ESPRIT III

ORES: TOWARDS THE FIRST GENERATION OF TEMPORAL DBMS (P7224)

DELIVERABLE C4

IMPLEMENTATION OF VALID TIME RELATIONAL ALGEBRA

Responsible: 01 PLIROFORIKI
Information Dynamics
Contributor: Agricultural University of Athens
University of Athens

Availability: Restricted

Athens, October 1993

IMPLEMENTATION OF VALID TIME RELATIONAL ALGEBRA (DELIVERABLE C4)

Responsible : 01 PLIROFORIKI
Information Dynamics
Contributor : Agricultural University of Athens
University of Athens

ABSTRACT

We report on the development of Valid Time Relational Algebra. The development has been based on a previous ORES deliverable, concerning the Specification of Valid Time Formalism.

TABLE OF CONTENTS

1. INTRODUCTION

This deliverable concerns the development of the operations of Valid Time Relational Algebra (VT-RA). A major characteristic of the development is the support of a new primitive data type, *interval*. An interval has format $[d_i, d_j)$, where d_i and d_j are valid dates (*ORES dates*), satisfying $d_i < d_j$. The development of VT-RA is in accordance with the previous ORES deliverable, *Specification of Valid Time Formalism* [01P 93a] with the following improvements:

1. In [01P 93a] an operation, **Compute**, had been defined which incorporated certain special ORES functions, not supported, in their majority, by any commercial DBMS. It was reported that neither **Compute** nor these functions would be included in the implementation of VT-RA. In fact, these functions have been implemented and are incorporated in the **Select** and **Project** operations of VT-RA. All the INGRES functions are also supported. Finally, all the ORES and INGRES functions can be combined, to form composite functions.
2. Although optimisation algorithms were not to be incorporated in the development of VT-RA, they are fully used in operations **Unfold**, **Fold**, **Normalise**, **Punion** and **Pexcept**.
3. Although no operation for the renaming of attributes was to be implemented, such a renaming is allowed in operation **Project**.
4. Four more operations have been defined, **Help**, **Display, Drop**, and **Quit** to simplify the user's work.
5. The functionality of the VT-RA operations is fully compatible with the functionality of standard SQL.

As it should be expected, the specifications of all the VT-RA operations have been extended in the present deliverable, so as to include the above enhanced features.

It should be noted that operation **Reformat** has not been implemented. This does not cause any practical problem because **Reformat** is composite and can be expressed by sequences of **Unfold** and **Fold** operations.

The remainder of this deliverable is summarised as follows: Section 2 contains guidelines for the installation of the necessary software. Section 3 is a guide to the user. Implementation issues are addressed in section 4. Conclusions are drawn in the last section. Finally, Appendix A contains the evaluation tests of the implementation.

2. INSTALLATION

The software installation consists of two parts, the installation of the INGRES kernel and the installation of VT-RA. They are described separately next.

Installation of the INGRES kernel

1. Log in as an INGRES user.
2. Issue the command *iishutdown*, to shut down the INGRES servers.
3. Insert the INGRES kernel tape into the tape drive.
4. Enter the commands

   cd demo
   mv udadts udadts.bak
   tar xvf *device-name*
   cd udadts
   iilink

   where *device-name* is the special file associated with the tape drive (e.g. /dev/rst0). When prompted by the *iilink* command for the object files, reply

   $II_SYSTEM/ingres/demo/udadts/*.o

   Later, you will be prompted for an extension for the new binaries. Simply, press RETURN. The new INGRES kernel will be built, and various diagnostic messages will be displayed.
5. Issue the command *iistartup*, to start up the INGRES servers. The new kernel will support the INTERVAL data type, along with all the new functions defined for this data type.
6. Log out.

Installation of the VT-RA query processor

1. Log in as an authorised INGRES user (if you are not sure that such a user exists, log in as an INGRES user).
2. Insert the VT-RA query processor tape into the tape drive.
3. Extract the VT-RA query processor from the tape, by issuing the command

   tar xvf *device-name*

   where *device-name* is the special file associated with the tape drive (e.g. /dev/rst0).
4. Enter the command

chmod+x vtal

to make sure that you have execute permission for the VT-RA query processor. The VT-RA query processor is then installed.

In order to get ORES intervals to work together with the INGRES date data type, the INGRES environment DATE must be set to SWEDEN. This can be accomplished from the INGRES account, with the command:

ingsetenv II_DATE_FORMAT Sweden.

3. USER'S GUIDE

In this section guidelines are provided to the user for the use of Valid Time Relational Algebra (VT-RA). They are in accordance with the underlying formalism given in [01P 93a] and [01P 93b], except that the implementation incorporates additional improvements, discussed separately wherever necessary.

3.1 Invocation of VT-RA

VT-RA is invoked from the environment of the Operating System by the command
vtal <database_name>

Example:
vtal **ores**

3.2 Data Types

All the data types supported by INGRES are also supported by VT-AL. Their description can be found in [INGRES 89]. We restrict here to the definition of data types developed within ORES.

In the following we use the following terms:

Internal Representation: The format in which a constant of a particular data type is physically stored in the database.

Input Format: The format in which a constant of a particular data type has to be provided as an input argument to a function or as a piece of data which must be physically recorded in the database.

Display Format: The format in which a constant of a particular data type is displayed.

ORES Date

An ORES date type represents a valid date in format SWEDEN (YYYY-MM-DD). The lowest value is 1970-01-01 and the greatest one is 2030-12-30.

Internal Representation: CHAR(10)

Input Format: CHAR(10)

Display Format: CHAR(10)

ORES Interval

For simplicity reasons, an ORES interval is also referenced as interval. An interval has format $[d_i, d_j)$, where d_i, d_j are two valid dates (the start and stop of the interval) satisfying 1970-01-01 $\leq d_i < d_j <$ 2030-12-30.

Internal Representation:
Internally, an interval is stored as two distinct long integers (32 bits each), which represent its start and stop. The offset of each integer represents a date in the above range, expressed in seconds.

Input Format: It is a CHAR(23) string with format
[YYYY-MM-DD,YYYY-MM-DD).

Display Format: It is a CHAR(26) string with format
[^YYYY-MM-DD,^YYYY-MM-DD^)
where "^" denotes the space character.

3.3 Functions

In VT-RA it is possible to incorporate both the VT-SQL [01P 93b] and INGRES functions. The formal definition of the former can be found in [01P 93b], Appendix A. Here they are described in alphabetic order. The INGRES function *c* is also described here, due to its relevance to VT-AL. The computation of composite functions is allowed, too.

Format

c(arg)

Description

Returns the respective character string.

Input Arguments

arg : integer | date | interval

Output

CHAR(?)

The size of the output character string varies with respect to the input argument. Examples of particular importance are given below.

Examples

c(5) = 5
Here the input argument is an integer and the output is a CHAR(6) string.

c(1993-11-23) = 1993-11-23
Here the input argument is an INGRES date and the output is a CHAR(25) string.

c('[1992-11-01,1993-10-31)') = [1992-11-01,1993-10-31).
Here the input argument is an ORES interval and the output is a CHAR(23) string.

The argument of c may be another function:

c(5+3) = 8.
Here the input argument is an arithmetic operation and the output is the result of the operation, provided as a CHAR(13) string.

c('ORES^'+'Project') = ores^project.
Here the input argument is the string concatenation operation and the output is the result of the operation, provided as a CHAR(12) string ("^" denotes the space character).

Function c can also be used as argument to other functions, including those defined in ORES:

window('1993-01-01'+','+c(2+3)+','+'0') = [^1993-01-01,^1993-01-06^).
Here the input argument of window is a concatenation of strings and the output is an interval, where "^" denotes the space character.

Format

dist(d1, d2)

Description

It returns the absolute number of days date d2 is away from date d1.

Input Arguments

d1, d2: ORES date.

Output

integer.

Examples

dist('1993-01-01', '1993-01-05') = 4
dist('1993-01-05', '1993-01-01') = 4.

Format

dur(di)

Description

Returns the number of dates in an interval.

Input Arguments

di: ORES interval.

Output

integer.

Example

dur('[1993-06-01,1993-06-11)') = 10.

Format

interv(d1, d2)

Description

Returns interval [d1, d2), where d1 and d2 are dates, provided that d1 < d2.

