Function Reference

1 DataTypes

“”,$[]	String
‘’	Variable name
0x12	Hexadecimal long integer
1L	Long integer
\s	A character that invokes an alternative interpretation on special characters in a string
date()	Convert a string or an integer to a date
datetime()	Convert a string or a long integer to a datetime
decimal()	Convert to big decimal number
float()	Convert to double-precision floating-point number
false	False value
ifdate()	Judge if the parameter is a date or a datetime
ifnumber()	Judge if the parameter is a of numeric data type
ifstring()	Judge if the parameter is a string
iftime()	Judge if the parameter is of time data type
int()	Convert to integer
long()	Convert to long integer
number()	Convert to real number
null	Null value
string()	Convert to string
time()	Convert a string or an integer to a time
true	True value

2 Operators

${ }	Macro
%,\	Get the remainder and the integer value
&	Generate a new sequence by merging members of two sequences (or two single values) in a certain order; common members appear only once
&&,||,!	Logic operation
$(x_1,x_2,\ldots,x_k)$	Compute a series of expressions one by one, and return the result of the last expression; the expression is allowed to reference the variable value computed by the previous expression for assigning values
*	Generate a new sequence by duplicating members of a sequence
++,--,**,//,%%,\\	Generate a new sequence by performing Alignment Arithmetic Operations between two sequences which are of the same length, such as aligning addition, aligning subtraction, aligning multiplication and seeking remainder and aliquot part
+,-,*,/	Signs of the four arithmetic operations
-a	Opposite number
==,!=,<,>,<=,>=	Comparison Operation

\	Generate a new sequence by removing members (or the single value) of sequence B from sequence A
^	Return a new sequence which is composed of common members from two sequences
a=x	Assign the result of an expression to a variable and return the result of the expression
a?=x	Compound assignment computation
case()	According to the various results of computing different expressions, return various values
cmp()	Compare the values of two expressions or two sequences
eq()	Judge if a sequence can be generated by swapping the positions of the members of another sequence
eval()	Dynamically parse and compute the expression
f@o(…)	Introduce the common rules of functions, that is, compute f(···) with option @o in this case
if()	According to the results of a boolean expression, return different values
in()	Judge if Parameter 1 is between Parameter 2 and Parameter 3
s_1+s_2	Concatenate two or more strings end-to-end
|	Concatenate members (or single values) of two sequences so as to generate a new sequence

3 Mathematics

abs()	Absolute value
acos()	Arc cosine value
asin()	Arc sine value
atan()	Arc tangent value
ceil()	Truncate data at the specified position, and carry the remaining part if any
cos()	Cosine value
exp()	*e* to the power of *n*
fact()	Factorial
floor()	Truncate data at the specified position, and reject all the remaining part if any
lg()	Logarithm with 10 as the base
ln()	Natural logarithm
pi()	The ratio of the circumference of a circle to its diameter and its multiple
power()	*x* to the power of *n*
rand()	Random value
rgb()	Convert the red, green, blue, and transparency value to the corresponding color values
round()	Truncate data at the specified position, and round off the remaining part
sign()	Judge whether the parameter is a positive or negative number or a 0
sin()	Sine value

- sqrt() Square root
- tan() Tangent value

4 Strings

- *A*.regex() Match a string with a regular expression
- *A*.string() Join all the members of a sequence with a delimiter
- asc() Get the Unicode value of the character at the specified position in a string; if it is an ASCII character, then return its ASCII code
- char() According to the given Unicode or ASCII code, get the corresponding characters
- fill() Create a string by concatenating multiple strings
- isalpha() Judge if the first character of a string is a letter
- isdigit() Judge if the first character of a string is a number
- islower() Judge if the first character of a string is in lower case
- isupper() Judge if the first character of a string is in upper case
- left() Get the substring to the left of a source string
- len() Compute the length of a string
- like() Judge if a string matches a format string
- lower() Convert all characters of a string to lower case
- mid() Return the substring of a string
- pad() Pad another string before a string
- pos() Search the position of a substring in a parent string, and return null if not found
- *r*.string() Convert all the fields that can be transformed into the texts in *r* to strings, with commas as the delimiters between field values
- rands() Get the random string
- replace() Change the substring of a source string
- right() Get the substring to the right of a source string
- *s*.array() Split a string *s* by delimiter so as to form a sequence and return it
- *s*.regex() Match the string members of a sequence with the regular expression
- *s*.words() Select the English words out of a string
- string() Convert the object to the character type. Formatting is allowed during the process of conversion
- trim() Remove the space character on both ends of a string
- upper() Convert all characters in a string to upper case

5 Datetime

- after() Compute the new date which is certain days after a date
- age() Compute the number of whole years between a specified date and the current time
- date() Convert a string or an integer to a date

- date(*datetimeExp*) Get the date part of the datetime value
- datetime() Convert the string or long integer to datetime
- datetime(*datetimeExp*) Adjust the precision of datetime expression and then return the expression
- day() Get the day from a specified date
- days() Get the number of days of the year, quarter or month to which the specified date belongs
- deq() Judge if two dates are the same, accurate to the day
- hour() Get the hour from a specified datetime
- interval() Compute the interval between two datetime values
- millisecond() Get the millisecond from a specified datetime
- minute() Get the minute from a specified datetime
- month() Get the month from a specified date/datetime
- now() Get the current system date and time
- pdate() Get the first and the last day of the week/month/quarter to which a specified date belongs
- periods() Generate a new sequence composed of datetimes according to specified interval
- second() Get the second from a specified datetime
- time() Convert the string or integer to time
- time(*datetimeExp*) Get the time part of the datetime value
- workday(*t,k,h*) Compute a datetime after *n* workdays of the specified date
- year() Get the year from a date

6 Sequence

- *A*(*i*) Get a member from a sequence
- *A*(*i*)=*x* Assign values to members of a sequence
- *A*(*p*) Get members from a sequence according to the sequence numbers which are the members of an integer sequence, so as to create a new sequence
- *A*(*p*)=*X* Assign the data from sequence *X* to the members of the integer sequence *p*, which consists of the sequence numbers of members of sequence *A*, in alignment
- *A*(*p*)=*x* Assign *x* to all members of the integer sequence *p* that is composed of the sequence numbers of members of sequence *A*
- *A*.delete() Delete specified members from a sequence
- *A*.dup() Duplicate a sequence
- *A*.insert() Insert members into a sequence
- *A*.len() Get the length of a sequence
- *A*.m() Get members of a sequence at specified positions
- *A*.modify() Assign values to one or more members of a sequence according to the specified position(s)
- *A*.p() Get sequence numbers of the members at the specified positions`
- *A*.step() Get members from a sequence according to specified start position and step,

so as to create a new sequence

- *A*.to() Get members starting from a specified position from a sequence, so as to create a new sequence
- [] Define a sequence having no members
- $[a_1,\ldots,a_n]$ Sequence constants
- ifa() Judge if an object is a sequence
- to() Generate a sequence composed of a series of consecutive integers

7 Table Sequence

- *A*.run(x_i:F_i,…) Compute expressions with each member of a sequence
- *A*.switch() Switch between the key value of the coding field and the indictor record
- *P*.paste() Modify records in batches
- *T*.append() Append in order the records of one table sequence to another
- *T*.create() Generate a new empty TSeq by duplicating the data structure of TSeq *T*
- *T*.delete() Delete specified records from a table sequence
- *T*.dup@t() Generate a new table sequence by duplicating the data structure as well as all the records of a table sequence
- *T*.fname() Get the names of fields of the TSeq
- *T*.fno() Get the serial number(s) of a single or multiple specified fields in a table sequence
- *T*.index() Create a hash index for the primary key of a TSeq
- *T*.insert() Insert one or more records into a table sequence
- *T*.modify() Modify one or more records of a table sequence
- *T*.primary() Set the primary key for a table sequence
- *T*.record() Use the members of sequence *A* to form new records of TSeq *T*
- *T*.reset() Clear the members of a table sequence
- *T*.rename() Modify field names
- create() Create an empty table sequence
- ifr() Judge if an object is a record
- ift() Judge if an object is a table sequence
- *r*.*F* Get value for field *F* of record *r*
- *r*.*F*=x Assign *x* to field *F* of record *r*
- *r*.array() Get the field values from a record respectively and return them as a sequence
- *r*.field() Get the value of a certain field in a record
- *r*.field(*i*,*x*) Modify values in the specified field of record *r*
- *r*.fname() Get the name of a field of the record according to the serial number of the field
- *r*.fno() Get the serial number of field *F* of record *r* in a table sequence
- *r*.modify() Modify the field values of a record
- *r*.paste() Modify the specified records
- *r*.pkey() Get the field value of the primary key of a record
- *r*.record() Make members of sequence *A* the new field values of record *r*

- *r*.run() Compute an expression based on a record and return the record itself
- *v*.v() Get the primary key value of a referencing field
- *r*.prior() Among the records, query fields referred by the foreign key recursively

8 Loop Functions

- *A*.(*x*) Compute an expression with each member of a sequence, and return the results
- *A*.avg() Compute the average value of the non-null members in a sequence
- *A*.avg(x) Compute x with each member of the sequence and then compute the average value of the non-null members of the new sequence
- *A*.avgif(A_i:x_i,…) Get the intersection of positions of the members in a sequence, and return the average of the members
- *A*.calc() Compute an expression with one or more specified records and return the results
- *A*.conj() Concatenate all the members in a sequence whose members are sequences
- *A*.conj(x) Compute x with each member of the sequence whose members are sequences, and then concatenate the computed results
- *A*.count() Count the number of non-null members in a sequence
- *A*.count(x) Compute x with each member of the sequence and then count the number of non-null sequence members of the new sequence
- *A*.countif(A_i:x_i,…) Get the intersection of sequences, and return the number of non-null ones of the common members in sequence *A*
- *A*.diff() Compute difference between members of a sequence whose members are sequences
- *A*.diff(x) Compute x with each member of the sequence whose members are sequences, and then compute difference between members of the new sequence
- *A*.f(x) The common rules of loop functions
- *A*.field() Get the value of the i^{th} field of each record of a sequence by loop
- *A*.field(*i*,*x*) Get the modified value of the *i*th field in a sequence
- *A*.find(*v*) Find a record by its primary key value
- *A*.ifn() Get the first non-null member from a sequence
- *A*.ifn(x) Compute x with each member of the sequence and return the first non-null member of the new sequence
- *A*.in(*B*) Judge if a sequence contains another sequence
- *A*.inv() Adjust the order of membes of a sequence
- *A*.isect() Compute the intersection of the member sequences of a sequence
- *A*.isect(x) Compute x with each member of the sequence whose members are sequences, and then compute intersection of members of the new sequence
- *A*.lookup() Locate positions of members in a sequence, get the intersection of the positions and return the members in these positions in sequence *A*
- *A*.loop(*x*;*a*;*c*) Iterative loop of an RSeq
- *A*.loops(*x*;*a*;*c*) Perform the iterative loop over an RSeq and return the result of the last

running of *x*

- A.max() Compute the maximum value of all the non-null members in a sequence
- A.max(x) Compute x with each member of the sequence and then compute the maximum value of the members of the new sequence
- A.maxif(A_i:x_i,…) Get the intersection of the sequences of postitions and return the maximum non-null value of members in the intersection positions in sequence *A*
- A.maxp() Return the member that makes the maximum value of the expression
- A.merge() Merge operation, which refers to merging all members of a sequence
- A.min() Compute the minimum value of all the non-null members in a sequence
- A.min(x) Compute x with each member of the sequence and then compute the minimum value of the members of the new sequence
- A.minif(A_i:x_i,…) Get the intersection of the sequences of positions and return the minimum non-null value of members in the intersection positions in sequence *A*
- A.minp() Return the member that makes the minimum value of the expression
- A.pfind() Find the sequence number of a record by its primary key
- A.pmax() Return the sequence number of the member of the maximum value in the sequence
- A.pmin() Return the sequence number of the member of the minimum value in the sequence
- A.pos() Get the position of a member in a sequence
- A.pos(*x*) Get the position of a sequence member in another sequence
- A.pselect() Return the sequence numbers of members that satisfy the query criterion
- A.psort() Return the sequence numbers of the sorted members in the original sequence
- A.ptop() Get sequence numbers of top *n* smallest members of the sequence
- A.rank() Compute the ranking of each member in a sequence
- A.rank(*x*) Get the ranking of sequence *A*.(*x*)
- A.ranki(*y*) Compute the ranking of a value in a sequence
- A.ranki(*y*,*x*) Get the ranking of a certain sequence member after the sequence is computed
- A.pseg(*x*) Return the position of a member in a sequence
- A.run(x_1,x_2,…x_i) Compute the expressions with each member in a sequence and return the modified sequence
- A.rvs() Generate a new sequence by reversing the members in a sequence
- A.select() Select members from a sequence which satisfies a condition and return members that make the value of the expression true
- A.sort() Generate a new sequence by sorting the members of a sequence and return the new sequence
- A.sum() Compute the summary value of members of a sequence
- A.sum(x) Compute x with each member of the sequence and compute the summary value of the members of the new sequence
- A.sumif(A_i:x_i,…) Get the intersection of sequences and return the sum of the non-null ones of the common members in sequence *A*
- A.swap() Generate a new sequence by swapping the member positions of two specified

intervals of a sequence

- A.top() Get the top n smallest records of the sequence members
- A.topx() Get the top n smallest values of a sequence
- A.union() Compute the union of members in a sequence whose members are sequences
- A.union(x) Compute x with each member of the sequence whose members are sequences, and then compute union of members of the new sequence
- A.variance() Compute the variance value of the non-null members in a sequence
- A.variance(x) Compute x with each member of the sequence and then compute the variance value of the members of the new sequence
- *n*.f(*x*) Perform a loop operation using an integer as the loop variable
- *p*.inv(*k*) Compute the sequence numbers of memerbers of an ISeq in another ISeq
- *r*.(*x*) Compute an expression with a record and return the result
- ~,#,F,A[*i*]... Introduce the common rules of expressions in a loop function

9 Relational Functions

- A.derive() Add one or more fields to a TSeq/RSeq
- A.group(*xi*,…) Perform equal grouping according to x_i,…
- A.group(*x*:*F*;*y*:*G*…) Group a sequence and then perform aggregate operations
- A.groups() Group a Tseq and then get the aggregating result cumulatively
- A.id() Perform merging and sorting, then the *distinct* operation on a sequence, and return the result of *distinct*
- A.new(x_i:F_i,…) Generate a new table sequence, of which the number of records is the same as that of sequence *A*
- A.rank@i(*x*) Remove the duplicate member of *A*.(*x*) first, then obtain its ranking
- A.ranki@i(*y*,*x*) Remove the duplicate member of *A*.(*x*) first, then obtain the ranking of *y* in *A*.(*x*)
- E.penum() Judge to which enum group of an RSeq a field value or an expression belongs, and return the serial number(s) of the group(s) that is (are) satisfied the grouping criterion
- P.align() Align the records of an RSeq to another sequence or Rseq by the relational fields and then sort them, so as to group the RSeq
- P.enum() Generate a new RSeq by performing enumeration grouping on an RSeq according to another RSeq of conditional expressions
- join() Join multiple sequences together
- pjoin() Join multiple sequences together by sequence numbers
- xjoin() Cross join multiple sequences together

10 File

- S.import() Retrieve contents from a string as records and return them in the form of TSeq
- directory() List the file name(s) satisfying a wildcard path
- export() Return a sequence as the string

