPE       *ptr_L;
{
        current_tuple = tuple;
        glob_attr_S = ptr_L;
        tuple_actualize( D, act_val);

} /* end of qt_tuple_actualize */
```