Input Arguments

d1, d2: ORES date.

Output

ORES interval.

Example

interv('1993-06-01','1993-06-30') = [^1993-06-01,^1993-06-30^).
interv('1993-06-30','1993-06-01') returns an error message.

"^" denotes the space character.

Format

intervsect(di1, di2)

Description

Returns the intersection of two intervals, di1 and di2, provided that di1 and di2 have at least one common point.

Input Arguments

di1, di2: ORES interval.

Output

ORES interval.

Example

intervsect('[1993-06-01,1993-06-25)','[1993-06-15,1993-06-30)' = '[^1993-06-15,^1993-06-25^)'

intervsect('[1993-06-01,1993-06-25)','[1993-07-01,1993-07-25)' returns an error message.

"^" denotes the space character.

Format

maxdate()

Description

Returns the greatest supported date. (In the current implementation it has been defined that the greatest date is 2030-12-30).

Input Arguments

none.

Output

ORES date.

Example

maxdate() = 2030-12-30.

Format

merge(di1, di2)

Description

Returns an interval which is the union of two intervals di1 and di2, provided that di1 and di2 are adjacent or have at least one common date.

Input Arguments

di1, di2: ORES interval.

Output

ORES interval.

Examples

merge('[1993-01-01,1993-01-31)','[1993-01-31,1993-02-28)') = [^1993-01-01,^1993-02-28^)

merge('[1993-01-01,1993-01-31)','[1993-01-15,1993-02-28)') = [^1993-01-01,^1993-02-28^)

merge('[1993-01-01,1993-01-31)','[1993-03-01,1993-03-30)') returns an error message.

"^" denotes the space character.

Format

middle(di)

Description

Returns the smallest date which is closest to the middle of an interval.

Input Arguments

di: ORES interval.

Output

ORES date.

Examples

middle('[1993-01-01,1993-01-06)') = 1993-01-03
middle('[1993-01-01,1993-01-07)') = 1993-01-03.

Format

mindate()

Description

Returns the lowest supported date. (In the current implementation it has been defined that the lowest date is 1970-01-01).

Input Arguments

none.

Output

ORES date.

Example

mindate() = 1970-01-01.

Format

now()

Description

Returns the current date.

Input Arguments

none.

Output

ORES date.

Example

If the current day is 1993-11-23 then
now() = 1993-11-23.

Format

span(d1, d2)

Description

It returns the number of days date d2 is away from date d1.

Input Arguments

d1, d2: ORES date.

Output
integer.

Examples

span('1993-01-01', '1993-01-05') = 4
span('1993-01-05', '1993-01-01') = -4.

Format

start(di)

Description

Returns the start of an interval.

Input Arguments

di: ORES interval.

Output

ORES date.

Example

start('[1992-11-01,1994-10-01)') = 1992-11-01.

Format

stop(di)

Description

Returns the stop of an interval.

Input Arguments

di: ORES interval.

Output

ORES date.

Example

stop('[1992-11-01,1994-10-01)') = 1994-10-01.

Format

succ(d, k)

Description

Returns a date which is k days away from date d.

Input Arguments

d: ORES date
k: integer.

Output

ORES date.

Example

succ('1993-06-01', 10) = 1993-06-11
succ('1993-06-11', -10) = 1993-06-01.

Format

tointerv(d)

Description

Transforms a date to an elementary interval. (*Elementary* is an interval of exactly one date).

Input Arguments

d: ORES date .

Output

ORES interval.

Example

tointerv('1993-06-01') = [^1993-06-01,^1993-06-02^).

"^" denotes the space character.

Format

topoint(di)

Description

Returns the start of an elementary interval. (*Elementary* is an interval of exactly one date).

Input Arguments

di: ORES interval.

Output

ORES date.

Example

topoint('[1993-06-01,1993-06-02)') = 1993-06-01.
topoint('[1993-06-01,1993-06-10)' returns an error message.

Format

window(chr)

Description

Argument chr is a character string with format 'd,m,n', where d is ORES date and m, n are integers, m > 0. If n ≥ 0 the function returns the n-th (n = 0, 1, 2, ...) interval of duration m after date d whereas if n < 0 it returns the n-th (n = -1, -2, ...) interval before date d.

Input Arguments

chr: A character string with format 'd,m,n' where

d: ORES date

m: integer, m > 0

n: integer.

From the above format it can be seen that space characters are not allowed at any place of the input argument.

Output

interval.

Examples

window('1993-01-01,5,0') = [1993-01-01,1993-01-06)
window('1993-01-01,5,1') = [1993-01-06,1993-01-11)
window('1993-01-01,5,-1') = [1992-12-27,1993-01-01)
window('1993-01-01,-3,0') returns an error message.

Notes

1. It would be most desirable if this function accepted three distinct arguments but INGRES imposes a restriction, that the number of arguments of a function defined directly in the INGRES server, should be at most 2.
2. This function will probably be redefined, so as not to accept a 0 value for n.

Format

windowno(chr)

Description

Argument chr is a character string with format 'd1,m,d2', where d1 and d2 are ORES dates and m is an integer, m > 0. If one starts from date d1 and all intervals of duration m which are to the right of d1 are counted (starting counting from 0) whereas all intervals of duration m which are to the left of d1 are also counted (starting counting from -1), the function returns the count of the interval in which date d2 lies.

Input Arguments

chr: A character string with format 'd1,m,d2' where
d1, d2: ORES date
m: integer, m>0.

From the above format it can be seen that space characters are not allowed at any place of the input argument.

Output

integer.

Examples

windowno('1993-01-01,5,1993-01-03') = 0
windowno('1993-01-01,5,1993-01-08') = 1
windowno('1993-01-01,5,1992-12-28') = -1
windowno('1993-01-01,-3,1993-01-03') returns an error message.

Notes

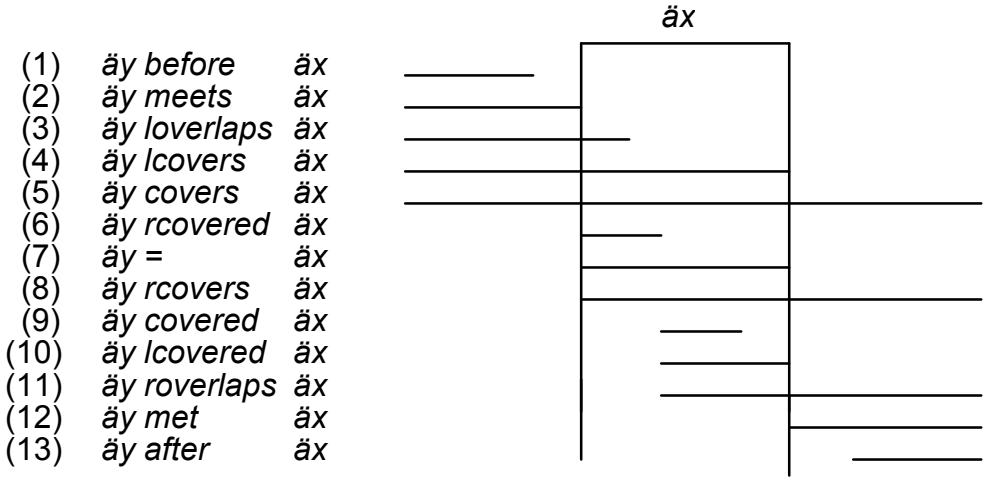
1. It would be most desirable if this function accepted three distinct arguments but INGRES imposes a restriction, that the number of arguments of a function defined directly in the INGRES kernel, should be at most 2.
2. This function will probably be redefined, so as not to accept a 0 value for m.

3.4 Relational Operators

VT-RA incorporates all the ordinary relational operators ($<, \leq, =, \neq, \geq, >$) which can be applied between two ordinary pieces of data. VT-RA also incorporates all the interval relational operators. Specifically, if δx and δy are two pieces of data of type interval and *irp* is an interval relational operator, then the general syntax is

$$\delta y \; irp \; \delta x.$$

It is shown below when each of these operators evaluates to true. Their definition can be found in [01P 1993b], Appendix A.