- f.date() Return the time and date of a file last modified
- f.exists() Verify the existence of a file
- f.export() Write a sequence into a file
- f.import() Read each row from a file object as a record, so as to form a table sequence and return it
- movefile() Move, delete, or rename a file
- f.property() Read the property value from the property file
- f.read() Read contents from a file object as strings and return them
- f.size() Return the length of a file
- f.write() Write a string or a string sequence into a file
- file() Open a file object with the specified file name
- filename() Split up the full path, get the file name and suffix

11 Database

- connect() Create a connection to a datasource
- db.close() Close a database connection
- db.commit() Commit the database transaction manually
- db.error() Obtain the last error information from the database connection
- db.execute() Execute SQL statements within a specified database connection
- db.proc() Call a database stored procedure
- db.query() Execute the SQL statement within a database connection and return the query results
- db.rollback() Roll back the database transaction
- db.update() Update database tables

12 Cursor

- CS.conj@x() Concatenate the members of a cursor sequence and return the result as a cursor
- CS.merge@x() Merge the cursor members of a sequence
- CS.pjoin() Join the cursor sequence and return the result as a cursor
- F.cursor() Create and return a cursor based on the sequence of binary file objects
- F.export() Write the cursor fields into the files in *F* respectively, with one file for one field
- P.cursor() Convert an in-memory RSeq to a cursor and return it
- cs.close() Close a cursor directly
- cs.conj() Split each record in a cursor into a TSeq or an RSeq, and return a cursor of the union set of the splitting results
- cs.derive() Add one or more fields to a cursor
- cs.fetch() Fetch one or more records from a cursor
- cs. groupn() Group cursor records by the group sequence number and return a cursor sequence
- cs.groups() Group the records in the cursor

- *cs*.groupx() Group the ordered records in the cursor and return result as a cursor
- *cs*.join() Join a cursor with an RSeq or another cursor through the foreign key
- *cs*.new() Generate a new cursor by computing the field values of the existing cursor
- *cs*.regex() Match the string members in a cursor with a regular expression
- *cs*.run() Compute the expresion with the records in a cursor
- *cs*.select() Generate a new cursor after filtering the records in the existing cursor
- *cs*.skip() Skip records while fetching records from a database cursor, and return the number of skipped records
- *cs*.sortx() Sort a cursor
- *cs*.switch() Switch between the key value of coding field and the indicator record in a cursor
- *db*.cursor() Create a database cursor by executing an SQL statement and return it
- *f*.cursor() Create a cursor according to a file and return it
- *f*.export() Retrieve data from the cursor *cs* and write them to a file
- *f*.icursor() Use the index file to retrieve the file content
- f.ncursor() Use the index file to retrieve file content specified by an interval of field values
- *f*.index() Generate an index file for a file
- *fi*.icount() Return the total number of records in an index file
- join@x() Join the sequence corresponding to a cursor

13 Statements

- #c The variable generated during the process of loop
- $(db)sql;… Execute the specified SQL statement on the datasource and return the result set
- break {a} Break the current loop
- call() Call the cellset file and return the first result set
- clear Clear the values of certain cells
- end s Log the error message
- for cs,n;x Loop through the cursor
- for *x* Start a loop
- for a,b,s Execute a loop according to the specified scope and span
- fork Execute the code block in the current cellset using multithreads
- func …{return x_i} Define the function block of a function
- func() Call a subroutine
- if *x* If statement
- if *x* else If statement
- if *x* …{ else { if *x*}… } If statement
- if *x*…else … If statement
- next {a} Skip the current loop and continue with the next loop
- pcursor(*dfx*,…) Generate a cursor using a program
- result x_i Return the result x_i,…,

14 System

- @v Define a global variable starting with @; maintain control synchronously if changing the variable table.
- ifv() Judge if a variable exists
- output() Output the printed out data to the console
- rmv() Remove variables
- system() Execute the system command

15 Parallel

- callx() Perform the parallel computing
- *cs*.joinx() Join cursor to a segmental dimension table
- dims() Generate a piecewise dimension table space
- directory(*path*,*z*) List the file names matching the wildcard path name in the Data section
- ds.fetch() Get values for certain records fetched from the dimension table
- ds.gf() In the dimension table space *ds*, append dimension table *V* and its segmental function *g*
- file(*fn*:*cs*,*z*,*h*) Open the file object of the remote file with the specified name
- movefile() Copy, move, delete or rename a file across partitions
- hosts() Find the idle ones among the nodes
- sync() Node machine synchronization

16 Chart

- G.draw() Draw on a canvas
- G.plot() Compute the chart plotting string and its parameters, plot them onto the canvas and return the result
- canvas() Return the canvas object

17 Users

- >statement With esProc JDBC, generate the contextual execute statement and return no result set once the statement is executed
- =expression With esProc JDBC, execute the statement directly and return the result set
- $(db)sql;… With esProc JDBC, execute the specified SQL statement and return the result set
- call path/dfx(…) With esProc JDBC, search for and execute a program file locally; if not found, then search the server
- dfx … With esProc JDBC, search for and execute a program file locally; if not found, then search the server
- *f*.exportxls() Write the sequence into an Excel file
- *f*.importxls() Retrieve contents as records from an Excel file object, and return the result in the form of a TSeq
- hdfsfile() Return HDFS file flow

- httpfile() Package the returned result of URL as file flow and return it
- invoke() Invoke the static function of class in the package
- *mdb*.aggregate() Perform aggregate query on a MongoDB and return the result as a table sequence
- *mdb*.count() Query the data of a MongoDB, count them and return the result in the form of a numerical value
- *mdb*.close() Close the connection to MongoDB
- *mdb*.distinct() Query the data of a MongoDB, filter them and return the result in the form of a sequence
- *mdb*.find() Query the data of a MongoDB and return the result in the form of a cursor
- mongodb() Create a connection to MongoDB

Function

#c

Description:

The variable generated during the process of loop.

Remark:

c The cell where *for* statement is placed

#*c* The sequence number of the loop

Example:

	A	B	C	
1	=demo.query("select * from EMPLOYEE")			[1,2,4,5,7,10,11]
2	=[]			
3	for A1			In the process of loop, insert all departments' sequence numbers of loops into sequence **A2,** except for **Sales**
4		if A3.DEPT=="Sales"		
5			next	
6		>A2=A2|[#A3]		

$(*db*)sql;…

$(*db*)sql;…

Description:

Execute the specified SQL statement on the datasource and return the result set.

Syntax:

$(*db***)***sql*;**…**

Remark:

On the datasource *db*, execute the specified *SQL* and return the execution result. *db* is the database connection. If omitting **(***db***)**, then use the datasource specified by the previous statement by default. If the datasource is not specified beforehand, then use any one of the connected datasources.

Options:

@1 Use this option if the result set is a single value

Parameters:

sql A SQL statement, like **select * from table**; here the SQL must be the select/insert/delete/update statement

(db) Datasource name

… SQL parameter’s value

Return value:

A table sequence composed of results of SQL.

Example:

- SELECT statement

	A	
1	$select * from EMPLOYEE	Error:" Missing the database connection factory "
2	$(demo)select * from EMPLOYEE where EID=?;1	Find the information of employee whose EID is 1
3	$select * from EMPLOYEE where EID in (?) or GENDER=?;[1,3,5,7],"M"	Find the employees whose EID is [1,3,5,7] and gender is M

- INSERT statement

	A	B
1	$(demo)insert into EMPLOYEE (EID, NAME) values(?,?);100,"test"	
2	[51,52,53,54]	
3	for A2	
4		$insert into STATECAPITAL (STATEID) values(?);A3

- DELETE statement

	A
1	$(demo)delete from EMPLOYEE where EID =? or EID=?;100,101
2	$delete from EMPLOYEE where EID in(?);[1,5,7,9]
3	$delete from EMPLOYEE where NAME ='Rebecca'

- UPDATE statement

	A	
1	$(demo)update EMPLOYEE set NAME =?, GENDER=? where EID =?;"testnew","M",100	
2	$update EMPLOYEE set NAME ='Peter' where EID =10	
3	$(sql)update Family set Name='Rose' where Eid=?;2	Use SQL datasource to update the name of a family member whose Eid is 2

Related concepts:

db.query()

$(*db*)sql;…

Description:

With esProc JDBC, execute the specified SQL statement in the database and return the result set.

Syntax:

$(*db*)*sql*;...

Remark:

In the specified database *db*, execute the SQL statement, with the parameters represented by … after semicolons, and return the result set once executed. Use ***st*.executeQuery()** for the execution and return the result set. Make sure the database *db* must be connected.

Parameters :

sql A SQL statement, like **select * from table**; SQL must be the select/insert/delete/update statement

(*db*) Datasource name

… SQL parameter's value

Return value:

Result set

Example:

```java
public void testDataServer() {
        Connection con = null;
        com.esproc.jdbc.InternalCStatement st;
        try{
            Class.forName("com.esproc.jdbc.InternalDriver");
            con= DriverManager.getConnection("jdbc:esproc:local://");
            st=(com.esproc.jdbc.InternalCStatement)con.createStatement();
            // Query on demo database student table to find the data of students who are order than 16
            ResultSet set = st.executeQuery("$(demo)select * from STUDENTS where AGE>?;16");
            printRs(set);
        }
        catch(Exception e){
            System.out.println(e);
        }
        finally{
            // Close the connection
            if (con!=null) {
                try {
                    con.close();
                }
                catch(Exception e) {
                    System.out.println(e);
                }
            }
        }
        }
```

${ *macroExp* **}**

Description:

This is used to complete the macro replacement operation.

Syntax:

${ *macroExp***}**

Remark:

The *macroExp* here is taken as an expression to be computed, the computation must be a string, and then the result will replace the **${** *macroExp***}**.

The macro enclosed in quotation marks or populated in the constant cell will not be replaced.

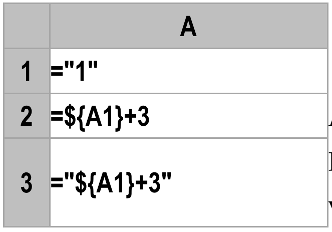
Parameters:

macroExp	The macro expression, the computation of which will replace the **${***macroExp***}**, so the result must be a string.

Return value:

String

Example:



	A	
1	="1"	
2	=${A1}+3	After replacing, the expression becomes **=1+3**, so the return value is **4**.
3	="${A1}+3"	Macro is enclosed in quotation marks, so it will not be replaced, and the return value is still "**${A1}+3**".

=expression

Description:

With the esProc JDBC, execute the statement directly, and return the result set

Syntax:

=*expression*

Remark:

Such statement works similarly to the formulas in the esProc computational cell. The Statement can be generated by directly using ***con*.createStatement()**. In executing, use ***st*.executeQuery()** to execute the statement directly, and return the result set. Please notice that the expression must start with the equal sign.

Parameter:

expression	SQL statement or formula. If there are multiple formulas in the bracket, then it indicates that the formulas will be computed one by one and the value of the last formula will be returned, for example: (arg1=1,arg2=2,arg1+arg2), the last return value is 3

Return value:

Result set

Example:

```
public void testDataServer() {
        Connection con = null;
        com.esproc.jdbc.InternalCStatement st;
        com.esproc.jdbc.InternalCStatement st2;
        try{
            Class.forName("com.esproc.jdbc.InternalDriver");
            con=DriverManager.getConnection("jdbc:esproc:local://");
            // Call stored procedure
            st=(com.esproc.jdbc.InternalCStatement)con.prepareCall("call test(3)");
            // Execute statement directly, return result set
            ResultSet set = st.executeQuery("=(arg=3,arg+3)");
            // Print results
            printRs(set);
            // Create Statement directly
            st2=(com.esproc.jdbc.InternalCStatement)con.createStatement();

            ResultSet set2 = st2.executeQuery("=(arg=3,arg+4)");
            printRs(set2);
        }
        catch(Exception e){
            System.out.println(e);
        }
        finally{
            // Close the connection
            if (con!=null) {
                try {
                    con.close();
                }
                catch(Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
```

>statement

Description:

With esProc JDBC, generate the contextual execution statement, and return no result set once executed

Syntax:

>*statement*

Remark:

Such statement works similarly to the formulas in the esProc execution cell. There **will be no result**

set returned once executed. In the esProc JDBC, the Statement can be generated by directly using ***con*.createStatement()**. In executing, use ***st*.execute()** to execute the statement directly.

Parameter:

statement Statement or clause

Example:

```
public void testDataServer() {
        Connection con = null;
        com.esproc.jdbc.InternalCStatement st;
        try{
            Class.forName("com.esproc.jdbc.InternalDriver");
            con= DriverManager.getConnection("jdbc:esproc:local://");
            // Create Statement directly
            st=(com.esproc.jdbc.InternalCStatement)con.createStatement();
            // Execute statement directly, and operate on the specified data set
            boolean b =st.execute(">demo.execute(\"delete from STUDENTS where ID =
1\")");
            // Print execution result
            System.out.println(b);
        }
        catch(Exception e){
            System.out.println(e);
        }
        finally{
            // Close the connection
            if (con!=null) {
                try {
                    con.close();
                }
                catch(Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
```

@v

Description:

Define a global variable starting with @. Maintain synchronously if changing the variable table.

Syntax:

@*v*

Parameters:

v The global variable name

Example:

	A	
1	>@name="tiger"	To begin with @ while defining
2	=@name	"tiger", to begin with @ while using it

A()

A (*i*)

Description:

Get a member from a sequence.

Syntax:

A (*i*)

Remark:

Get the *i*th member from the sequence *A*.

Parameters:

A	Sequence object
i	Sequence number expression of the member, which starts from 1

Return value:

The member value of a specified sequence number in the sequence A

Example:

	A	
1	=[1,2,4]	
2	=A1(3)	4
3	=["a","b"](1)	"a"

Related concepts:

A (p)

A (*p*)

Description:

Get members from a sequence according to the sequence numbers which are the members of an integer sequence, so as to create a new sequence

Syntax:

A(*p*)

Remark:

A is an *n* sequence, and *p* is an *n* integer sequence whose length is *m*. Get the members in *p* in turn, and use the member values of them as the sequence numbers to get the members in *A* to generate a new sequence.

Parameters:

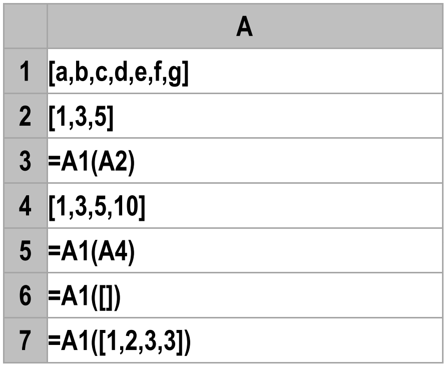
p The *n* integer sequence whose length is *m* and member values are larger than 0 and less than or equal to the length of *A*. For example, if the length of sequence *A* is 5, then the members in *P* must be integers larger than 0 and less than or equal to 5. If *P* is an empty sequence, then return an empty sequence.

A A sequence whose length is *n*

Return value:

A new sequence whose length is *m*

Example:



	A	
1	[a,b,c,d,e,f,g]	
2	[1,3,5]	
3	=A1(A2)	[a,c,e]
4	[1,3,5,10]	
5	=A1(A4)	Report errors, as the index exceeds the boundary.
6	=A1([])	[]
7	=A1([1,2,3,3])	[a,b,c,c]

Related concepts:

A (i)

A.()

A.(*x*)

Description:

Compute an expression with each member of a sequence.

Syntax:

A.(*x*)

A.() return *A* itself

Remark:

Generate a new sequence composed of the results of computing expression *x* with each member in sequence or record sequence *A*. Use "**~**" in *x* to reference the current member in *A*.

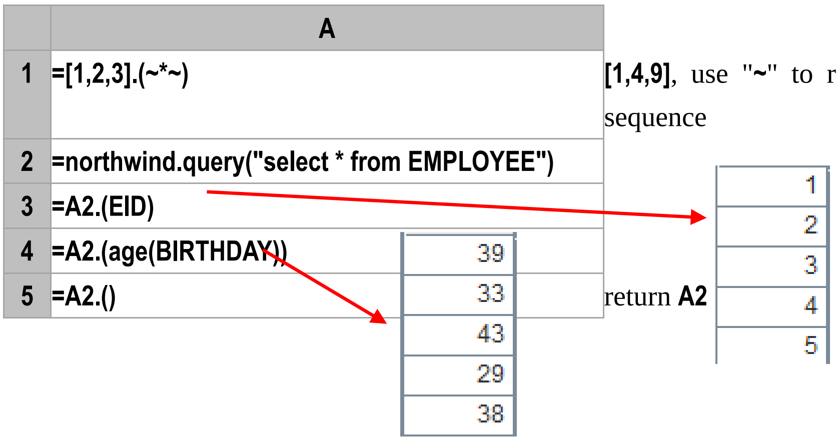
Parameters:

A	A sequence/a record sequence
x	An expression, which is generally a field name or a legal expression composed of field names and in which "**~**" is used to reference the current record.

Return value:

A new sequence composed of the results of computing expression *x* with each member in *A*

Example:



	A	
1	=[1,2,3].(~*~)	[1,4,9], use "~" to reference the current member of the sequence
2	=northwind.query("select * from EMPLOYEE")	
3	=A2.(EID)	
4	=A2.(age(BIRTHDAY))	
5	=A2.()	return A2

```
1
2
3
4
5
```

```
39
33
43
29
38
```

$[a_1,\ldots,a_n]$

Description:

Define a constant sequence

Syntax:

$[a_1,a_2\ldots,a_n]$

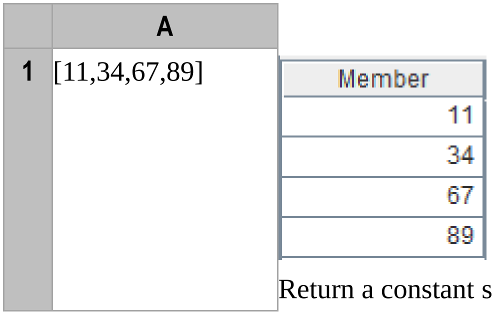
Remark:

Define a sequence that is composed of n members $a_1\ldots a_n$.

Return value:

A constant sequence

Example:



	A
1	[11,34,67,89]

Member
11
34
67
89

Return a constant sequence composed of 11, 34, 67 and 89

abs()

Description:

Compute the absolute value of the parameter

Syntax:

abs(*numberExp*)

Remark:

Compute the absolute value of *numberExp*

Parameters:

numberExp Data for which you want to compute the absolute value

Return value:

Numeric

Example:

–	**abs(-3245.54)**	**3245.54**
–	**abs(-987)**	**987**

acos()

Description:

Compute the arc cosine value of the parameter

Syntax:

acos(*number***)**

Remark:

The parameter *number* is a real number between -1 and 1

Parameters:

number The real number for which you want to compute the arc cosine

Return value:

Arc cosine

Example:

–	**acos(-1)**	**3.141592653589793**
–	**acos(cos(pi()/2))**	**1.5707963267948966**
–	**acos(cos(0))**	**0.0**

Related concepts:

asin()

atan()

after()

Description:

Compute the new date which is certain days after a date

Syntax:

after (*dateExp*, *n***)**

dateExp ± *n* **after (***dateExp*, *n***)**

Remark:

Compute the new date which is *n* days/*n* months/*n* years after the date *dateExp*

When getting the same date in last year according to the specified date, if the date does not exist, then return the last day of the same month in last year

For example, **after@m("2009-03-31",-1)** returns **2009-02-28**

Parameters:

dateExp The specified orgin date expression whose result must be a date or a string of standard date format.

n The integer expression for computing a new date *n* days/years/months later; a negative integer indicates computing a new date of *n* days/years/months before.

Options:

@y Compute the new date which is *n* years after the specified date

@q Compute the new date which is n quarters since the specified date

@m Compute the new date which is *n* months after the specified date

@s Compute the date/time which is *n* seconds after the specified date

@ms Compute the date/time which is *n* milliseconds after the specified date

By default, compute the new date which is *n* days after the specified date

Return value:

Date/time

Example:

- **after(datetime("19800227","yyyyMMdd"),5)** **1980-03-03 00:00:00**
- **after@y(datetime("19800227","yyyyMMdd"),5)** **1985-02-27 00:00:00**
- **after@q(datetime("19800227","yyyyMMdd"),5)** **1981-05-27 00:00:00**
- **after@m(datetime("19800227","yyyyMMdd"),5)** **1980-07-27 00:00:00**
- **after@s(datetime("19800227","yyyyMMdd"),5)** **1980-02-27 00:00:05**
- **after@ms(datetime("19800227","yyyyMMdd"),5)** **1980-02-27 00:00:00**
- **after(datetime("19800227","yyyyMMdd"),-3)** **1980-02-24 00:00:00**
- **after@y(datetime("19800227","yyyyMMdd"),-3)** **1977-02-27 00:00:00**
- **after@q(datetime("19800227","yyyyMMdd"),-3)** **1979-05-27 00:00:00**
- **after@m(datetime("19800227","yyyyMMdd"),-3)** **1979-11-27 00:00:00**
- **after@s(datetime("19800227","yyyyMMdd"),-3)** **1980-02-26 23:59:57**
- **after@ms(datetime("19800227","yyyyMMdd"),-3)** **1980-02-26 23:59:59**
- **datetime("19800227","yyyyMMdd")+5** **1980-03-03 00:00:00**
- **datetime("19800227","yyyyMMdd")-5** **1980-02-22 00:00:00**

age()

Description:

Compute the number of whole years between a specified date and the current time

Syntax:

age(*dateExp***{,** *formatExp* **})**

age(*stringExp***,***formatExp***)**

Remark:

Compute the number of whole years between the date specified by parameter *dateExp* and the current time

Parameters:

dateExp Date expression whose result is the date

stringExp String expression whose result is normal date or Chinese datetime format string

formatExp Format the expression, such as "**yyyyMMdd**","**yyyy-MM-dd**"

Options:

@y	The computation is accurate to the year
@m	The computation is accurate to the month
	The computation is accurate to the day by default

Return value:

Integer

Example:

- age(date("1980-09-01"))
- age@m(datetime("1980-09-01 12:23:56"))
- age@y("19800227","yyyyMMdd")

aggregate()

mdb.aggregate()

Description:

Perform aggregate query on a MongoDB and return the result as a table sequence

Syntax:

mdb.**aggregate**(*c*,*g*)

Remark:

Perform aggregate operation on data of table *c* in a MongoDB according to *g*, and return the result in the form of a cursor

Parameters:

mdb	Connection to MongoDB
c	Table name
g	AGGREGATE statement, whose syntax is the same as that of the MongoDB

Return value:

A table sequence

Example:

	A	
1	=mongodb("mongo://127.0.0.1:27017/myTest?user=root&password=sa")	
2	=A1.aggregate("Score","[{$group:{_id:\"$id\",totalscore:{$sum:\"$score\"}}},{$match:{totalscore:{$gt:230}}}]")	Select the students whose total score is greater than 230 from table Score and then group them by id
3	=A1.close()	Close the database connection

Related concepts:

mongodb()

mdb.close()

mdb.find()

mdb.distinct()

mdb.count()

align()

P.align()

Description:

Align the records of a record sequence to a sequence, so as to group the record sequence.

Syntax:

P.**align**(*A*:*x*,*y*)	If omitting *x*,*y*, then align the current records of *P* with members of *A*.
P.**align**(*n*,*y*)	Equal to P**.align (to(**n**)**,y**)**, and support **@r**.

Remark:

Align the records of *P* to *A* by the associated fields *x* and *y*, which means that, compute y against each record of *P* and compute *x* against each record of *A*, and then compare each value of *x* and *y*, if two of them are equal, then the two records are aligned.

This function is mainly for the scenario of primary table and subtable. Usually, there is a ref field of the subtable associated with the primary table to reference the associated record of the primary table. In this occasion, the *x* expression usually acts as the ref field. Then, retrieve the values from the ref field in the subtable, judging if the records match the corresponding record in the primary table. If they match, then it indicates that the two tables are aligned.

There is usually the one-to-many correspondence between the primary table and the subtable, that is, a record in *A* is associated with multiple records in *P*. Therefore, construct a sequence composed of multiple associated records of *P*, take and store this sequence as a member of the result sequence, then the number of members in the result sequence is the same as that of the *A*.

Parameters:

P Sub record sequence/table sequence

A Primary record sequence/ primary table sequence, according to which another record sequence is aligned

x The field or field expression of *A* for relational operation. If omitted, then it is interpreted as **~**

y The alignment expression of *P*; If omitted, then it is interpreted as *P*.**~**

n Integer

Options:

@a Of *P* records, return all members which are aligned according to the members of A, and group the resulting sequence. By default, only the first member will be returned.

@b If *A* is an ordered sequence, the binary search will be used.

@r *y* is an integer sequence, and every member of *y* is taken as the alignment position for aligning *P* to *A*. *P* will be aligned to the designated position of *n* overlappedly.

@p The return value is composed of the sequence numbers of members in the P

@n Return all members of *P* whose records can be aligned to members of *A*. Of the result sets, there is an extra group to hold those members whose records fail to be aligned.

@s In *P*, sort members by the same order as members of *A*, and put non-corresponding member(s) to *A* at the tail.

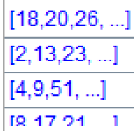
Return value:

Aligned *P*

Example:

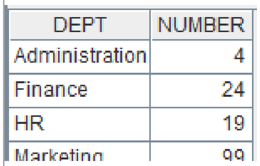
- Computation after the subtable has been aligned to the primary table

	A
1	=demo.query("select * from DEPARTMENT")
2	=demo.query("select * from EMPLOYEE")
3	=A2.align@a(A1:DEPT,DEPT)
4	=A1.new(DEPT, A3(#).count():NUMBER)
5	=A2.align(A1:DEPT,DEPT)



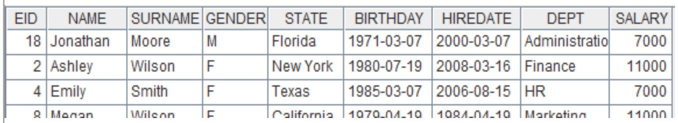
[18,20,26, ...]
[2,13,23, ...]
[4,9,51, ...]
[8,17,21, ...]

Align the **EMPLOYEE** table to the **DEPARTMENT** table



DEPT	NUMBER
Administration	4
Finance	24
HR	19
Marketing	99

Perform the relational operation on **A3** and **A1**

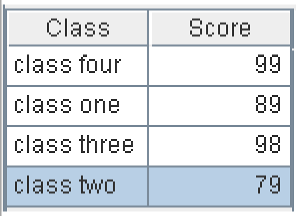


EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
18	Jonathan	Moore	M	Florida	1971-03-07	2000-03-07	Administratio	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
8	Megan	Wilson	F	California	1979-04-19	1984-04-19	Marketing	11000

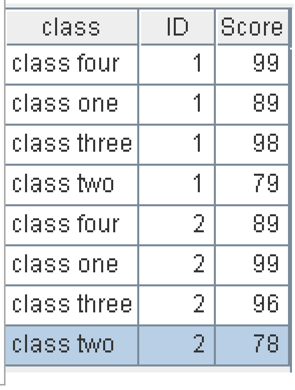
Return the first satisfying member by default

- For special sorting

	A
1	=create(Class,Score)
2	=A1.record(["class four",99,"class one",89, "class three",98,"class two",79])
3	=["class one","class two","class three","class four"]
4	=create(class,ID,Score)
5	=A4.record(["class four",1,99,"class one",1,89, "class three",1,98,"class two",1,79, "class four",2,89, "class one",2,99,"class three",2,96,"class two",2,78])

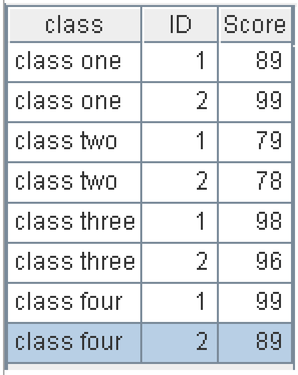


Class	Score
class four	99
class one	89
class three	98
class two	79



class	ID	Score
class four	1	99
class one	1	89
class three	1	98
class two	1	79
class four	2	89
class one	2	99
class three	2	96
class two	2	78

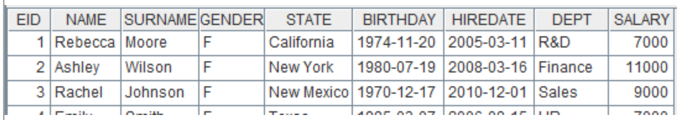
6	=A5.align@a(A3,class)	Use the **@a** option because each Class has multiple records In this case, if using the **sort** function, the order will be "four,one,three,two" according to the alphabetical order.
7	=A5.align@a(A2:Class,class)	



class	ID	Score
class one	1	89
class one	2	99
class two	1	79
class two	2	78
class three	1	98
class three	2	96
class four	1	99
class four	2	89

- ➢ Alignment with integer sequence, which can usually speed up the computation

	A	
1	=demo.query("select * from FAMILY ")	
2	=demo.query("select * from EMPLOYEE")	
3	=A1.align@b(A2:EID,EID)	**A2** is sorted and you can use the binary search to speed up the computation



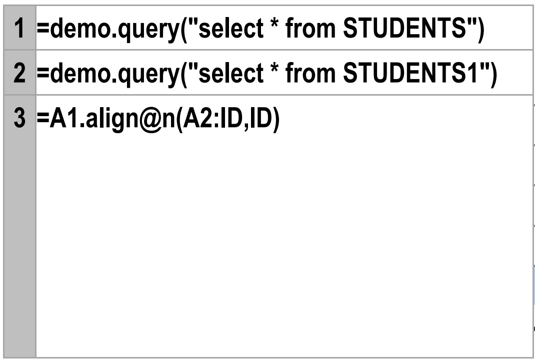
EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000

- ➢ @r option is to align the primary table to the subtable overlappedly by the position designated by the integer sequence

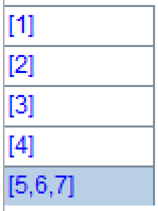
	A	
1	=demo.query("select * from FAMILY")	
2	=demo.query("select * from EMPLOYEE")	
3	=A1.derive(A2.pselect@a(EID==A1.EID):SubtableNo)	First, find the sequence numbers of **EMPLOYEE** records corresponding to the **FAMILY** table, second, store these sequence numbers in the **SubtableNo** field as the integer sequence
4	=A33.align@r(11,SubtableNo)	Align to n directly by the **SubtableNo**
5	=A3.align@rp(11,SubtableNo)	Add the **@p** option, return the sequence number of the record instead of the record itself

- ➢ @n option is to align the primary table to the subtable and return all members. The result set's last group will be used to store the unaligned members.