δy	*psubinterv*	*δx*	⇔	*(6 V 9 V 10) (δy is a pure subinterval of δx)*
δy	*subinterv*	*δx*	⇔	*(6 V 7 V9 V10) (δy is a subinterval of δx)*
δy	*psupinterv*	*δx*	⇔	*(4 V 5 V 8) (δy is a pure superinterval of δx)*
δy	*supinterv*	*δx*	⇔	*(4 V 5 V 7V8) (δy is a superinterval of δx)*
δy	*overlaps*	*δx*	⇔	*(3V 11)*
δy	*merges*	*δx*	⇔	*(2 V 3 V . . . V12)*
δy	*cp*	*δx*	⇔	*(3 V 4 V . . . V 11) (δy has common points with δx)*
δy	*precedes*	*δx*	⇔	*(1 V 2 V 3 V 4 V 5V 6)*
δy	*follows*	*δx*	⇔	*(8 V 9 V 10 V11 V 12 V 13)*
δy	*prequals*	*δx*	⇔	*(1 V 2V 3 V 4 V 5 V 6 V 7) (δy precedes or equals δx)*
δy	*folequals*	*δx*	⇔	*(7 V 8V 9V 10 V 11V 12 V13) (δy follows or equals δx)*
δy	*adjacent*	*δx*	⇔	*(2 V12)*

3.5 Relational Algebra Operations

The definition and specifications of VT-RA operations can be found in [01P 93a]. Certain improvements of major practical interest are the following:

1. Operations **Select** and **Project** incorporate all the VT-SQL functions [01P 93b] and all the INGRES functions. In addition, all these functions can be combined, to form composite functions.
2. Optimisation algorithms have been used for the development of operations **Unfold**, **Fold**, **Normalise**, **Punion** and **Pexcept**.
3. Operation **Project** enables the renaming of attributes.
4. Four more operations have been defined, **Display, Drop, Help**, and **Quit**.
5. The functionality of the VT-RA operations is fully compatible with the functionality of standard SQL.

Full specifications of the above enhanced features are given in the sub-sections which follow.

It should be noted that operation **Reformat** has not been implemented. This does not cause any practical problem because **Reformat** is composite and can be expressed by sequences of **Unfold** and **Fold** operations.

General Rules

Two general rules which all the VT-RA operations obey, are the following:

1. No redundant spaces (including spaces after a ",") are allowed anywhere in the formulation of a VT-RA operation except exactly one space, to separate keywords from the remainder portion of the input string.
2. If at the execution of an operation one of the input arguments is invalid, or an invalid result is computed at some intermediate step, the operation is completely dropped.

As an example of this second rule, assume that a **project** operation has been formulated. Assume also that the operation requires a projection on the result computed by succ(Date, Number), where Date and Number are attribute names of type CHAR(10) and INTEGER4, respectively. If one of the strings recorded in Date is invalid, then the operation is dropped and an error message is returned. This behaviour is fully compatible with that of standard SQL.

All the VT-RA operations are given next in alphabetic order. Their names are given in **bold**. Square brackets (**[**, **]**) which are also given in **bold**, form part of the syntax.

Display

Syntax

display <table-name>

Description

Displays the contents of <table-name>.

Drop

Syntax

drop <table-name>

Description

Drops <table-name>.

Except

Syntax

<table-name-3>=<table-name-1> **except** <table-name-2>

Description

It retrieves into <table-name-3> the tuples of <table-name-1> except those tuples which are also in <table-name-2>.

Specifications

1. <table-name-1> and <table-name-2> must already exist in the database.
2. <table-name-3> must not exist before the execution of **except**.
3. <table-name-1> and <table-name-2> must be union-compatible, exactly as this compatibility is interpreted by INGRES. (*)
4. The scheme or <table-name-3> is identical with the scheme of <table-name-1>.
5. <table-name-3> must not contain duplicate tuples.

(*) Two relations are union-compatible if they satisfy the following two conditions:

(i) They have the same number of attributes.

(ii) The i-th attribute of <table-name-1> is type-compatible with the i-th attribute of <table-name-2>.

Two attributes are type-compatible if their types are the same, regardless of their length. For example, char(20) is type-compatible with char(15) and i2 is type-compatible with i4. In addition, types MONEY and VARCHAR are type-compatible with types FLOAT and CHAR, respectively.

Fold

Syntax

<table-name-3>=**fold[**<attribute-list>**]**(<table-name-1>)

Description

It folds successively <table-name-1> on the attributes in <attribute-list> and returns the result in <table-name-3>. Folding starts from the leftmost attribute in <attribute-list> and proceeds to the rightmost attribute.

Specifications

1. <table-name-1> must already exist in the database.
2. <table-name-3> must not exist before the execution of **fold**.
3. The <attribute-list> must be a non-empty list of the attribute names of <table-name-1> of a date or an interval type.
4. The number of attributes in <attribute-list> is limited by the number of columns a relation may have in INGRES.
5. Before folding starts, duplicate tuples are eliminated from <table-name-1>.
6. The domain of all the folded attributes in <table-name-3> is of type interval.
7. If <table-name-1> is empty, then <table-name-3> is empty, too.
8. <table-name-3> must not contain duplicate tuples.

Convention

It is assumed that the attributes in <attribute-list> contain intervals whose start and stop points are not-null.

Help

Syntax

help

Description

Gives a list of the relations in the database.

Normalise

Syntax

<table-name-3>=**normalise[**<attribute-list>**]**(<table-name-1>)

Description

It normalises <table-name-1> on the attributes in <attribute-list> and retrieves the result in <table-name-3>. The operation is semantically equivalent to <table-name-3>=**fold[**<attribute-list>**]** ∘ **unfold[**<attribute-list>**]**(<table-name-1>).

Specifications

1. <table-name-1> must already exist in the database.
2. <table-name-3> must not exist before the execution of **normalise**.
3. The <attribute-list> must be a non-empty list of attributes of <table-name-1> of a date or interval type.
4. Before folding starts, duplicate tuples are be eliminated from **unfold[**<attribute-list>**]**(<table-name-1>).
5. The number of attributes in <attribute-list> is limited by the number of columns a relation may have in INGRES.
6. The domain of all the normalised attributes in <table-name-3> is of type interval.
7. If the initial relation is empty, the output relation is also empty.
8. <table-name-3> must not contain duplicate tuples.

Convention

It is assumed that the attributes in <attribute-list> contain intervals whose start and stop points are not-null.

Pexcept

Syntax

<table-name-3>=<table-name-1> **pexcept[**<attribute-list>**]**<table-name-2>)

Description

It retrieves into <table-name-3> the points-difference of relations <table-name-1> and <table-name-2>. The operation is semantically equivalent to

<table-name-3> =
fold[<attribute-list>**]**
(
unfold[<attribute-list>**]**(<table-name-1>)
except
unfold[<attribute-list>**]**(<table-name-2>)
).

Specifications

1. <table-name-1> and <table-name-2> must already exist in the database.
2. <table-name-3> must not exist before the execution of **pexcept**.
3. The <attribute-list> must be a non-empty list of the attributes of <table-name-1> of a date or interval type.
4. <table-name-1> and <table-name-2> must be union-compatible, exactly as this compatibility is interpreted by INGRES. (*)
5. Duplicate tuples must be eliminated from both **unfold[**<attribute-list>**]**(<table-name->) and **unfold[**<attribute-list>**]**(<table-name-2>). Next, their difference is obtained and finally **fold** is applied to the result.

(*) Two relations are union-compatible if they satisfy the following two conditions:

(i) They have the same number of attributes.

(ii) The i-th attribute of <table-name-1> is type-compatible with the i-th attribute of <table-name-2>.

Two attributes are type-compatible if their types are the same, regardless of their length. For example, char(20) is type-compatible with char(15) and i2 is type-compatible with i4. In addition, types MONEY and VARCHAR are type-compatible with types FLOAT and CHAR, respectively.

6. The scheme of <table-name-3> is the same with the scheme of <table-name-1> except that the domain of the attributes in <attribute-list> is of an interval type.
7. <table-name-3> must not contain duplicate tuples.

Convention

It is assumed that the attributes in <attribute-list> contain intervals whose start and stop points are not-null.

Product

Syntax

<table-name-3>=<table-name-1> **product** <table-name-2>.

Description

It retrieves into <table-name-3> the Cartesian product of <table-name-2> and <table-name-3>.