	A



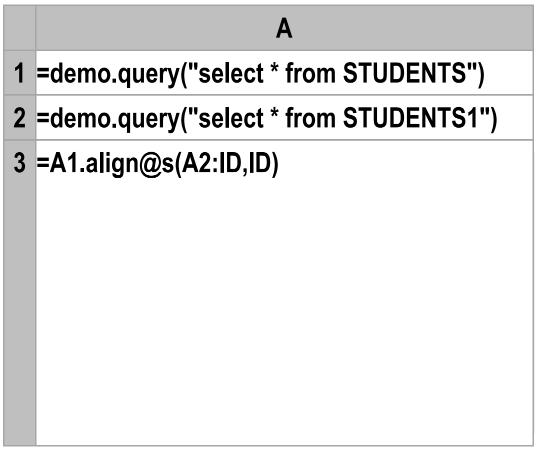
```
1 =demo.query("select * from STUDENTS")
2 =demo.query("select * from STUDENTS1")
3 =A1.align@n(A2:ID,ID)
```



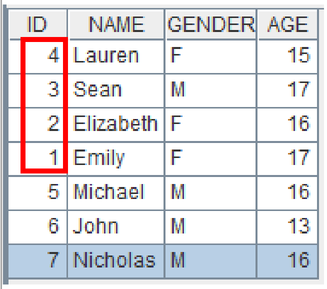
```
[1]
[2]
[3]
[4]
[5,6,7]
```

The members in [5,6,7] are unaligned

- @s option is to sort the records in the subtable by the members of primary table.



```
  A
1 =demo.query("select * from STUDENTS")
2 =demo.query("select * from STUDENTS1")
3 =A1.align@s(A2:ID,ID)
```



ID	NAME	GENDER	AGE
4	Lauren	F	15
3	Sean	M	17
2	Elizabeth	F	16
1	Emily	F	17
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

append()

T.append(*T*$_i$, …)

Description:

Append in order the records of one table sequence to another

Syntax:

T.append(*T*$_i$,…)

Remark:

Append the records of *T*$_i$ to the table sequence *T*, and empty the *T*$_i$ table sequence

Options:

@p If there is a primary key in *T* and a record of *T* whose primary key value is the same as that of a record of *T*$_i$, the former will be replaced by the latter.

Parameters:

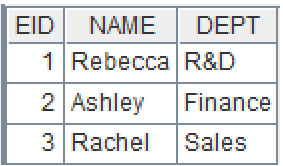
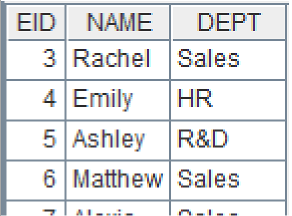
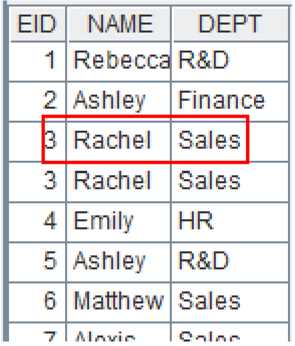
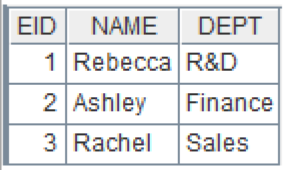
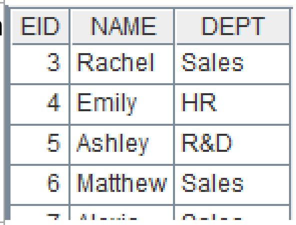
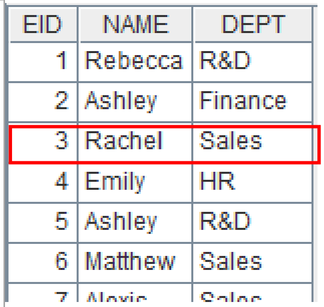
T	Table sequence
T$_i$	Table sequence which has the same number of fields as *T* indicates

Return value:

The table sequence *T* after being modified

Example:

A

1	=demo.query("select EID,NAME,DEPT from EMPLOYEE where EID<4")	 EID NAME DEPT 1 Rebecca R&D 2 Ashley Finance 3 Rachel Sales
2	=demo.query("select EID,NAME,DEPT from EMPLOYEE where EID>=3")	 EID NAME DEPT 3 Rachel Sales 4 Emily HR 5 Ashley R&D 6 Matthew Sales 7 Alexis Sales
3	=A1.append(A2)	 EID NAME DEPT 1 Rebecca R&D 2 Ashley Finance 3 Rachel Sales 3 Rachel Sales 4 Emily HR 5 Ashley R&D 6 Matthew Sales 7 Alexis Sales The same record appears repeatedly
4	=demo.query("sselect EID,NAME,DEPT from EMPLOYEE where EID< 4").primary(EID)	 EID NAME DEPT 1 Rebecca R&D 2 Ashley Finance 3 Rachel Sales A primary key is set
5	=demo.query("select EID,NAME,DEPT from EMPLOYEE where EID>= 3")	 EID NAME DEPT 3 Rachel Sales 4 Emily HR 5 Ashley R&D 6 Matthew Sales 7 Alexis Sales
6	=A4.append@p(A5)	 EID NAME DEPT 1 Rebecca R&D 2 Ashley Finance 3 Rachel Sales 4 Emily HR 5 Ashley R&D 6 Matthew Sales 7 Alexis Sales The record that appeared in A5 will be replaced to avoid the duplication

array()

r.array()

Description:

Get the field values from a record respectively and return them as a sequence

Syntax:

r.**array**()

Remark:

Get all the field values from record r respectively and return them as a sequence

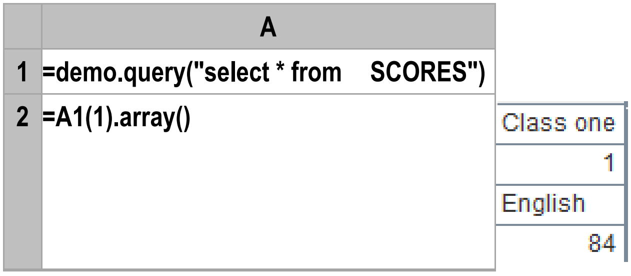
Parameters:

r	A record

Return value:

A sequence

Example:



	A
1	=demo.query("select * from SCORES")
2	=A1(1).array()

Class one
1
English
84

s.array()

Description:

Split a string by delimiter so as to form a new sequence and return it

Syntax:

s.**array**(*d*)

Remark:

Split string *s* by delimiter *d* to form a new sequence. The data type of the members of the new sequence will be processed by default, that is, consider the numbers as numeric values, **[]** as a sequence, **2001-01-01** as a date, and so on.

Parameters:

s	The string to be splitted
d	The delimiter; If it is omitted, comma will be used by default

Options:

@s	Split the string into a sequence of strings, the data type will not be processed. Those delimiters between the quotation marks or the brackets will be ignored.
@1	This option is a suboption of **@s**. It splits string into 2 parts by the first *d* found
@b	This option is a suboption of **@s**. Those delimiters between the quotation marks or the brackets will not be ignored and the data type will be processed

Return value:

A new sequence of strings

Example:

	A

1	="1,[a,b],(2,c),'5,6'"	
2	=A1.array()	[1,[a,b],"(2,c)","5,6"]
3	=A1.array@1()	[1,"[a,b],(2,c),'5,6'"]
4	=A1.array@s()	["1","[a,b]","(2,c)","'5,6'"]
5	=A1. array@b()	[1,"[a","b]","(2","c)","'5","6'"]
6	="a:b:c".array(":")	["a","b","c"]

Related concepts:

A.string()

r.string()

asc()

Description:

Get the Unicode value of the character at the specified position in a string; if it is an ASCII character, then return its ASCII code.

Syntax:

asc(*string*{, *nPos*} **)**

Remark:

Obtain the Unicode value of the character at the specified position *nPos* of *string*, if it is ASCII character, then return its ASCII code.

In general, English characters and their extended characters are all the ASCII characters; Chinese, Japanese, Korean, and other Asian characters are all Unicode characters. ASCII character is an 8-bit character set, and the Unicode character is a 16-bit character set, of which 3 bits are used to indicate the character type.

Parameters:

string	The given strings
nPos	Integer expression, its value is 1 by default

Return value:

Integer

Example:

–	asc("def")	100 (ascii)
–	asc("def",2)	101 (ascii)
–	asc("China")	67 (Unicode)
–	asc("China",2)	104(Unicode)

Related concepts:

char()

asin()

Description:

Compute the arc sine value of the parameter.

Syntax:

asin(*number*)

Remark:

The parameter *number* is real number between -1 and 1.

Parameters:

number The real number for which you want to compute the arc sine value

Return value:

Arc sine value

Example:

asin(-1)	-1.5707963267948966
asin(sin(pi()/2))	1.5707963267948966
asin(sin(0))	0.0

Related concepts:

acos()

atan()

atan()

Description:

Compute the arc tangent value of the parameter.

Syntax:

atan (*number*)

Remark:

The parameter *number* is a real number

Parameters:

number Real number for which you want to compute the arc tangent

Return value:

Arc tangent

Example:

atan(1)	0.7853981633974483
atan(tan(pi()/2))	1.5707963267948966
atan(tan(0))	0.0

Related concepts:

asin()

acos()

avg()

A.avg()

Description:

Compute the average value of the non-null members in sequence *A*.

Syntax:

A.**avg()** Equivalent to **avg**(x_1,…,x_n)

Remark:

To compute the average value of the non-null members in a sequence is equal to executing the *A*.**sum()**/*A*.**count()** operation towards the sequence; if the number of non-null members is 0, the average value will be null. Members that are not numerical values will be automatically skipped.

Parameters:

A A sequence

Return value:

The average value of the non-null members in sequence *A*

Example:

	A	
1	=[1,3,5,6].avg()	3.75
2	=[2,null,4,3].avg()	3.0
3	=[].avg()	null
4	=[2,4,3,"aaa"].avg()	**3.0** Skip members that are not numerical values
5	=avg(2,null,4,3)	3.0

Related Concepts:

A.sum()

A.count()

A.min()

A.max()

A.variance()

A.avg()

A.avg(*x*)

Description:

Compute x with each member of the sequence and then compute the average value of the non-null members of the new sequence.

Syntax:

A.**avg**(*x*) Equivalent to *A*.(*x*).**avg**()

Remark:

Compute x with each member of the sequence and return the average value of the non-null members of the new sequence

Parameters:

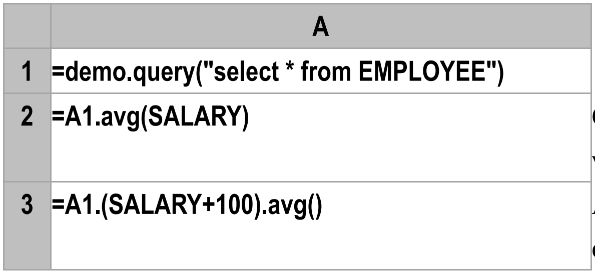
A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names

Return value:

Numerical values

Example:



	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.avg(SALARY)	Compute the average value of non-null values of SALARY
3	=A1.(SALARY+100).avg()	Add 100 to each value of SALARY and then compute the average value

Related concepts:

A.avg()

avgif()

A.avgif()

Description:

Get the intersection of positions of the members in a sequence, and return the average of the members

Syntax:

A.**avgif(**A_i:x_i, …**)**

Remark:

Locate all the positions of member x_i or sub-sequence x_i in sequence A_i, acquire the intersection of these positions and return the average value of the non-null members of *A* in these positions

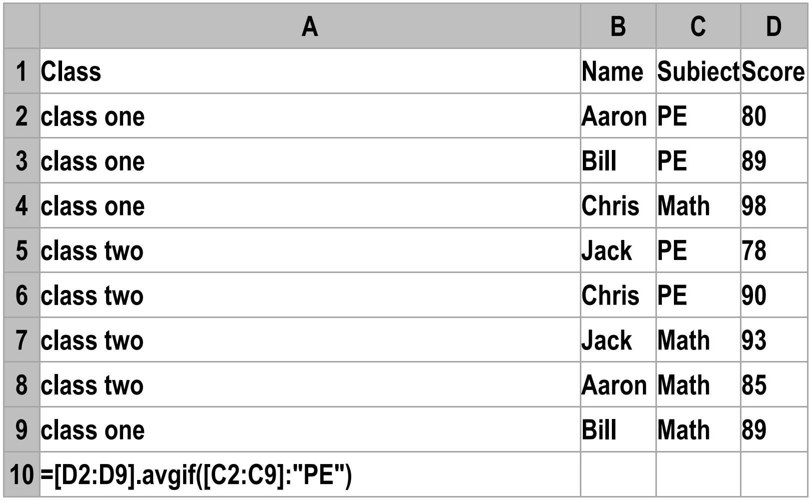
Parameters:

A_i	A sequence
x_i	The members in A_i or the sequence consisting of members of A_i
A	The target sequence

Return value:

The average value of the non-null members in those specified positions in *A*

Example:



	A	B	C	D	
1	Class	Name	Subject	Score	
2	class one	Aaron	PE	80	
3	class one	Bill	PE	89	
4	class one	Chris	Math	98	
5	class two	Jack	PE	78	
6	class two	Chris	PE	90	
7	class two	Jack	Math	93	
8	class two	Aaron	Math	85	
9	class one	Bill	Math	89	
10	=[D2:D9].avgif([C2:C9]:"PE")				**84.25**, the search with a

					single condition
11	=[D2:D9].avgif([C2:C9]:"PE",[A2:A9]:"class one")				**84.5**, the search with multiple conditions

Related concepts:

A.countif(A_i:x_i,…)

A.sumif(A_i:x_i,…)

A.minif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

break {}

Description:

To break the current loop.

Syntax:

break {*a*}

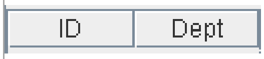
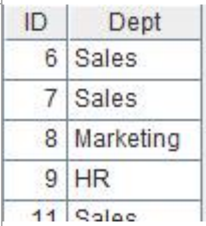
Remark:

Break a loop whose master cell is *a*. If *a* has value, then break the current loop.

Parameters:

a The master cell of a loop block. If *a* is omitted, then the function indicates the current loop.

Example:

	A	B	C	D	
1	=[]				[1, 1, 2, 2, 3, 3]
2	for 5				The computation of **A1** is a sequence **[1, 1, 2, 2, 3, 3]**;
3		for 2			break **A2** is to control the loop of the **A2** level
4			if A2==4	break A2	
5			>A1=A1|[A2]		
6	=create(ID,Dept)				 ID Dept
7	=demo.query("select * from EMPLOYEE")				
8	for A7				
9		if A8.NAME=="Lucy"			Extract the data of **DEPT** from the **EMPLOYEE** table
10			Break		and insert them to a table sequence uninterruptedly until the record whose **NAME** value is "**Lucy**" appears
11		>A6.insert(0,A8.EID:ID,A8.DEPT:Dept)			 ID Dept 6 Sales 7 Sales 8 Marketing 9 HR 11 Sales The return value of **A6** is shown in the figure

calc()

A.calc()

Description:

Compute an expression with one or more specified records and return the results.

Syntax:

A.**calc**(*k*,*x*)	Compute *x* against the *k*th member of *A* and return the result
A.**calc**(*p*,*x*)	Compute *x* against the members of *A* specified by the integer sequence *p* and return the result sequence

Remark:

Compute an expression against one or more specified records and return the result.

Parameters:

A	A sequence/a record sequence
x	An expression, which is generally a field name or a legal expression composed of field names, and "**~**" is used to reference the current record.
k	An integer, specifying a record
p	An integer sequence, specifying multiple records

Return value:

A computation of *x* or a sequence composed of the computations of *x*

Example:

- Compute *x* against the *kth* member

	A	
1	=[1,3,6,2,8]	
2	=A1.calc(3,~*2)	12

Compute expression "**~*2**" against the third member of sequence **A1**, "**~**" indicates the current member. The result is **12**.