Specifications

1. <table-name-1> and <table-name-2> must already exist in the database.
2. <table-name-3> must not exist before the execution of **product**.
3. <table-name-1> and <table-name-2> must not have attributes with common names.
4. The attributes of <table-name-3> are those of the attributes of relations <table-name-1> and <table-name-2>.
5. <table-name-3> must not contain duplicate tuples.

Project

Syntax

<table-name-3>=**project[**<name-attribute-list> | ***]**(<table-name-1>)

where

<name-attribute> ::= [<attribute-name-2>=]<attribute-name-1>
[<attribute-name>=]<function>.

Description

It retrieves into <table-name-3> the projection of all the tuples of <table-name-1> on all the attributes contained in <attribute-list>.

Specifications

1. <table-name-1> must already exist in the database.
2. <table-name-3> must not exist before the execution of **project**.
3. The <name-attribute-list> must be non-empty.
4. The attributes of <table-name-3> are those specified in <name-attribute-list>.
5. A "*" for <name-attribute-list> indicates a projection on all the attributes of <table-name-1>.
6. If <function> is specified the respective attribute of <table-name-3> contains the result returned by the function.
7. If "<attribute-name-2>=" is specified then an attribute renaming occurs or a name is assigned to an attribute which contains a computed result.
8. If "<function>" is specified, not proceeded by <"<attribute-name-2>=" then a default <attribute-name> (col1, col2, ...) is assigned by the system.
9. <table-name-3> must not have two attributes with identical names.
10. <table-name-3> must not contain duplicate tuples.

Punion

Syntax

<table-name-3>=<table-name-1> **punion[**<attribute-list>**]**<table-name-2>)

Description

It retrieves into <table-name-3> the points-union of relations <table-name-1> and <table-name-2>. The operation is semantically equivalent to
<table-name-3> =
fold[<attribute-list>**]** ∘ **unfold[**<attribute-list>**]**(<table-name-1> **union** <table-name-2>).

Specifications

1. <table-name-1> and <table-name-2> must already exist in the database.
2. <table-name-3> must not exist before the execution of **punion**.
3. The <attribute-list> must be a non-empty list of the attributes of <table-name-1> of a date or interval type.
4. <table-name-1> and <table-name-2> must be union-compatible, exactly as this compatibility is interpreted by INGRES. (*)
5. Before folding starts, duplicate tuples are eliminated from **unfold[**<attribute-list>**]**(<table-name-1> **union** <table-name-2>).
6. The scheme of <table-name-3> is the same with the scheme of <table-name-1> except that the domain of the attributes in <attribute-list> is of a interval type.
7. <table-name-3> must not contain duplicate tuples.

(*) Two relations are union-compatible if they satisfy the following two conditions:

(i) They have the same number of attributes.

(ii) The i-th attribute of <table-name-1> is type-compatible with the i-th attribute of <table-name-2>.

Two attributes are type-compatible if their types are the same, regardless of their length. For example, char(20) is type-compatible with char(15) and i2 is type-compatible with i4. In addition, types MONEY and VARCHAR are type-compatible with types FLOAT and CHAR, respectively.

Convention

It is assumed that the attributes in <attribute-list> contain intervals whose start and stop points are not-null.

Select

Syntax

<table-name-3>=**select[**<search-condition>**]**(<table-name-1>)

Description

It retrieves into <table-name-3> the tuples of <table-name-1> which satisfy the <search-condition>.

Specifications

1. <table-name-1> must already exist in the database.
2. <table-name-3> must not exist before the execution of **select**.
3. <table-name-3> must not contain duplicate tuples.
4. If no <search-condition> is specified then all the tuples of <table-name-1> are selected.
5. The <search-condition> is fully compatible with the SQL <search-condition> and enables the incorporation of all the functions and relational operators of both SQL and VT-RA.
6. <search-condition> is defined as follows:

<search-condition> ::= <empty-string>
| <boolean-term>
| <search-condition> OR <boolean-term>.

<boolean-term> ::= <boolean-factor>| <boolean-term> AND <boolean-factor>.

<boolean-factor> ::= [NOT] <boolean-primary>.

<boolean-primary> ::= <predicate> | (<search-condition>).

<predicate> ::= <value-exp> <compare-op> <value-exp>.

<compare-op> ::= <> | = | < | > | <= | >= | <interval-compare-op>.

<interval-compare-op> ::= *before*
meets
| *loverlaps*

| *lcovers*
| *covers*
| *rcovered*
| =
| *rcovers*
| *covered*
| *lcovered*
| *roverlaps*
| *met*
| *after*
| *psubinterv*
| *subinterv*
| *psupinterv*
| *supinterv*
| *overlaps*
| *merges*
| *cp*
| *precedes*
| *follows*
| *prequals*
| *folequals*
| *adjacent*.

<value-exp> : : = <attribute-name>
| <literal>
| <function>(<attribute-name>)
| <function(<literal>).

Unfold

Syntax

<table-name-3>=**unfold[**<attribute-list>**]**(<table-name-1>)

Description

It unfolds successively <table-name-1> on the attributes in <attribute-list> and retrieves the result in <table-name-3>. Folding starts from the leftmost attribute in <attribute-list> and proceeds to the rightmost attribute.

Specifications

1. <table-name-1> must already exist in the database.
2. <table-name-3> must not exist before the execution of **unfold**.
3. The <attribute-list> must be a non-empty list of the attributes of <table-name-1> of a date or interval type.
4. The number of attributes in <attribute-list> is limited by the number of columns a relation may have in INGRES.
5. The domain of all the unfolded attributes in <table-name-3> is of type date.
6. If the initial relation is empty, the output relation is also empty.
7. <table-name-3> must not contain duplicate tuples.

Convention

It is assumed that the attributes in <attribute-list> contain intervals whose start and stop points are not-null.

Union

Syntax

<table-name-3>=<table-name-1> **union** <table-name-2>

Description

It retrieves into <table-name-3> the union of two relations.

Specifications

1. <table-name-1> and <table-name-2> must already exist in the database.
2. <table-name-3> must not exist before the execution of **union**.
3. <table-name-1> and <table-name-2> must be union-compatible, exactly as this compatibility is interpreted by INGRES. (*)
4. The scheme or <table-name-3> is identical with the scheme of <table-name-1>.
5. <table-name-3> must not contain duplicate tuples.

(*) Two relations are union-compatible if they satisfy the following two conditions:

(i) They have the same number of attributes.

(ii) The i-th attribute of <table-name-1> is type-compatible with the i-th attribute of <table-name-2>.

Two attributes are type-compatible if their types are the same, regardless of their length. For example, char(20) is type-compatible with char(15) and i2 is type-compatible with i4. In addition, types MONEY and VARCHAR are type-compatible with types FLOAT and CHAR, respectively.

Quit

Syntax

quit

Description

Exits the VT-RA environment.

4. IMPLEMENTATION

VT-RA has been implemented in two distinct steps, discussed separately below.

4.1 Implementation of the Interval Data Type and Relevant Functions

The *Object Management* of INGRES is a module that allows the user to add to the INGRES *Data Base Server* new data types and new SQL functions. A user-defined data type can be used in any context in which a standard INGRES data type can be used. Added SQL functions can be used in queries and can manipulate both user-defined and standard INGRES data types. However, the code for the definition of new types and functions has to be written by the user. To this end, INGRES supplies the *Object Management Extension*s which contains the complete structure and symbol definitions which are necessary.

Once the code which supports a user-defined data type or a new function has been completed, it runs as part of the underlying INGRES System. All the DBMS modules access it, therefore it is available to the entire installation. Because of this, user-supplied code has the same rights, privileges and responsibilities with the INGRES code.