- Compute *x* against the members of *A* specified by the integer sequence *p*

	A	
1	=[1,3,6,2,8]	
2	=A1.calc([4,3],~*2)	[4,12]

Compute expression "**~*2**" against the fourth and the third members of sequence **A1** separately and the result is **[4,12]**.

- Do the computation against a table sequence

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.calc(2,age(HIREDATE)+5)	Compute expression against the second record, the result is **11**
3	=A1.calc(A1.pselect@a(DEPT=="Administration"), age(HIREDATE)+5)	Compute "**age(HIREDATE)+5**" against all the records whose **DEPT** is "**Administration**" and return the result sequence **[19,11,11,10]**

call()

Description:

Call the cellset file and return the first result set.

Syntax:

call(*dfx,arg1,…*)

Remark:

Pass in the parameter *arg1,…*, call the cellset file *dfx*, return the first result value, and then closes. The **call** function is not a cellset function but a function can be used outside the cellset. The function will use the @s option automatically to search the dfx. It will search the file names with the non-absolute paths in a specified order: Class path -> Searching path -> Main path. By default, the main path is the current path. *dfx* can be the file object, which can be imported and executed directly.

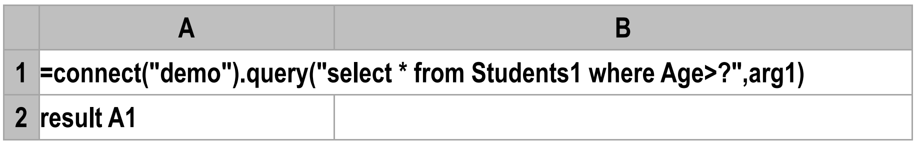
Call **call** with parameter arg1…. These parameter values will be assigned to each parameter of *dfx* one by one, independent of parameter names in the parameter list of *dfx*.

Parameters:

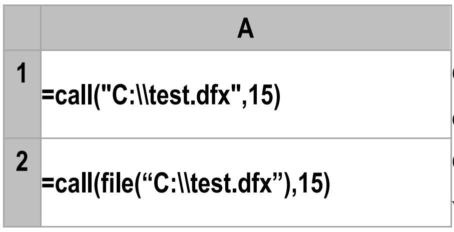
dfx	Cellset file
arg1,…	Argument

Example:

The contents of C:\\test.dfx cellset file are as follows, arg1 is the cellset parameter:



	A	B
1	=connect("demo").query("select * from Students1 where Age>?",arg1)	
2	result A1	



	A	
1	=call("C:\\test.dfx",15)	Call the cellset file, and return the first result value. Query the data of students over 15 years old
2	=call(file(“C:\\test.dfx”),15)	Get the the value of result from the file object and query those who are older than 15

Note:

Comma can be used as the separator when multiple results are returned. E.g. result A1,B2.

call path/dfx(…)

Description:

With esProc JDBC, search for and execute a program file locally. If not found, then search the server

Syntax:

call *path*/*dfx*(…)

Remark:

Pass in parameter … and call the cellset file *dfx* waiting to be executed. Firstly, search the cellset file with absolute or relative addressing path. If not found, then search the server for it (server list is configured in the dfxConfig.xml file). Once the computation is completed, return a sequence composed of members of the result set; if there are multiple result sets, then return a sequence composed of these result sets. This is similar to calling a stored procedure in the normal database driver. During execution, use ***con*.prepareCall()** to call statement. The parameters can be written in the statement, or just use ***st*.setObject()** to set them. Once Statement is generated, use ***st*.execute()** to execute it and return the result set.

When calling **call**, use the parameter arg1…. These parameter values can be set into each parameter of *dfx*, and irrelevant to the parameter names on the parameter list of *dfx*.

Parameters:

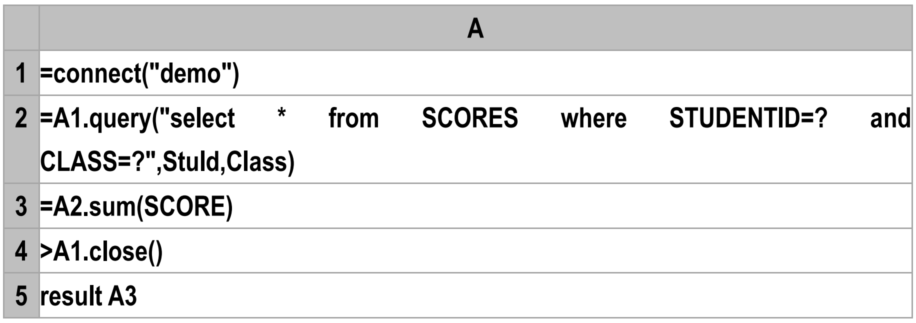
path The relative addressing path or absolute path of a file. When not specified, then it represents the relative addressing path

dfx Cellset file

… Parameters. Use comma to separate if there are multiple parameters. The parameter names for receiving parameters will not be used. Instead, they will be assigned in their order.

Example:

test.dfx cellset file has the following contents, in which **StuId** and **Class** is the cellset parameters.



	A
1	=connect("demo")
2	=A1.query("select * from SCORES where STUDENTID=? and CLASS=?",StuId,Class)
3	=A2.sum(SCORE)
4	>A1.close()
5	result A3

Test code is shown below:

```
public void testDataServer() {
        Connection con = null;
        com.esproc.jdbc.InternalCStatement st;
        try{
            Class.forName("com.esproc.jdbc.InternalDriver");
            con= DriverManager.getConnection("jdbc:esproc:local://");
            // Call the stored procedure in which test is the file name of dfx
            st =(com.esproc.jdbc.InternalCStatement)con.prepareCall("call test(?)");
            // Set the first parameter
            st.setObject(1,"4");
           // Set the second parameter
            st.setObject(2,"Class one");
            // Execute stored procedure
            st.execute();
            // Get result set
            ResultSet set = st.getResultSet();
            // Print results
            printRs(set);
            // The following statement leads to the same result as the above-mentioned
method of calling
            st =(com. esproc.jdbc.InternalCStatement)con.prepareCall("call test(4,\
"Class one\")");
            st.execute();
            set = st.getResultSet();
            printRs(set);
        }
        catch(Exception e){
            System.out.println(e);
        }
        finally{
            // Close the connection
            if (con!=null) {
                try {
                    con.close();
                }
                catch(Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
```

callx()

Description:

Compute cellset file *dfx* in parallel; return a sequence composed of result sets of the multiple tasks..

Syntax:

callx(*dfx*,*arg1*…;*h***)**

Remark:

Allocate tasks to servers specified by the sequence parameter *h* to compute the cellset file *dfx* in parallel. The length of parameter sequence *arg1* represents the number of tasks. When the number of tasks allocated to all servers reach the limit, the task-allocating will be suspended, and will resume allocating only if any server become idle. Once any error occurs to a certain server, tasks on it will be allocated to other servers.

Once the computation is completed by invoking servers running in parallel, the result set will be returned to piece together a sequence; in case there are multiple result sets, return the sequence of sequences.

Parameters:

dfx Cellset file, dfx can be set as an absolute or a relative path. The relative path is the **Addressing Path** under the Tool ->Option-> [Environment] menu.

arg1,… The parameter value to be passed to *dfx*. This parameter is usually in the form of a sequence. The number of parameters to be passed to *dfx* is just the number of sequences. In parallel computing, the computational task will be decomposed into subtasks according to the length of parameter sequence. Each member of the sequence will be passed to the respective subtask as the parameter value of *dfx*. If this parameter is a single-value parameter, then this single value will be copied to every subtask.

h Server sequence in which each server has a description in the form of a character string as "address:port number", for example, "192. 168. 0. 86: 4001". By default, the server will not be used. The computation will be performed by being distributed to multiple threads just in the current process.

Options:

@a Tasks are strictly allocated to the corresponding server one by one. The list of servers and the length of parameters must be the same.

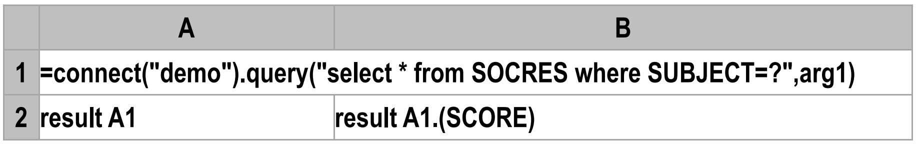
@p Based on the actual task allocating situation, create a sequence composed of the return value with two levels. In other words, the return results of subtasks will be grouped by the servers.

@1 Indicates that the number of parallel tasks being handled on a node machines is 1.

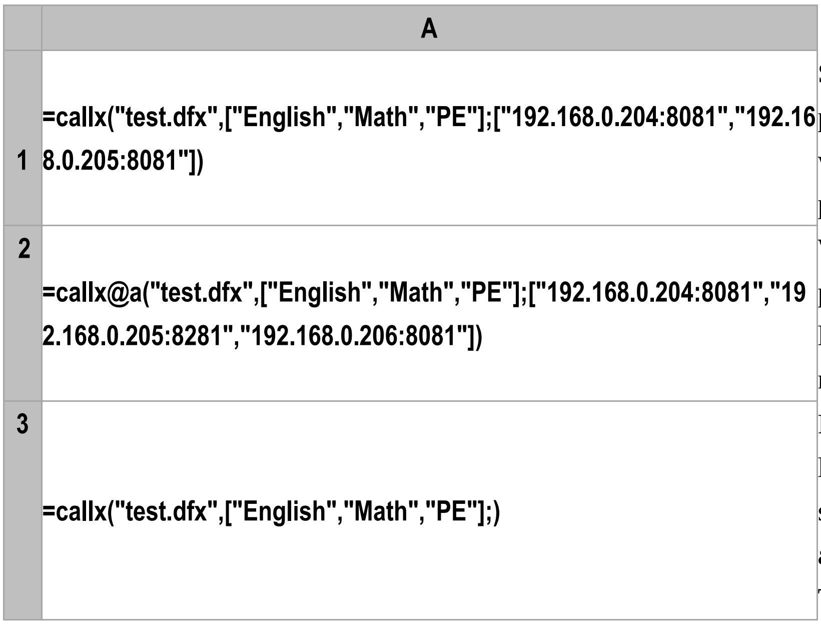
Example:

The test.dfx is the deployment file for the remote node server. The node machines are respectively the [”192.168.0.204”,” 192.168.0.205”,” 192.168.0.206”], and the port number is 8081. The file contents are shown below:

1) **Single Parameter**



	A	B
1	=connect("demo").query("select * from SOCRES where SUBJECT=?",arg1)	
2	result A1	result A1.(SCORE)



	A	
1	=callx("test.dfx",["English","Math","PE"];["192.168.0.204:8081","192.168.0.205:8081"])	Servers running in parallel. The 3 tasks will be allocated to two parallel machines.
2	=callx@a("test.dfx",["English","Math","PE"];["192.168.0.204:8081","192.168.0.205:8281","192.168.0.206:8081"])	With @a, the parameters and server lists are of the same number
3	=callx("test.dfx",["English","Math","PE"];)	Run parallelly and locally. The file will be stored under the **addressing path** of Tools->Options->[Envir

		onment] menu

2) **Multiple Parameters**

	A
1	=connect("demo")
2	=A1.query("select * from EMPLOYEE where EID in (?)and GENDER=?",arg1,arg2)
3	=A1.close()
4	result A2

	A	
1	=callx("test.dfx",[[1,20,6,14,5]],"F";"192.168.0.204:8081")	Pass 2 parameters of string and sequence types to one node machine
2	=callx("test.dfx",[[1,20,6,14,5],[22,33,44]],"F";["192.168.0.204:8081","192.168.0.205:8081"])	Send multiple parameters to multiple node machines. For a parameter with the same value, they can be passed in the form of single assignment.
3	=callx("test.dfx",[[1,20,6,14,5],[22,33,44]],["F","M"];["192.168.0.204:8081","192.168.0.205:8081"])	When multiple node machines receive the same parameter yet with different values, the parmeter needs to be passed in the form of a sequence

canvas()

Description:

Create a canvas object.

Syntax:

canvas()

Remark:

To define canvas in esProc, simply use the canvas() function in a cell. Then, the plotting program can invoke the canvas object with the cell name directly, set the plotting parameters, or plot straightforwardly.

Example:

	A

1	>map1=canvas()	Assign the canvas to map1 which can be referenced directly in plotting.

case()

Description:

According to the various results of computing different expressions, return various values.

Syntax:

case($x,x_1:y_1,\ldots,x_k:y_k;y$**)**

Remark:

This function will compute from left to the right. Compute the expression x for judgment that comes first, and then compute the other x_k. If there is any result of expression x_k equals to the result of x, then return the result of y_k, and the computation is terminated. If none of the result of expression x_k equals to the result of x, and there exists default expression y, then return the result of y, otherwise, return **null**.

Parameters:

x	Expression for judgment
x_k	Value expression
y_k	result expression
y	Default expression.

Example:

	A	
1	=3	
2	=case(A1,1:"Dept 1",2:"Dept 2",3:"Dept 3";"Admin Dept")	Dept 3
3	>A1=4	
4	=case(A1,1:"Dept 1",2:"Dept 2",3:"Dept 3",4:;"Admin Dept")	null

Related concepts:

if()

in()

ceil()

Description:

Truncate data at the specified position, and carry the remaining part if any.

Syntax:

ceil(*numberExp*, {*nExp*}**)**

Remark:

Truncate the data *numberExp* at the specified position *nExp*, and carry the remaining part (if any).

Parameters:

numberExp	Data to be truncated
nExp	Integer number for specifying the truncation position, **>0**: Move the decimal point to the right for *nExp* places,

<0: Move the decimal point to the left for *nExp* places,

=0: Indicate the current decimal place.

Return value:

Numeric

Example:

–	ceil(3450001.004,0)	3450002.0
–	ceil(3450001.004,-1)	3450010.0
–	ceil(3450001.004,-2)	3450100.0
–	ceil(3450001.004,1)	3450001.1
–	ceil(3450001.004,2)	3450001.01

Related concepts:

floor()

round()

char()

Description:

According to the given Unicode or ASCII code, get the corresponding characters.

Syntax:

char(*int* **)**

Remark:

In general, the English characters and their extended characters are all the ASCII code; Chinese, Japanese, Korean, and other Asian characters are all Unicode characters. ASCII character is an 8 bit character set, and Unicode character is a 16 bit character set, of which 3 bits are used to indicate the character type.