The ORES interval type, the VT-RA functions and VT-RA relational operators have been implemented using this module, in language C. Some remarks on their implementation are the following:

ORES Interval Data Type and VT-RA Functions

Its format is $[d_i, d_j)$, where d_i, d_j are two valid dates (the start and stop of the interval), satisfying

$$1970\text{-}01\text{-}01 \leq d_i < d_j \leq 2030\text{-}12\text{-}30.$$

Because of this, the following is a valid VT-SQL statement:

```
CREATE TABLE   TEST(Id   I4,
                    Name CHAR(15),
                    Time INTERVAL)
```

Internally, an interval is stored as two distinct long integers (32 bits each), which represent its start and stop. The offset of each integer represents a date, in the range given above, expressed in seconds. The advantage of the use of a numeric format, enabled the development of efficiently running code for the interval functions and interval relational operations. However, one limitation of INGRES is that it does not allow a

user-defined data type to be composed of complex INGRES data types, such as *Date* and *Money*. To overcome this limitation, all the functions that return a date, have been implemented in such a way so as to return a CHAR(10) string. This string is formatted according to the ISO date format "YYYY-MM-DD". If the result of a function needs be compared with an INGRES date data type, INGRES coerces the string into a date data type.

There are two ways to insert an interval into a relation:

(i) Supply the interval as a character string: The string can be any form of text-string type from those supported by INGRES. The CHAR type is preferred since most of the tests have been performed using this type. The input format has to be

[YYYY-MM-DD, YYYY-MM-DD)

of a CHAR(23) type.

(ii) Incorporate the *interv* function: Then the form is

interv('YYYY-MM-DD', 'YYYY-MM-DD')

An interval is displayed as a CHAR(26) string, with format

[^YYYY-MM-DD, ^YYYY-MM-DD^)

where "^" denotes the space character.

Another limitation of INGRES is that at most two input arguments can be passed to a user-defined function. This was an inconvenience for functions *window* and *windowno*, since they both require three input arguments, according to the specifications in [01P 93a]. The easiest way to overcome this problem was to redefine the specifications of these functions: Their input has been thus defined to be a single character string, composed by the concatenation of the initially defined arguments. This strings also includes two commas (,) which function as the physical separators of the three arguments. An internal string processing decomposes the string into the three initially defined arguments.

VT-RA Relational Operators

One more limitation is that INGRES does not allow the definition of new relational operators. (In fact only a redefinition of the existing SQL operators is allowed). To emulate therefore the behaviour of the VT-RA, each of them has to be internally transformed into an equivalent boolean function with two arguments.

4.2 Implementation of VT-RA Operations

The VT-RA parser has been developed using *lex*. The input string is processed and appropriate action is taken as follows:

(i) If the input string represents either of the VT-RA operations **Union**, **Except**, **Project**, **Product** or **Select**, it is transformed into an equivalent SQL statement. The parsing of **Select** has an additional requirement, that interval relational operators are transformed to equivalent user-defined boolean functions.

(ii) If the input string represents either of the VT-RA operations **Unfold**, **Fold**, **Normalise**, **Punion** or **Pexcept**, then the respective routine, implemented in C, is invoked. Optimised code has been developed for the implementation of these operations, described in detail in [Lorentzos et al 92] and [Lorentzos & Manolopoulos 93].

The INGRES dictionary is assessed before the execution of certain operations, to verify that the columns of the input table(s) are of the proper data type.

5. CONCLUSIONS

In this report we presented the Valid Time Relational Algebra which has been developed in the ORES project. The evaluation tests of the developed programs have been exhaustive and have shown that the software runs according to the specifications.

REFERENCES

[01P 93a] 'Specification of Valid Time Formalism', ORES Deliverable C3, Apr. 1993.

[01P 93b] 'Specification of Valid Time SQL', ORES Deliverable D2, Amendment I, Oct. 1993.

[INGRES 89] Ingres 'SQL Reference Manual', Release 6.3, Nov. 1989.

[INGRES 91] Ingres 'Object Management Extension User's Guide', Release 6.4, Dec. 1991.

[Lorentzos et al 92] N. A. Lorentzos, A. Poulovassilis, and C. Small. 'Update Operations For Multi-dimensional Interval Data', Int. Rept., Informatics Laboratory, Agricultural University of Athens, 1993.

[Lorentzos & Manolopoulos 93] N. A. Lorentzos, and Y. Manolopoulos. 'Optimised Update of 2-Dimensional Interval Relations' Paper accepted to the 4th Greek Computer Society Conference, 1993.

APPENDIX A - VT-RA EVALUATION

1. Introduction

The present appendix contains the tests of VT-RA. The evaluation has been based on the notion of *test cases* also known as *black box testing*. A *test case* is a specific scenario with a predefined input and output aiming at checking only *one* aspect at a time. The test cases have been applied to a test database particularly created for the evaluation.

Two kinds of tests have been used, *normal* ones, with a valid input and an expected valid output and tests which should normally be flagged by an error message. Effort has been made for the set of tests to be exhaustive.

The remainder of this appendix is outlined as follows :

In section 2 we provide a sample database against which part of the evaluation took place. In section 3 we provide tables on the evaluation of VT-RA functions. In section 4 we provide tables on the the evaluation of VT-RA relational operators. In section 5 we provide tables on the evaluation of VT-RA operations. Conclusions are drawn in the last section.

2. Sample Database

Part of the evaluation test has been made against the tables which follow. All these tables contain valid data and care has been taken for them to contain duplicate rows and to be non-normalised. Similar tables containing invalid data have also been used. Finally, tables with two time related attributes have been created that contain duplicate rows and to are non-normalised .

(S)ALARY

Name	Amount	Days
John	10000	[1993-06-02,1993-06-06)
John	10000	[1993-06-09,1993-06-12)
John	12000	[1993-06-15,1993-06-18)
Alex	14000	[1993-06-09,1993-06-12)

(A)SSIGNMENT

Name	Dept.	Days	Type
John	shoe	[1993-06-03,1993-06-07)	salesman
John	food	[1993-06-07,1993-06-11)	supervisor
John	toys	[1993-06-11,1993-06-15)	salesman
Alex	shoe	[1993-06-05,1993-06-10)	supervisor
Mary	toys	[1993-06-05,1993-06-11)	supervisor

(D)uplicated SALARY

Name	Amount	Days
John	10000	[1993-06-02,1993-06-06)
John	10000	[1993-06-02,1993-06-06)
John	12000	[1993-06-15,1993-06-18)
Alex	14000	[1993-06-09,1993-06-12)

(N)on Normalised ASSIGNMENT

Name	Dept.	Days	Type
John	shoe	[1993-06-03,1993-06-07)	salesman
John	food	[1993-06-07,1993-06-11)	supervisor
John	food	[1993-06-08,1993-06-15)	supervisor
Alex	shoe	[1993-06-05,1993-06-10)	supervisor
Mary	toys	[1993-06-05,1993-06-11)	supervisor

(I)nflation

Country	Percentage	Period
greece	0.15	[1993-06-01,1993-06-02)
spain	0.10	[1993-06-03,1993-06-04)
EEC	0.08	[1993-06-07,1993-06-08)

(O)vertime

Name	Date	No of Hours
John	1993-06-02	2
john	1993-06-03	2
John	1993-06-04	2
john	1993-06-05	2
John	1993-06-15	1
John	1993-06-20	2
Alex	1993-06-20	2

Dup(L)icated Overtime

Name	Date	No of Hours
John	1993-06-02	2
John	1993-06-03	2
John	1993-06-15	1
John	1993-06-15	1
Alex	1993-06-20	2

(C)complication

Patient	Complication	Period
john	Hypotassemia	[1993-06-01,1993-06-05)
john	Hyperglykemia	[1993-06-01,1993-06-09)
john	Leykopenia	[1993-06-01,1993-06-10)
john	Hypomagnesemia	[1993-06-01,1993-06-15)
john	Dialysis	[1993-06-01,1993-06-20)
john	Atelectasis	[1993-06-08,1993-06-10)
john	Hemothorax	[1993-06-08,1993-06-15)
john	Pneumothorax	[1993-06-08,1993-06-20)
john	Ultrafiltration	[1993-06-09,1993-06-11)
john	Psychosis	[1993-06-12,1993-06-15)
john	Pancreatitis	[1993-06-12,1993-06-20)
john	Bone	[1993-06-14,1993-06-20)
john	Hemolytic Anemia	[1993-06-17,1993-06-20)

Interval for comparison

S2

Name	Amount	Date
john	10000	[1993-06-17,1993-06-20)

D2

Name	Amount	Date
john	10000	[1993-06-17,1993-06-20)
John	10000	[1993-06-17,1993-06-20)
John	12000	[1993-06-15,1993-06-18)
Alex	14000	[1993-06-09,1993-06-12)

(SO)

Name	Amount	Days	Hours	Date
John	10000	[1993-06-02,1993-06-06)	2	1993-06-02
John	10000	[1993-06-09,1993-06-12)	2	1993-06-03
John	10000	[1993-09-09,1993-06-12)	2	1993-06-04
john	12000	[1993-09-05,1993-06-12)	1	1993-06-15
alex	14000	[1993-06-09,1993-06-12)	2	1993-06-20

(OO)

Name	Date	Hours	Period
John	1993-06-02	2	1993-06-02
John	1993-06-03	2	1993-06-02
John	1993-06-04	2	1993-06-02
john	1993-06-04	2	1993-06-03
alex	1993-06-20	2	1993-06-03

(SOD)

Name	Amount	Days	Hours	Date
John	10000	[1993-06-02,1993-06-06)	2	1993-06-02
John	10000	[1993-06-09,1993-06-12)	2	1993-06-03
John	10000	[1993-09-09,1993-06-12)	2	1993-06-03
john	12000	[1993-09-05,1993-06-12)	1	1993-06-15
alex	14000	[1993-06-09,1993-06-12)	2	1993-06-20

(OOD)

Name	Date	Hours	Period
John	1993-06-02	2	1993-06-02
John	1993-06-03	2	1993-06-02
John	1993-06-03	2	1993-06-02
john	1993-06-04	2	1993-06-03
alex	1993-06-20	2	1993-06-03

(DA)

name	amount	dept	days	period
john	10000	shoe	[1993-06-02,1993-06-06)	[1993-06-03,1993-06-07)
john	10000	food	[1993-06-02,1993-06-06)	[1993-06-03,1993-06-07)
john	10000	food	[1993-06-09,1993-06-12)	[1993-06-07,1993-06-11)
john	12000	toys	[1993-06-15,1993-06-18)	[1993-06-11,1993-06-15)
john	14000	shoe	[1993-06-09,1993-06-12)	[1993-06-05,1993-06-11)

(DAN)

name	amount	dept	days	period
john	10000	shoe	[1993-06-02,1993-06-06)	[1993-06-03,1993-06-07)
john	10000	food	[1993-06-02,1993-06-06)	[1993-06-07,1993-06-11)
john	10000	food	[1993-06-02,1993-06-06)	[1993-06-08,1993-06-15)
john	12000	toys	[1993-06-15,1993-06-18)	[1993-06-11,1993-06-15)
john	14000	shoe	[1993-06-09,1993-06-12)	[1993-06-05,1993-06-11)

(E)

name	amount	date

3. Evaluation of VT-RA Functions

The following evaluation has been made through the use of INGRES SQL as well as through the "project" operation of VT-RA. Here only the SQL portion of tests is included. The following categories of tests have been included :

1) Syntax Errors
2) Invalid Input
3) Output beyond required range
4) Valid Syntax and Input Data

As far as we can estimate, these tests have been exhaustive.

Start(di)

di = "[YYYY-MM-DD,YYYY-MM-DD)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument	select start('1993-09-27') from s	<error message>	√	
Any string as argument	select start('nonsense') from s	<error message>	√	
Not valid date interval	select start('[1993-09-29,1993-09-27)') from s	<error message>	√	
Not complete date interval	select start('[1993-09,1993-10)') from s	<error message>	√	
Date argument	select start(date) from o	<error message>	√	
Valid Syntax and Input Data				
date comparison	select * from s where start(days)=date('1993-06-09')		√	
simple	select start('[1993-09-27,1993-09-29)') from s		√	

Stop(di)

di = "[YYYY-MM-DD,YYYY-MM-DD)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument	select stop('1993-09-27') from s	<error message>	√	
Any string as argument	select stop('nonsense') from s	<error message>	√	
Not valid date interval	select stop('[1993-09-29,1993-09-27)') from s	<error message>	√	
Not complete date interval	select stop("[1993-09,1993-10[)) from s	<error message>	√	
Date argument	select stop(date) from o	<error message>	√	
Valid Syntax and Input Data				
date comparison	select * from s where stop(days)=date('1993-06-09')		√	
simple	select stop('[1993-09-27,1993-09-29)') from s			

Date Topoint(di)

di = "[YYYY-MM-DD,YYYY-MM-DD+1)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument	select topoint('1993-09-27') from s	<error message>	√	
Any string as argument	select topoint('nonsense') from s	<error message>	√	
Not valid date interval	select topoint('[1993-09-29,1993-09-27)') from s	<error message>	√	
Duration not 1	select topoint('[1993-09-20, 1993-09-25')) from s	<error message>	√	
Date argument	select topoint(date) from o	<error message>	√	
Valid Syntax and Input Data				
date comparison	select * from s where topoint(days)=date('1993-06-012')		√	
simple	select topoint('[1993-09-27,1993-09-28)') from s		√	

DateInterv Tointerv(d)

d = "YYYY-MM-DD"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors				
Invalid Input				
Date Interval argument	select tointerv('[1993-09-27,1993-09-29)') from s	<error message>	√	
Any string as argument	select tointerv('nonsense') from s	<error message>	√	
Not valid date	select tointerv('1993-09') from s	<error message>	√	
Date argument	select tointerv(date) from o	<error message>	√	
Valid Syntax and Input Data				
date comparison	select * from o where tointerv(c(date))='[1993-06-20,1993-06-21)'		√	
simple	select tointerv('1993-09-27') from s		√	

DateInterv Intervsect(di1,di2)

di1, di2 = "[YYYY-MM-DD,YYYY-MM-DD)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument	select intervsect('1993-09-27') from s	<error message>	√	
Any string as argument	select intervsect('nonsense') from s	<error message>	√	
Not valid date interval	select intervsect('[1993-09-29,1993-09-27)','[1993-09-24,1993-09-27)') from s	<error message>	√	
Not Common Points	select intervsect('[1993-09-22,1993-09-27)','[1993-09-14,1993-09-17)') from s	<error message>	√	
Date argument	select intervsect(date) from o	<error message>	√	
Valid Syntax and Input Data				
In where clause	select * from n where dur(intervsect(days,'[1993-06-08,1993-06-15)')) < 10		√	
simple	select intervsect('[1993-09-23,1993-09-29)','[1993-09-24,1993-09-27)') from s		√	

Date Succ(d,n)

d = "YYYY-MM-DD"
n = integer

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Any string as argument	select succ('nonsense',2) from s	<error message>	√	
Not valid date	select succ('1993-09',2) from s	<error message>	√	
Date argument	select succ(date,2) from o	<error message>	√	
Date interval argument	select succ(days,5) from o	<error message>	√	
Valid Syntax and Input Data				
Negative 2nd argument	select succ('1993-09-20',-3) from s		√	
In where clause	select * from o where succ(c(date),6) > '1993-06-15'		√	
simple	select succ('1993-09-20',3) from s		√	

Int Dur(di)

di = "[YYYY-MM-DD,YYYY-MM-DD)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument	select dur('1993-09-27') from s	<error message>	√	
Any string as argument	select dur('nonsense') from s	<error message>	√	
Not valid date interval	select dur('[1993-09-29,1993-09-27)') from s	<error message>	√	
Date argument	select dur(date) from o	<error message>	√	
Valid Syntax and Input Data				
In where clause	select * from i where dur(days) > 4		√	
simple	select dur(days) from s		√	

Date middle(di)

di = "[YYYY-MM-DD,YYYY-MM-DD)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument	select middle('1993-09-27') from s	<error message>	√	
Any string as argument	select middle('nonsense') from s	<error message>	√	
Not valid date interval	select middle('[1993-09-29,1993-09-27)') from s	<error message>	√	
Date argument	select midle(date) from o	<error message>	√	
Valid Syntax and Input Data				
In where clause	select * from i where middle(days)='1993-06-10' or middle(days)='1993-06-11'		√	
simple	select middle('[1993-09-27,1993-09-30)') from s		√	