Parameters:

int	Integer expression, Unicode code or ASCII code

Return value:

Character

Example:

–	char(87)	'W'

Related concepts:

asc()

clear

Description:

Clear the cell value

Syntax:

clear *a*:*b*,*c*,*d*:*e*

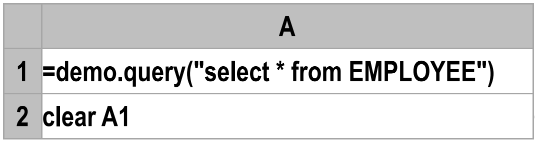
Remark:

Clear the cell value of certain cells. Leaving *a* alone indicates clearing the cell values of a code block which takes *a* as the master cell

Parameters:

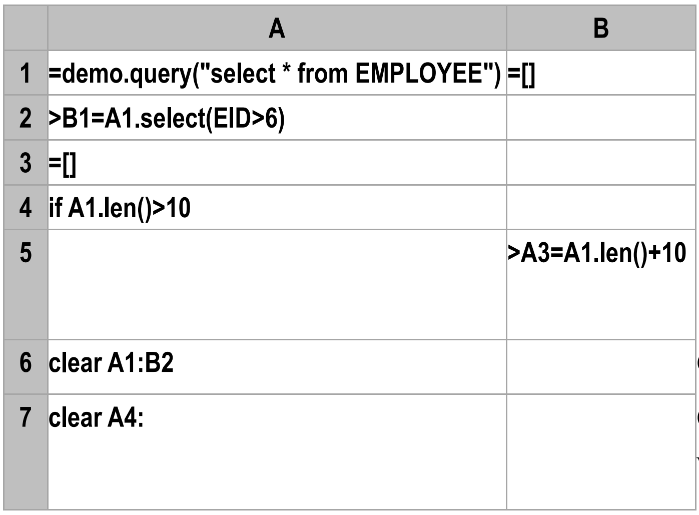
a:*b*,*c*,*d*:*e* The cells whose values need to be cleared

Example:



	A
1	=demo.query("select * from EMPLOYEE")
2	clear A1

Clear value of cell A1



	A	B
1	=demo.query("select * from EMPLOYEE")	=[]
2	>B1=A1.select(EID>6)	
3	=[]	
4	if A1.len()>10	
5		>A3=A1.len()+10
6	clear A1:B2	
7	clear A4:	

Clear cell values of A1to B2

Clear cell values of a code block which takes A4 as master cell

close()

cs.close()

Description:

Close a cursor directly.

Syntax:

cs.close()

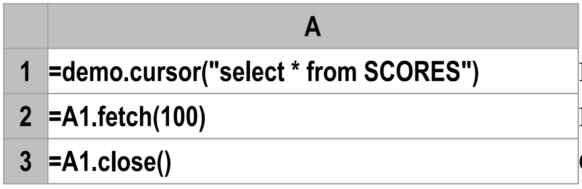
Remarks:

Close a cursor directly. By default, it will be closed automatically when all data have been fetched out.

Parameters:

cs A cursor

Example:



	A
1	=demo.cursor("select * from SCORES")
2	=A1.fetch(100)
3	=A1.close()

Return a cursor of retrieved data

Retrieve records

Close the cursor

Related concepts:

cs.fetch()

db.cursor()

db.close()

Description:

Close a datasource connection.

Syntax:

db.**close**()

Remark:

Close a database connection. By default, data will be committed before closing the database.

Parameters:

db	Database connection

Example:

	A	
1	=connect("demo")	
2	>A1.close()	Close the connection to database **demo**

Note:

connect and *close* must be used in pairs; otherwise, the connection cannot be closed.

Related concepts:

connect()

db.error()

db.commit()

db.rollback()

mdb.close()

Description:

Close the connection to a MongoDB

Syntax:

mdb.**close**()

Remark:

Close the connection to the MongoDB. By default, data will be committed before the closing.

Parameters:

mdb	Connection to MongoDB

Example:

	A	
1	=mongodb("mongo://127.0.0.1:27017/myTest?user=root&password=sa")	
2	=A1.close()	Close the MongoDB connection

Related concepts:

mongodb()

mdb.find()

mdb.count()

mdb.distinct()

mdb.aggregate()

cmp()

Description:

Compare the values of two expressions or two sequences

Syntax:

cmp(*x*, *y*)

cmp(*A*{, *B*})

cmp(*A*,**0**)

Remark:

Compare the values of two expressions *x* and *y* or two sequences *A* and *B*. An error will be reported if *x* and *y* or *A* and *B* cannot be compared*:*

When comparing the values of two expressions *x* and *y*, return **0** if they are equal; return **1** if *x* is greater than *y*; return **-1** if *x* is less than *y*.

To compare two sequences of *A* and *B*, compare the two members in the same position of *A* and *B* one by one. Return **0** if all the members are equal, otherwise compare members of the two sequences in a norml order; for the first members those are not equal, return **1** if the one in *A* is larger and return **-1** if the one in *A* is smaller. If the number of members in the sequence *A* and *B* are not the same, and members in the sequence whose member number is smaller are completely equal to their corresponding ones in the other sequence, then the value of sequence with less members will be smaller.

Parameters:

x	Expression
y	Expression
A	An *n* sequence
B	An *m* sequence. If sequence *B* does not exist, then it will be taken as a **0** sequence by default, that is, comparison will be done between sequence *A* and sequence **[0...0]**.

Example:

	A	
1	=cmp(9,5*1.2)	1
2	=cmp([3,2,1],[1,8,9])	1
3	=cmp([-1,4,8])	-1, equals to **cmp([-1,4,8],[0,0,0])**
4	=cmp([0,3,5],0)	1, equals to **cmp([0,3,5],[0,0,0])**
5	=cmp(1000,"a")	The numeric value and character string cannot be compared
6	=cmp("s","a")	1
7	=cmp([0,3],[0,3,-5])	-1

Related concepts:

Difference sequence

Intersection sequence

Union sequence

[Multiply sequence](#)

[Alignment Arithmetic Operation](#)

[Concatenate sequence](#)

commit()

db.commit()

Description:

To commit the database transaction manually.

Syntax:

db.**commit**()

Remark:

Commit the transaction manually, which is the same as commit() of the Connection class in Java.

Parameters:

db	Database connection

Options:

@k	After the execution is completed, the transaction won't be committed. If this option is omitted, the transaction will be committed.

Example:

	A	B	
1	=file("D://files//student.txt")		
2	=A1.import@t()		
3	=connect@e("demo")		Create a connection and automatically control the commit and rollback operations
4	>A3.execute@k(A2,"update STUDENTS2 set NAME=?,GENDER=?,AGE=? where ID=?",NAME,GENDER,AGE,ID)		The transaction is not committed
5	=A3.error()		Read the error code generated by the execution of the previous *sql*
6	if A5==0	>A3.commit()	Commit if there is not an error
7	else	>A3.rollback()	Roll back if there is an error
8	>A3.close()		Close the connection

Related concepts:

[db.close()](#)

[db.error()](#)

[connect()](#)

[db.rollback()](#)

conj()

A.conj()

Description:

Concatenate all the members in a sequence whose members are sequences.

Syntax:

A.conj()

Remark:

Generate a new sequence by concatenating all the members in sequence *A* whose members are sequences.

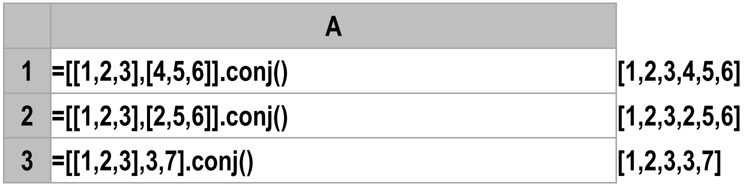
Parameters:

A A sequence whose members are sequences

Return value:

The new sequence by concatenating all the members in sequence *A*

Example:



	A	
1	=[[1,2,3],[4,5,6]].conj()	[1,2,3,4,5,6]
2	=[[1,2,3],[2,5,6]].conj()	[1,2,3,2,5,6]
3	=[[1,2,3],3,7].conj()	[1,2,3,3,7]

Related concepts:

A.union()

A.diff()

A.isect()

A.conj(*x*)

Description:

Compute x with each member of the sequence whose members are sequences, and then concatenate the computed results.

Synatax:

A.conj(x)

Remark:

Compute x against sequence *A*, whose members are sequences, by loop, and then concatenate the computed results to form a new sequence.

Parameters:

A A sequence whose members are sequences

x An expression that returns a sequence as the result

Return value:

A sequence

Example:

	A	
1	=demo.query("select * from EMPLOYEE where GENDER = 'M' order by NAME")	
2	=demo.query("select * from EMPLOYEE where GENDER = 'F' order by NAME")	
3	=[A1,A2].conj(~.(NAME))	[Rebecca,Ashley,…] Form a sequence by concatenating NAME field of A1 and A2

Related concepts:

A.conj()

cs.conj()

Description:

Split each of the records in a cursor into a TSeq or an RSeq, and return a cursor of the union set of the splitting results

Syntax:

cs.**conj**(…)

Remarks:

Compute the expression … with the records in cursor *cs*. The computation includes splitting each record into a TSeq or an RSeq, getting the union of the members or records of the splitting result and return it as a cursor.

Parameters:

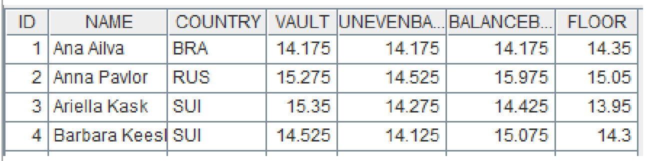
cs	A cursor
…	An expression that returns an RSeq (or a TSeq)

Return value:

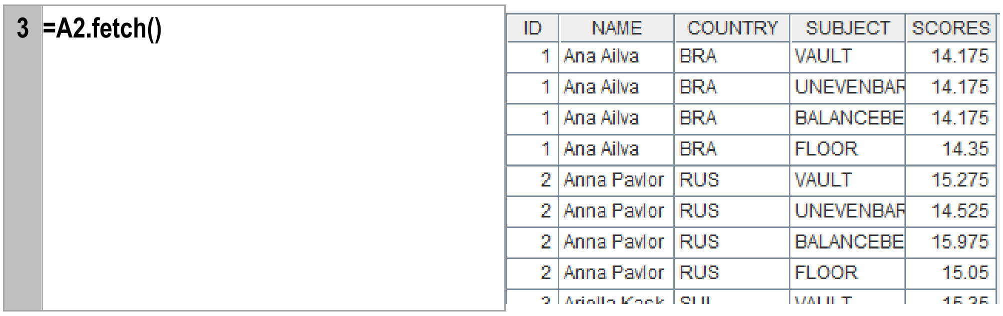
A cursor

Example:

	A
1	=demo.cursor("select * from GYMNASTICSWOMEN")
2	=A1.conj(create(ID,NAME,COUNTRY,SUBJECT,SCORES).record([ID,NAME,COUNTRY,"VAULT",VAULT,ID,NAME,COUNTRY,"UNEVENBARS",UNEVENBARS,ID,NAME,COUNTRY,"BALANCEBEAM",BALANCEBEAM,ID,NAME,COUNTRY,"FLOOR",FLOOR]))



ID	NAME	COUNTRY	VAULT	UNEVENBA...	BALANCEB...	FLOOR
1	Ana Ailva	BRA	14.175	14.175	14.175	14.35
2	Anna Pavlor	RUS	15.275	14.525	15.975	15.05
3	Ariella Kask	SUI	15.35	14.275	14.425	13.95
4	Barbara Keesl	SUI	14.525	14.125	15.075	14.3



3	=A2.fetch()

ID	NAME	COUNTRY	SUBJECT	SCORES
1	Ana Ailva	BRA	VAULT	14.175
1	Ana Ailva	BRA	UNEVENBAR	14.175
1	Ana Ailva	BRA	BALANCEBE	14.175
1	Ana Ailva	BRA	FLOOR	14.35
2	Anna Pavlor	RUS	VAULT	15.275
2	Anna Pavlor	RUS	UNEVENBAR	14.525
2	Anna Pavlor	RUS	BALANCEBE	15.975
2	Anna Pavlor	RUS	FLOOR	15.05
3	Ariella Kask	SUI	VAULT	15.25

CS.conj@x()

Description:

Union the members of a cursor sequence, and return the result as a cursor.

Syntax:

CS.**conj@x**()

Remark:

CS is the sequence of cursors whose members will be concatenated, and return a cursor. This is equivalent to merging the data in the cursors. The number of fields in each cursor of the cursor sequence must be the same.

Parameters:

CS A sequence consisting of cursors

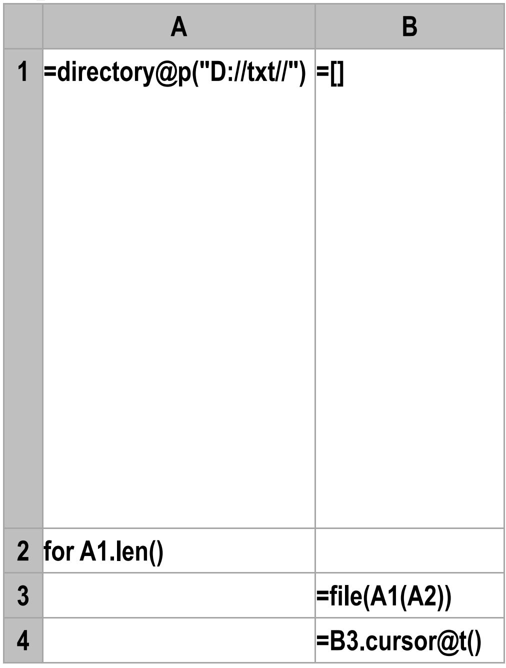
Options:

@m Union cursors in parallel in order to increase the data retrieval speed; members in the result set haven't a definite order. The option is often used to retrieve big data. There should be more than one parallel for registration code option and configuration

Return value:

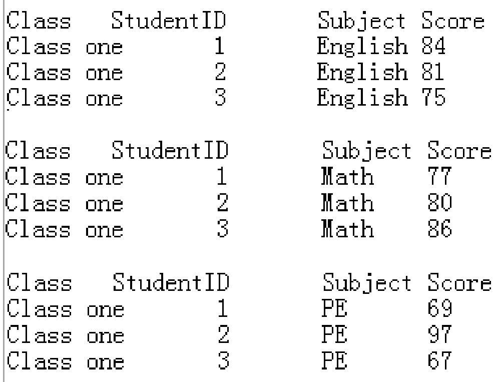
Cursor

Example:



	A	B
1	=directory@p("D://txt//")	=[]
2	for A1.len()	
3		=file(A1(A2))
4		=B3.cursor@t()

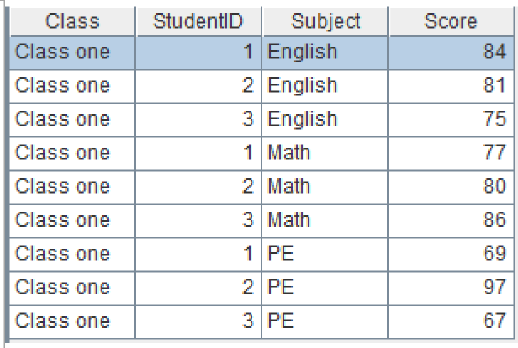
Contents of the three files in txt folder are as follows:



```
Class    StudentID        Subject Score
Class one        1        English 84
Class one        2        English 81
Class one        3        English 75

Class    StudentID        Subject Score
Class one        1        Math    77
Class one        2        Math    80
Class one        3        Math    86

Class    StudentID        Subject Score
Class one        1        PE      69
Class one        2        PE      97
Class one        3        PE      67
```

5		>B1=B1|B4	Store the cursor sequence in B1
6	=B1.conj@x()		Union the cursors
7	=A6.fetch()		 Fetch data from the cursor
8	=directory@p("D://txt1//")	=[]	The file in txt1 folder is big
9	for A8.len()		
10		=file(A8(A9))	
11		=B10.cursor@t(;",")	
12		>B8=B8|B11	
13	=B8.conj@xm()		speed up data retrieval
14	=A13.fetch()		

Class	StudentID	Subject	Score
Class one	1	English	84
Class one	2	English	81
Class one	3	English	75
Class one	1	Math	77
Class one	2	Math	80
Class one	3	Math	86
Class one	1	PE	69
Class one	2	PE	97
Class one	3	PE	67

Related concepts:

A.merge()

CS.merge@x()

CS.pjoin()

connect()

Description:

Create a connection to a database.