DateInterv merge(di1,di2)

di1,di2 = "[YYYY-MM-DD,YYYY-MM-DD)"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Data				
Date string argument	select merge('[1993-09-10, 1993-09-15)','1993-09-27') from s	<error message>	√	
Any string as argument	select merge(days,'nonsense') from s	<error message>	√	
Not valid date interval	select merge(days,'[1993-09-29,1993-09-27)') from s	<error message>	√	
Non common points	select merge('[1993-09-20, 1993-09-25)','[1993-09-25, 1993-09-30)') from s	<error message>	√	
Date argument	select merge(date,'[1993-09-20, 1993-09-25)') from o	<error message>	√	
Valid Syntax and Input Data				
In where clause	select * from s where dur(merge(days,'[1993-06-05,1993-06-15)') > 6		√	
simple	select merge('[1993-09-20, 1993-09-25'),'[1993-09-22, 1993-09-30)') from s		√	

Int span(d1, d2)

d1, d2 = "YYYY-MM-DD"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Data				
One Date string argument missing	select span('1993-09-27',) from s	<error message>	√	
Any string as argument	select span('1993-09-27','nonsense') from s	<error message>	√	
Date argument	select span(date,date) from o	<error message>	√	
Valid Syntax and Input Data				
In where clause	select * from o where span(c(date),c(date)) > 6	Normal situation	√	
simple	select span('1993-09-27','1993-09-30') from s	Normal situation	√	
simple	select span('1993-09-27','1993-09-10') from s	Normal situation	√	

Positive Int dist(d1,d2)

d1,d2 = "YYYY-MM-DD"

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
One Date string argument missing	select dist('1993-09-10',) from s	<error message>	√	
Any string as argument	select dist('1993-09-10','nonsense') from s	<error message>	√	
Date argument	select dist(date,date) from o	<error message>	√	
Valid Syntax and Input Data				
date comparison	select * from o where dist(c(date),c(date)) > 6	Normal situation	√	
simple	select dist('1993-09-20','1993-09-25') from s	Normal situation	√	
simple	select dist('1993-09-20','1993-09-15') from s	Normal situation	√	

DateInterval window(d1 i1 i2)

d1 = "YYYY-MM-DD"
i1, i2 = integer

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
Date string argument missing	select window(,9,3) from s	<error message>	√	
Any string as argument	select window('nonsense',3,4) from s	<error message>	√	
Not valid date argument	select window('1993-90-29',2,3) from s	<error message>	√	
Valid Syntax and Input Data				
simple	select window('1993-90-29,2,3') from o		√	
In where clause	select * from o where window(c(date)+' ,3,4') > 2		√	

Int windowno(d1,i1, d2)

d1,d2 = "YYYY-MM-DD"
i1 = integer

Test Case Input Description	Test Case Input	Expected Output	Pass	Comments
Syntax Errors		<error message>	√	
Invalid Input				
One Date string argument missing	select windowno('1993-09-10,9,') from s	<error message>	√	
Any string as argument	select windowno('1993-09-01,3,nonsense') from o	<error message>	√	
Not valid date	select windowno('1993-90-29,3,1993-09-27') from s	<error message>	√	
Valid Syntax and Input Data				
In where clause	select * from o where windowno(c(date)+' ,3,' + c(date)) > 2		√	
simple	select windowno('1993-09-20,2,1993-09-25') from s		√	

4. Evaluation of VT-RA Relational Operators

The general format of an interval relational operator is *x irp y* where x and y are intervals and irp is an interval relational operator. The following tests have been made for :

1) Invalid x
2) Invalid y
3) valid x and y

As far as we can estimate these tests have been exhaustive.

Test Case Descr.	Test Case Command	Pass	Comments
Invalid x		√	
Invalid y		√	
Valid x and y		√	
<srch> contains *before*	SE = select [period before interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *meets*	SE = select [period meets interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *loverlaps*	SE = select [period loverlaps interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *lcovers*	SE = select [period lcovers interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *covers*	SE = select [period covers interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *rcovered*	SE = select [period rcovered interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains =	SE = select [period = interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *rcovers*	SE = select [period rcovers interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *covered*	SE = select [period covered interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *lcovered*	SE = select [period lcovered interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *roverlaps*	SE = select [period roverlaps interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *met*	SE = select [period met interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *after*	SE = select [period after interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *psubinterv*	SE = select [period psubinterv interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *subinterv*	SE = select [period subinterv interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *psupinterv*	SE = select [period psupinterv interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *overlaps*	SE = select [period overlaps interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *merges*	SE = select [period merges interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *cp*	SE = select [period cp interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *precedes*	SE = select [period precedes interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *follows*	SE = select [period follows interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *prequals*	SE = select [period prequals interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *folequals*	SE = select [period folequals interv('1993-06-08','1993-06-15')] (C)	√	
<srch> contains *adjacent*	SE = select [period adjacent interv('1993-06-08','1993-06-15')] (C)	√	

5. Evaluation of VT-RA Operations

The following categories of tests have been done :

1) Syntax errors.
2) Violation of VT-RA operations Specifications.
3) Valid Syntax and Specifications satisfaction
4) Support of SQL functions. Not exhaustive test.(applicable to *project* and *select* only)
5) Support of composite VT-RA and SQL functions.(applicable to *project* and *select* only)

Unfold

<tbl3> = **unfold [**<atr>**] (**<tbl1>**)**

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> does not exist	U = unfold [days] (R)	<error message>	√	
<tbl3> exists	A = unfold [days] (S)	<error message>	√	
<atr> not present	U = unfold [] (S)	<error message>	√	
<atr> is not point or interv	U = unfold [name] (S)	<error message>	√	
Valid Syntax and Specifications Statisfaction				
<tbl1> with duplicates, time interval	U = unfold [days] (D)	U should have no duplicates	√	
<tbl1> with duplicates, time ipoint	U = unfold [date] (L)	U should have no duplicates	√	
<atr> is time point	U = unfold [date] (O)	U=O, O remains unchanged	√	
<atr> is time interval	U = unfold [days] (S)	Normal situation	√	
Non normalised <tbl1>	U = unfold [days] (N)	U should have no duplicates	√	
<tbl1> is empty	U = unfold [date] (E)	Normal situation	√	

Fold

<tbl3> = **fold [**<atr>**] (**<tbl1>**)**

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing	<error message>	√	
Specifications Violation				
<tbl1> does not exist	F = fold [days] (R)	<error message>	√	
<tbl3> exists	A = fold [days] (S)	<error message>	√	
<atr> not present	F = fold [] (S)	<error message>	√	
<atr> is not point or interval	F = fold [name] (S)	<error message>	√	`
Valid Syntax and Specifications Satisfaction				
<tbl1> with duplicates, time point	F = fold [date] (L)	F should have no duplicates	√	
<tbl1> with duplicates, time interval	F = fold [days] (D)	F should have no duplicates	√	
<atr> is time interval	F = fold [days] (S)	F=S, S remains unchanged	√	
<atr> is time point	F = fold [date] (O)	Normal situation	√	
Non normalised <tbl1>	F = fold [days] (N)	F should have no duplicates	√	
<tbl1> is empty	F = fold [date] (E)	Normal situation	√	

Normalise

<tbl3> = **normalise [**<atr>**] (**<tbl1>**)**

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> does not exist	NOR = normalise [days] (R)	<error message>	√	
<atr> not present	NOR = normalise [] (S)	<error message>	√	
<atr> is not point or interval	NOR = normalise [name] (S)	<error message>	√	
Valid Syntax and Specifications Satisfaction				
<tbl1> with duplicates, time interval	NOR = normalise [days] (D)	NOR should have no duplicates	√	
<tbl1> with duplicates, time point	NOR = normalise [date] (L)	NOR should have no duplicates	√	
<atr> is time interval	NOR = normalise [days] (S)	[days] of NOR should be Interval	√	
<atr> is time point	NOR = normalise [date] (O)	[date] of NOR should be Interval	√	
Non normalised <tbl1>	NOR = normalise [days] (N)	[days] of NOR should be Interval	√	
<tbl1> is empty	NOR = normalise [date] (E)	Normal Situation	√	

Project

<tbl3> = **project [**<atr>**] (**<tbl1>**)**

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> does not exist	P = project [date] (R)	<error message>	√	
<tbl3> exists	A = project [date] (O)	<error message>	√	
<atr> not present	P = project [] (R)	<error message>	√	
Valid Syntax and Specifications Satisfaction				
attribute casting	P=project [newname = days] (s)	P has an attribute with name 'newname'	√	
<tbl1> with duplicates, time point	P= project [date] (L)	P has no duplicates	√	
<tbl1> with duplicates, time interval	P= project [days] (D)	P has no duplicates	√	
<atr> is not point or interval	P = project [name] (S)	Normal Situation	√	
<atr> is time interval	P = project [days] (S)	Normal Situation	√	
<atr> is time point	P = project [date] (O)	Normal Situation	√	
Non normalised <tbl1>	P = project [days] (N)	Normal Situation	√	
<tbl1> is empty	P = project [date] (E)	Normal Situation	√	
<atr> = *	P = project [*] (O)	All atrs are processed	√	
Support of SQL functions	P = project [abs(-2)] (S)	Normal Situation	√	
Support of VT-RA and SQL functions				
topoint in <atr>, atr non unary interval	P = project [topoint(days)] (S)	Normal Situation	√	
tointerv in <atr>, atr not time point	P = project [tointerv(days)] (S)	Normal Situation	√	
topoint in <atr>	P = project [topoint(period)] (I)	Normal Situation	√	
tointerv in <atr>	P = project [tointerv(c(date))] (O)	Normal Situation	√	

Product

<tbl3> = <tbl2> **product** <tbl1>

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> or <tbl2> does not exist	P = S product R P = R product S	<error message>	√	
<tbl3> exists	A = O product L	<error message>	√	
<tbl1>,<tbl2> have attrs with the same name	P = S product A	<error message>	√	
Valid Syntax and Specifications Satisfaction				
<tbl1> or <tbl2> or both are empty	P = S2 product E P = E product S2 P = E product E	Normal Situation	√	
<tbl1>, <tbl2> have not attrs with the same name	P = S product I	Normal Situation	√	

Union

<tbl3> = <tbl2> **union** <tbl1>

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> or <tbl2> does not exist	U = S union R U = R union S	<error message>	√	
<tbl3> exists	A = O union L	<error message>	√	
<tbl1>,<tbl2> have different no. of attrs	U = S union A	<error message>	√	
<tbl1>,<tbl2> have type incompatible attrs	U = S union O	<error message>	√	
Valid Syntax and Specifications Satisfaction				
<tbl1> or <tbl2> with duplicate tuples	U = S union D U = D union S U = O union L U = L union O	Normal Situation	√	
tables with different attr names, type compatible	U = S union S2 U = S2 union S	Normal Situation	√	
<tbl1> or <tbl2> or both are empty	U = S union E U = E union S U = E union E	Normal Situation	√	

Except

<tbl3> = <tbl2> **except** <tbl1>

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> or <tbl2> does not exist	EX = S except R EX = R except S	<error message>	√	
<tbl3> exists	A = O except L	<error message>	√	
<tbl1>,<tbl2> have different no. of attrs	EX = S except A	<error message>	√	
<tbl1>,<tbl2> have type incompatible attrs	EX = S except O	<error message>	√	
Valid Syntax and Speifications Staisfaction				
<tbl1> or <tbl2> or both with duplicate tuples	EX = S except D EX = D except S EX = O excpet L EX = L except O EX = D except D2 EX = D2 except D	EX with no duplicates	√	
<tbl1> or <tbl2> non normalised	EX = A except N EX = N except A	EX normalised	√	
<tbl1> and <tbl2> with different attr names	EX = S except S2 EX = S2 except S	EX with <tbl1> schema	√	
<tbl1> or <tbl2> or both are empty	EX = S2 except E EX = E except S2 EX = E except E	EX is empty	√	

Punion

<tbl3> = <tbl2> **punion [**<atr>**]** <tbl1>

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> or <tbl2> does not exist	PU = S punion [days] R PU = R punion [days] S	<error message>	√	
<tbl3> exists	A = O punion [date] L	<error message>	√	
empty <attr>	PU = S punion [] (S)	<error message>	√	
<tbl1>,<tbl2> have different no. of attrs	PU = S punion [days] A	<error message>	√	
<tbl1>,<tbl2> have type incompatible attrs	PU = S punion [days] O	<error message>	√	
<atr> belongs to <tbl1>	PU = S punion [date] O	<error message>	√	
<atr> not time point or time interval	PU = S punion [name] D	<error message>	√	
Valid Syntax and Specifications Staisfaction				
<tbl1> or <tbl2> with duplicate tuples	PU = S punion [days] D PU = D punion [days] S PU = O punion [date] L PU = L punion [date] O	Duplicates removed	√	
<atr> is time point	PU = O punion [date] O	PU has time interval attr	√	
<atr> is time interval	PU = S punion [days] S	Normal Situation	√	
<tbl1> or <tbl2> non normalised	PU = A punion [days] N PU = N punion [days] A	PU should be normalised	√	
<tbl1> and <tbl2> have different atr names	PU = S punion [days] S2 PU = S2 punion [date] S	Normal Situation	√	
<tbl1> or <tbl2> or both empty	PU = S punion [days] E PU = E punion [date] S PU = E punion [date] E	Normal Situation	√	

Pexcept

<tbl3> = <tbl2> **pexcept [**<atr>**]** <tbl1>

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> or <tbl2> does not exist	PE = S pexcept [days] R PE = R pexcept [days] S	<error message>	√	
<tbl3> exists	A = O pexcept [date] L	<error message>	√	
empty <attr>	PE = S pexcept [] D	<error message>	√	
<tbl1>,<tbl2> have different no. of attrs	PE = S pexcept [days] A PE = A pexcept [days] S	<error message>	√	
<tbl1>,<tbl2> have type incompatible attrs	PE = S pexcept [days] O PE = O pexcept [days] S	<error message>	√	
<atr> belongs to <tbl1>	PE = S pexcept [date] O	<error message>	√	
<atr> not time point or time interval	PE = S pexcept [name] D	<error message>	√	
<tbl1> and <tbl2> have different attr names	PE = S pexcept [days] S2 PE = S2 pexcept [date] S	<error message>	√	
Valid Input and Specifications Staisfaction				
duplicate tuples, time interval	PE = S pexcept [days] D PE = D pexcept [days] S	Duplicates removed	√	
duplicate tuples, time point	PE = O pexcept [date] L PE = L pexcept [date] O	time interval attr, duplicates removed	√	
<tbl1> or <tbl2> non normalised	PE = N pexcept [days] A PE = A pexcept [days] N	PE should be normalised	√	
<tbl1> or <tbl2> or both are empty	PE = S2 pexcept [days] E PE = E pexcept [date] S2 PE = E pexcept [date] E	Normal Situation	√	

Select

<tbl3> = **select [**<srch>**] (**<tbl1>**)**

Test Case Descr.	Test Case Command	Expected Result	Pass	Comments
Syntax Errors	('=' or '[' or ']' or '(' or ')' or <tbl1> is missing)	<error message>	√	
Specifications Violation				
<tbl1> does not exist	SE = select [patient='john'] (R)	<error message>	√	
<tbl3> exists	A = select [patient='john'] (C)	<error message>	√	
Valid Input and Specifications Satisfaction				
<srch> is empty	SE = select [] (C)	Selects all tuples	√	
<tbl1> is empty	SE = select [] (e)		√	
<srch> contains AND	SE = select [patient='john' AND period = interv('1993-06-08','1993-06-15')] (C) SE = select [complication='dialysis' AND (patient='john' OR interv('1993-06-08','1993-06-15')] (C)		√	
<srch> contains *OR*	SE = select [patient='john' OR period = interv('1993-06-08','1993-06-15')] (C)		√	
<srch> contains *NOT*	SE = select [NOT (period cp interv('1993-06-08','1993-06-15'))] (C)		√	
<srch> contains a combination of AND, OR and NOT	SE = select [(patient='john' AND period = interv('1993-06-08','1993-06-15')) OR NOT (period after tointerv('1993-06-08'))] (C)		√	
Support of SQL functions	SE = select [patient='john' OR abs(-2)=2] (S)		√	
Support of VT-RA and SQL functions	SE = select [patient='john' OR period after tointerv('1993-06-08')] (C)		√	

6. Conclusions

From the evaluation tables provided above, it can be verified that major effort was made for the tests to be exhaustive. The results which have been obtained are completely satisfactory, since no undesirbale side-effect has been identified.