Syntax:

connect (*dataSource*)

Remark:

Establish the data source connection

Parameters:

dataSource	The name of a data source

Options:

@e	If there is an error, the error message returned will be processed by code by itself;without the option, an interrupt occurs.

Return value:

Data source connection

Example:

	A	
1	=connect("demo")	Connect the data source **demo**. If any database operation error occurs, the data source connection will be interrupted.
2	=connect@e("demo")	Connect the data source **demo**. For any operation on this connection, if any error occurs, a message will be returned and processed by code.

Related concepts:

db.close()

db.error()

db.commit()

db.rollback()

cos()

Description:

Compute the cosine value

Syntax:

cos(*numberExp*)

Remark:

The parameter *numberExp* is defined in radians.

Parameters:

numberExp	The radian number of the cosine to be computed

Return value:

Float type

Example:

–	**cos(pi())**	**-1.0**
–	**cos(pi(2))**	**1.0**

Related concepts:

sin()

tan()

count()

A.count(*x*)

Description:

Compute x with each member of the sequence and then count the number of non-null sequence members of the new sequence

Syntax:

A.**count**(*x*)	Equivalent to *A*.(*x*).**count**()

Remark:

Compute *x* against *A* by loop and return number of the records that can make *x* not null.

Parameters:

A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names

Return value:

An integer

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.count(HIREDATE)	Count the number of records whose HIREDATE is not null
3	=A1.(SALARY+1000).count()	Add 1,000 to each value of SALARY and then count the number of records

Related concepts:

A.count()

A.count()

Description:

Count the number of non-null members in a sequence.

Syntax:

A.count() Equivalent to **count**(x_1,…,x_n)

Remark:

Count the number of the non-null members in the sequence *x*.

Parameters:

A An *n* sequence

Return value:

The integer which is the number of the non-null members in the sequence *A*

Example:

	A	
1	=[1,2,3,4].count()	4
2	=[1,null,3,4].count()	3
3	=count(1,null,3,4)	3

Related concepts:

A.sum()

A.avg()

A.min()

A.max()

A.variance()

A.count()

mdb.count()

Description:

Query the data of a MongoDB, count them and return the result in the form of a numerical value

Syntax:

mdb.**count**(*c*,*f*)

Remark:

Select the data satisfying *f* from table *c* in a MongoDB, count them and return the result in the form of a numerical value

Parameters:

mdb Connection to MongoDB

c The table name

f Filtering condition, which is the same as the syntax of MongoDB

Return value:

A numerical value

Example:

	A	
1	=mongodb("mongo://127.0.0.1:27017/myTest?user=root &password=sa")	
2	=A1.count("Score","{score:{$gt:95}}")	Get the number of the members whose scores are greater than 95 from table Score
3	=A1.close()	Close the database connection

Related concepts:

mongodb()

mdb.close()

mdb.find()

mdb.distinct()

mdb.aggregate()

countif()

A.countif()

Description:

Locate all the positions of a member in a sequence, get the intersection of these positions and count the members in the common positions in another sequence.

Syntax:

A.**countif**(A_i:x_i,…)

Remark:

Locate all the positions of member or sub sequence x_i in A_i, get the intersection of these positions and return the count of the non-null members of *A* in these positions

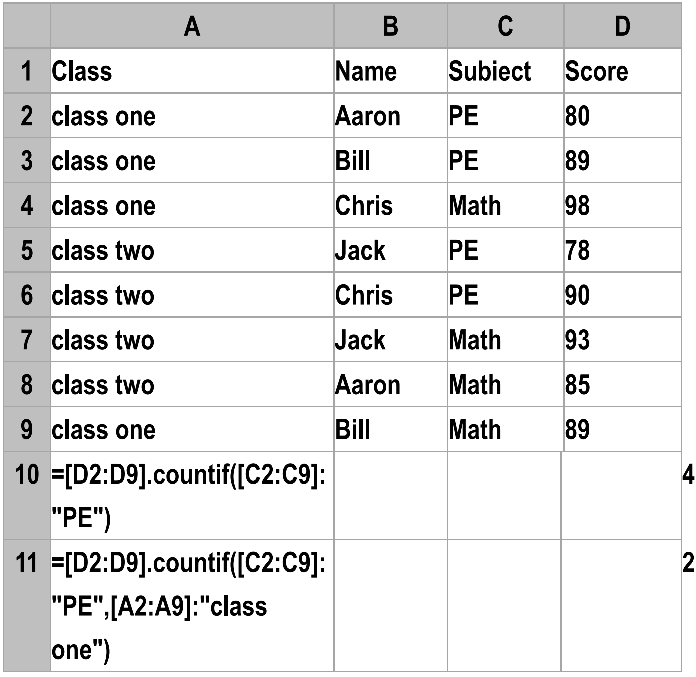
Parameters:

A_i	A sequence
x_i	Members in A_i
A	The target sequence

Return value:

The count of the non-null members of *A* in those result positions

Example:



	A	B	C	D	
1	Class	Name	Subiect	Score	
2	class one	Aaron	PE	80	
3	class one	Bill	PE	89	
4	class one	Chris	Math	98	
5	class two	Jack	PE	78	
6	class two	Chris	PE	90	
7	class two	Jack	Math	93	
8	class two	Aaron	Math	85	
9	class one	Bill	Math	89	
10	=[D2:D9].countif([C2:C9]:"PE")				**4**, the search with a single condition
11	=[D2:D9].countif([C2:C9]:"PE",[A2:A9]:"class one")				**2**, the search with multiple conditions

Related concepts:

A.sumif(A_i:x_i,…)

A.avgif(A_i:x_i,…)

A.minif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

create()

create(*F*$_i$, …)

Description:

Create an empty table sequence.

Syntax:

create(*F*$_i$, …)

Remark:

Create an empty table sequence taking *F*$_i$, … as its fields

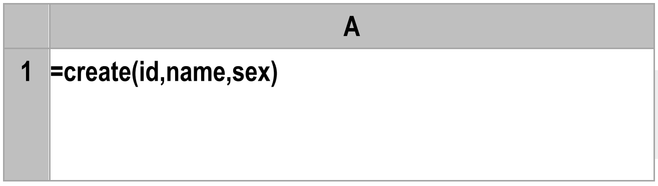
Parameters:

F_i	Field name

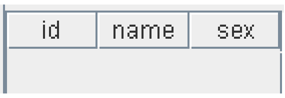
Return value:

An empty table sequence

Example:



	A
1	=create(id,name,sex)



id	name	sex

T.create()

Description:

Create a new empty table sequence by duplicating the data structure of table sequence *T*.

Syntax:

T.create()

Remark:

If *T* has a primary key, then duplicate the primary key at the same time.

Parameters:

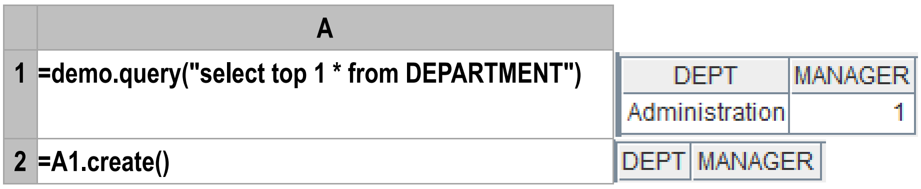
T A table sequence

Return value:

A new empty table sequence

Example:

➢ Create from a normal table sequence



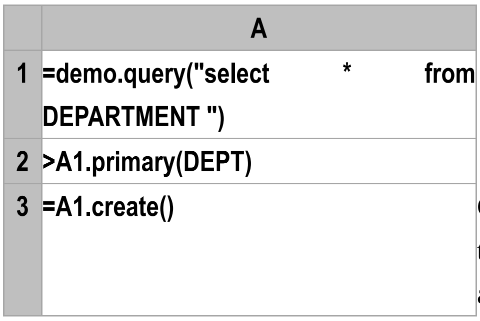
	A
1	=demo.query("select top 1 * from DEPARTMENT")
2	=A1.create()

DEPT	MANAGER
Administration	1

DEPT	MANAGER

Create an empty table sequence **A2** through "**=A1.create ()**", and this empty table sequence has the same data structure as table sequence **A1**. They are all composed of field **DEPT** and field **MANAGER**

➢ Copy the primary key



	A
1	=demo.query("select * from DEPARTMENT ")
2	>A1.primary(DEPT)
3	=A1.create()

Create a new empty table sequence that has not only the same data structure with table sequence **A1**, but also the same primary key

cursor()

F.cursor()

Description:

Create and return a cursor based on the sequence of binary file objects

Syntax:

F.cursor()

Remark:

Create and return a cursor based on sequence *F* of binary file objects . Each binary file object has only one field. Generate a cursor by joining these fields together and its length is determined by the shortest member cursor. When retrieving files from a directory which is only allowed to have binary files with a single field, an error will occur if there are ones with more than one field.

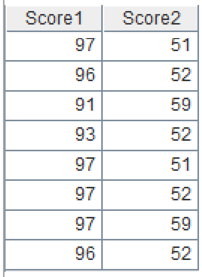
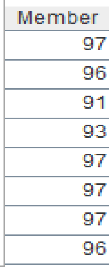
Parameters:

F	A sequence of binary file objects

Return value:

Cursor

Example:

	A	B	
1	=directory@p("D://txt//")	=[]	Retrieve all binary files under the directory
2	for A1.len()		
3		=file(A1(A2))	
4		>B1=B1|B3	Save the circularly retrieved file objects in B1 cell
5	=B1.cursor()		
6	=B1.cursor()		
7	=A5.fetch()		Retrieve number from cursor A5  Score1 Score2 97 51 96 52 91 59 93 52 97 51 97 52 97 59 96 52
8	=A6.fetch().field(1)		Get the first field value from cursor A6  Member 97 96 91 93 97 97 97 96

Related concepts:

cs.fetch()

cs.skip()

f.cursor()

db.cursor()

P.cursor()

Description:

Convert an in-memory RSeq to a cursor and return it

Syntax:

P.**cursor**()

Remark:

Generate a cursor on the basis of an RSeq to work with other cursor functions

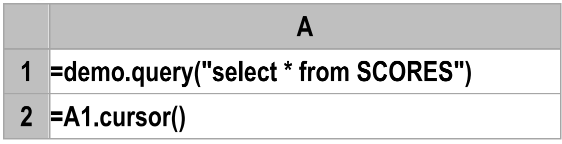
Parameters:

P	An RSeq

Return value:

A cursor

Example:



	A	
1	=demo.query("select * from SCORES")	RSeq
2	=A1.cursor()	Return a cursor

db.cursor()

Description:

Create a database cursor by executing an SQL statement and return it.

Syntax:

db.**cursor**(*sql* {,*args* …})

Remark:

Create a database cursor by executing SQL and return it. In the function, *db* is a database connection.

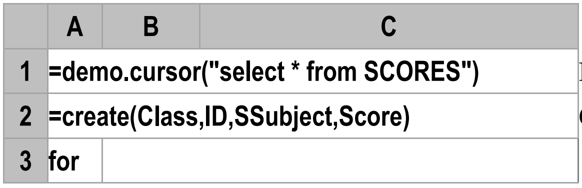
Parameters:

db	Database connection
sql	An SQL statement; for example, **select * from tabl**e
args	If there are parameters in the SQL,they must get assigned; their values can be existing ones or *args* specified in the statement. Note: Parameters shall be separated by commas.

Return value:

A cursor

Example:



	A	B	C	
1	=demo.cursor("select * from SCORES")			Return a fetch cursor
2	=create(Class,ID,SSubject,Score)			Create a table sequence
3	for			

4		if A3==1	=A1.skip(5)	When the loop number is **1**, jump five rows forward
5		=A1.fetch(3)		Fetch three records from the cursor **A1** at once
6		if B5==null		If **B5** is empty, break the loop
7			break	
8		else		
9			>A2.insert(0:B5,CLASS,STUDENTID,SUBJECT,SCORE)	Insert the records in B5 into **A2**

Related concepts:

cs.fetch()

cs.skip()

f.cursor()

Description:

Create a cursor according to a file and return it

Syntax:

f.cursor()

f.cursor(F_i:*type*,…;*s*,*b*:*e*)

Remark:

Create a cursor according to file *f* and return it.

Parameters:

f File object, only support the textual file object.

F_i Fields retrieved, and all fields will be retrieved by default.

type Field types, including bool, int, long, float, decimal, number, string, date, time and datetime. Data type of the first row will be used by default.

s Customizable separator. The default separator is tab.

b The beginning character. Omitting *b* indicates retrieving characters from the first to the e^{th}. The row, which is not filled with characters to the full, shall still be regarded as one row. Without @t option, the value of *b* must be 0 or 1 when retrieving the first row.

e The ending character. Omitting *e* indicates retrieving characters from the b^{th} to the last. The row, which is not filled with characters to the full, shall still be regarded as one row. The ":" cannot be omitted. If *e* is greater than the actual number of rows, then the actual number of rows shall prevail.

If omitting both *b* and *e*, retrieve data from the first byte to the last one. In this case, "," can be omitted.

Options:

@t Use the first row of *f* as the field names. If omitting this option, then use_1,_2,… as the field names

@b Retrieve data from the exported binary file, with the support for parameter F_i, *b* and *e*, and with no support available for parameters *type* and *s*. *@t* will be ignored. Here the binary file is the one stored in the unit of data block and all data of the block where both *b* - the beginning character and *e* - the ending character - falls in, will be retrieved

@z It will roughly divide the file into *e* parts and the part *b* will be retrieved. This option applies to both the text file and the binary file. The actual starting/ending position will be auto-computed programmatically to ensure the integrity of the records. For binary files exported with f.export@g(A,x:F,…;x...), the option also ensures that the exported records in one group will not be split apart.

@x Delete files automatically on closing

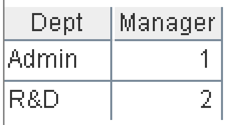
@s Not split the fields; import the file as a TSeq consisting of strings of a single field and ignore the parameters.

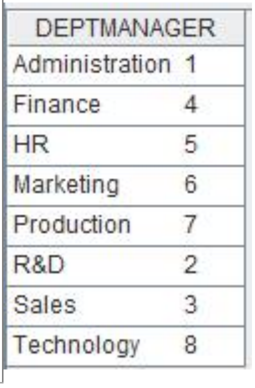
@m Using multithreads will increase data retrieval speed. Members of the result set haven't an definite order and this option will be ignored when parameters *b* and *e* exist. The option is often used to retrieve data from big files. There should be more than one parallel for registration code option and configuration.

Return value:

Cursor

Example:

	A	B	C	
1	=file("D://Student.txt").cursor@tx()			Return the cursor for retrieving data, and take the record of the first row as the field names, and delete the file automatically when closing the cursor.
2	=create(Class,ID,SSubject,Score)			Construct a new TSeq
3	for			
4		if A3==1	=A1.skip(5)	If the number of loop is 1, then skip 5 rows consecutively.
5		=A1.fetch(3)		Retrieve data from cursorA1, 3 rows each time
6		if B5==null		Jump out from the loop when B5 is null
7			break	
8		else		
9			>A2.insert(0:B5,Class,StudentID,Subject,Score)	Insert records in B5 into A2
10	=file("D://Department.txt").cursor@t(Dept,Manager;"/",1:25)	=A10.fetch()		 Dept Manager Admin 1 R&D 2 Contents of **Department. txt** are separated with the slashes and read out according to the specified fields of **Dept** and **Manager**.
11	=file("D://Department5.txt").cursor@t(;,1:25)	=A11.fetch()		With the picked out fields and delimiter being omitted, get the

			records from the 1st byte to the 25th byte.
12	=file("D:// EMPLOYEE.txt").cursor@b(GENDER;,1:128)	=A12.fetch()	Retrieve GENDER field and the data from the first byte to the 128th byte from the binary file EMPLOYEE.txt. Since the 128th byte falls in the first data block, all data of this block will be retrieved.
13	=file("D://EMPLOYEE.txt").cursor@zb(;,1:2)	=A13.fetch()	Retrieve the binary file **EMPLOYEE.txt,** which is exported in the example of f.export@g(A,x:F,…;x…), divide the file contents into 2 shares, and get the first share. The same group of exported records will not be split apart
14	=file("D:\\orders.txt").cursor@mt(;",")	=A14.fetch()	As the file is big, the data retrieval speed should be increased. The order in the result is not the same as the record order in the file.
15	=file("D:\\Department.txt").cursor@ts()	=A15.fetch()	 DEPTMANAGER Administration 1 Finance 4 HR 5 Marketing 6 Production 7 R&D 2 Sales 3 Technology 8

Related concepts:

cs.fetch()

cs.skip()

db.cursor()

date()

date(*datetimeExp*)

Description:

Get the date part of the datetime value

Syntax:

date(*datetimeExp***)**

Remarks:

Get the date part of the *datetimeExp*

Parameter:

datetimeExp	Datetime

Return value:

Date

Example:

– **date(now())**	**2013-12-09**

Related concepts:

date()

datetime(*datetimeExp*)

time(*datetimeExp*)

datetime()

time()

date()

Description:

Convert a string or an integer to a date

Syntax:

date(*stringExp,format***)** Convert the type of *stringExp* to date according to the format defined by *format*

date(*stringExp***)** The format of the result returned by *stringExp* should be in consistent with the date format in configuration information; if time is contained in the result, the time will not be converted

date(*year,month,day***)** Convert *year,month,day* of integer type to date type

Remark:

Convert the string *stringExp* or integer *year,month,day* to date

Parameters:

format	Format string
stringExp	String expression
year	Integer
month	Integer
day	Integer

Return value:

Date

Example:

– **date("1982-08-09")**	**1982-08-09**
– **date("1982-08-09 10:20:30")**	**1982-08-09**
– **date(1982,08,09)**	**1982-08-09**
– **date(1982,-8,09)**	**1981-04-09**

- **date(1982,18,09) 6/9/1983**
- **date("12/28/1972","MM/dd/yyyy") 1972-12-28**

Related concepts:

date(*datetimeExp*)

datetime(*datetimeExp*)

time(*datetimeExp*)

datetime()

time()

f.date()

Description:

Return the time and date of a file last modified.

Syntax:

***f*.date()**

Remark:

Return the time and date of file *f* last modified

Parameter:

f File object

Example:

	A
1	=file("D://p1.dfx").date()

"2013-05-02 09:45:08"

Related concept:

f. exists()

f. size()

movefile()

datetime()

datetime(*datetimeExp*)

Description:

Adjust the precision of datetime expression and then return the expression

Syntax:

datetime(*datetimeExp***)**

Remarks:

Adjust the precision of *datetimeExp* and then return it. By default, the precision is the day

Parameter:

datetimeExp Datetime value

Options:

@m Measure to minute

@s Measure to second

Return value:

Datetime value

Example:

– **datetime(now())**	**2013-12-09 00: 00: 00**
– **datetime@m(now())**	**2013-12-09 16:56: 00**
– **datetime@s(now())**	**2013-12-09 16:56:45**

Related concepts:

date()

date(*datetimeExp*)

time(*datetimeExp*)

datetime()

time()

datetime()

Description:

Convert the string or long integer to date/time

Syntax:

datetime(*string*{, *format* }**)** Convert the data type of *string* to date/time according to the format defined by *format*; if parameter *format* doesn't exist, the format of *string* of string type should be the same as the data type of date and time in configuration information

datetime(*long***)** Convert *long* of long integer type to date/time

datetime(*date*,*time***)** Concatenate date type data and time type data into data of date/time type

datetime(*y*,*m*,*d*,*h*,*m*,*s***)** Convert *y*,*m*,*d*,*h*,*m*,*s* of integer type to date/time data

Remark:

The format of *string* should match *format*

Parameters:

string	String
format	Format string
long	Long integer counted in microseconds
date	Date type
time	Time type
y	Positive integer, year
m	Positive integer, month
d	Positive integer, day
h	Positive integer, hour
m	Positive integer, minute
s	Positive integer, second

Return value:

Date/time

Example:

– **datetime("2006-01-01 10:20:30")**	**2006-01-01 10:20:30**
– **datetime("12/28/1972 10:23:43","MM/dd/yyyy hh:mm:ss")**	**1972-12-28 10:23:43**
– **datetime("2006-01-01 10:20:30:111")**	**2006-01-01 10:20:30**

–	**datetime(12345)**	**1970-01-01 08:00:12**
–	**datetime(date("1982-08-09"),time("12:12:12"))**	**1982-08-09 12:12:12**
–	**datetime(2006,01,01,-10,-20,30)**	**2005-12-31 13:40:30**

Related concepts:

date()

time()

date(*datetimeExp*)

datetime(*datetimeExp*)

time(*datetimeExp*)

day()

Description:

Get the day from a specified date

Syntax:

day(*dateExp***)**

Remark:

Get the day from *dateExp* of date type

Parameters:

dateExp Date expression whose result must be the date or the string of Chinese date and time format

Options:

@w Get the day of the week from the specified date. For Sunday, return 1; For Monday, return 2, and so on. By default, get the day of the month from the specified date.

Return value:

Integer

Example:

–	**day(datetime("19800227","yyyyMMdd"))**	**27**
–	**day(datetime(12345))**	**1**
–	**day(datetime("2006-01-15 10:20:30"))**	**15**
–	**day@w(datetime("19800227","yyyyMMdd"))**	**4**
–	**day@w(datetime("2006-01-15 10:20:30"))**	**1**

Related concepts:

year()

month()

hour()

minute()

second()

millisecond()

days()

Description:

Get the number of days of the year, quarter or month to which the specified date belongs

Syntax:

days(*dateExp***)**

Remark:

Get the number of days of the year, quarter or month to which the specified date *dateExp* belongs

Parameters:

dateExp	A date or a date string of standard format

Options:

@q	Get the number of days of the quarter to which the specified date belongs
@y	Get the number of days of the year to which the specified date belongs
	By default get the number of days of the month to which the specified date belongs

Return value:

An integer

Example:

– **days(datetime("19800227","yyyyMMdd"))**	**29**
– **days(datetime("2006-01-15 10:20:30"))**	**31**
– **days@y(datetime("19800227","yyyyMMdd"))**	**366**
– **days@q(datetime("2006-01-15 10:20:30"))**	**90**

decimal()

Description:

Convert a string or numeric value to a big decimal number.

Syntax:

decimal(*stringExp***)**

decimal(*numberExp***)**

Remark:

The parameter *stringExp* must be a string that consist of a number (and sometimes a decimal point).

The parameter *numberExp* must be a numeric value which is less than or equal to 64 bit. For value more than 64 bits, you will have to use the *stringExp* to replace the *numberExp*.

Parameters:

stringExp	A specified numeric string, which may contain a decimal point.
numberExp	The numeric value which is less than or equal to 64 bit.

Return value:

Big decimal number

Example:

– **decimal("123456789012345678901234567890")**	**123456789012345678901234567890**
– **decimal(1234567890123456)**	**1234567890123456 (BigDecimal type)**

Related concepts:

float()

int()

long()

number()

string()

delete()

T.delete()

Description:

Delete specified records from a table sequence.

Syntax:

T.**delete**(*k*)	Delete the k^{th} record
T.**delete**(*p*)	Delete the records whose sequence numbers exist in *p*
T.**delete**(*A*)	Delete the records that exist in *A*

Remark:

Delete the specified records from *T*. The deleted records will be saved in the delete buffer.

Parameters:

T	A table sequence
k	A positive integer, which specifies the position of a record to be deleted
p	An *n* integer sequence, which specifies the positions of the records to be deleted
A	A sequence, which specifies the records to be deleted

Return value:

The table sequence *T*, some records of which have been deleted

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.delete(1)	Delete the first record
3	=A1.delete([2,4,6])	Delete the second, the forth and the sixth records.
4	=A1.select(EID>5)	
5	=A1.delete(A4)	Delete the records whose **EID** is more than **5**.

Note:

We use the store address, instead of the field name or field value, to judge whether a record of *T* is in *A*. So if you want to delete the records which fulfil certain conditions, you shall generally use such syntax: *T*.**delete**(*T*.**select**(…)).

Related concepts:

T.modify()

T.insert()

A.delete()

A.delete()

Description:

Delete specified members from a sequence.

Syntax:

***A*.delete(*k*)**	Delete the k^{th} member
***A*.delete(*p*)**	Delete the members whose sequence numbers exist in *p*

Remark:

Delete the k^{th} member or the members whose sequence numbers exist in *p* from sequence *A*.

Parameters:

A	A sequence
k	A positive integer that indicates the position of a member to be deleted in the sequence
p	An *n* integer sequence that defines the positions of the members to be deleted

Options:

@n	*A* is not changed, and only the new sequence is returned

Return value:

A sequence

Example:

	A	
1	=["a","c","d","e","f"]	
2	=A1.delete@n(2)	[a,d,e,f], but **A1** is not changed
3	=A1.delete([2,4,5])	[a,d], **A1** is changed

Related concepts:

A.insert()

A.modify()

T.delete()

deq()

Description:

Judge if two dates are the same

Syntax:

deq (*datetimeExp1*,*datetimeExp2***)**

Remark:

Compare the two parameters - *dateExp1* and *dateExp2* - to see if they are the same

Parameters:

datetimeExp1	Date or standard datetime format string such as yyyy-MM-dd HH:mm:ss, yyyy-MM-dd, or HH:mm:ss
datetimeExp2	Date or standard datetime format string such as yyyy-MM-dd HH:mm:ss, yyyy-MM-dd, or HH:mm:ss

Options:

@y	Precise to the year

@q	Precise to the quarter
@m	Precise to the month
@t	Precise to the ten-day period
@w	Precise to the week
	Precise to the day by default

Return value:

A Boolean value

Example:

– deq("1988-12-08","1988-12-07")	false
– deq@y(date("1988-11-08"),date("1988-09-12"))	true
– deq@m(date("1988-11-08"),date("1988-09-12"))	false
– deq@q(date("1988-12-08"),date("1988-10-12"))	true
– deq@t(date("1988-10-08"),date("1988-10-12"))	false
– deq@w(date("1988-10-05"),date("1988-10-08"))	true

derive()

A.derive()

Description:

Add one or more fields to a TSeq/RSeq

Syntax:

A.**derive** (x_i :F_i,…)

Remark:

Add *Fi*,… field to TSeq/RSeq *A* to re-structure it to "original fields in *A* plus F_i,…". Then traverse every record of *A*, and assign each F_i with value x_i.

Parameter:

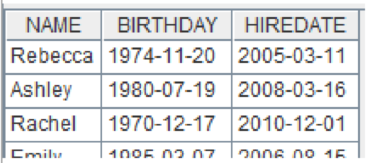
F_i	The field name. In this case, F_i cannot have the same name as the original fields in *A*
x_i	Expression, whose computational results are used as the field values
A	TSeq/RSeq

Return value:

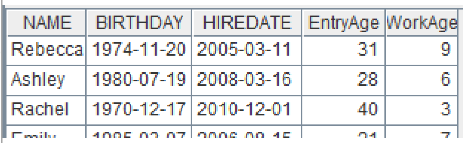
The TSeq/RSeq with added field(s)

Example:

	A
1	=demo.query("select NAME,BIRTHDAY,HIREDATE from EMPLOYEE")
2	=A1.derive(interval@y(BIRTHDAY, HIREDATE):EntryAge, age(HIREDATE):WorkAge)



NAME	BIRTHDAY	HIREDATE
Rebecca	1974-11-20	2005-03-11
Ashley	1980-07-19	2008-03-16
Rachel	1970-12-17	2010-12-01
Emily	1985-03-07	2006-08-15



NAME	BIRTHDAY	HIREDATE	EntryAge	WorkAge
Rebecca	1974-11-20	2005-03-11	31	9
Ashley	1980-07-19	2008-03-16	28	6
Rachel	1970-12-17	2010-12-01	40	3
Emily	1985-03-07	2006-08-15	21	7

Note:

The difference between new() and derive(): The new() is to newly structure a TSeq without changing the original one. By comparison, the derive() is to copy the original fields and then add fields.

cs.derive()

Description:

Add one or more fields to a cursor

Syntax:

cs.**derive**(x_i :F_i,…)

Remark:

Add *Fi*,… fields to cursor *cs*, so as to form a cursor with the structure of "original fields in *cs*, F_i…". Then, transverse every record in *cs*, and assign the value x_i to each F_i.

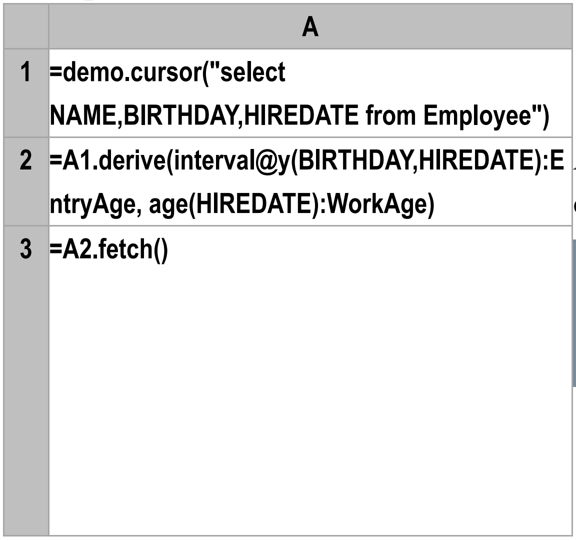
Parameter:

cs	Cursor
F_i	Field name. In this case, F_i cannot be of the same name as the existing fields in *cs*
x_i	Expression, whose computed results are the field values

Return value:

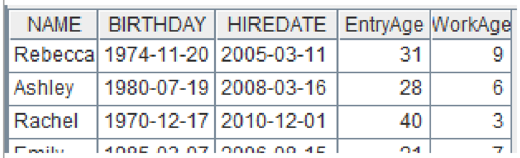
Cursor

Example:



	A
1	=demo.cursor("select NAME,BIRTHDAY,HIREDATE from Employee")
2	=A1.derive(interval@y(BIRTHDAY,HIREDATE):EntryAge, age(HIREDATE):WorkAge)
3	=A2.fetch()

Add fields EntryAge and WorkAge to the original cursor.



NAME	BIRTHDAY	HIREDATE	EntryAge	WorkAge
Rebecca	1974-11-20	2005-03-11	31	9
Ashley	1980-07-19	2008-03-16	28	6
Rachel	1970-12-17	2010-12-01	40	3
Emily	1985-03-07	2006-08-15	21	7

Related concepts:

cs.new()

dfx …

Description:

With esProc JDBC, search for and execute a program file locally. If not found, then search the server.

Syntax:

dfx …

Remark:

Pass in the parameter … and call the cellset file *dfx* waiting to be executed. Firstly, locate the cellset file with relative searching path for execution. If not found, then search the server (The server list is configured in dfxConfig.xml file). Once the computation is completed in esProc or server, return a sequence composed of members of the result set; if there are multiple result sets, then return a sequence composed of these result sets. **dfx …** can be regarded as the simple notation of **call *dfx*(…)**, which can be executed with ***st*.executeQuery()** and returns the result set.

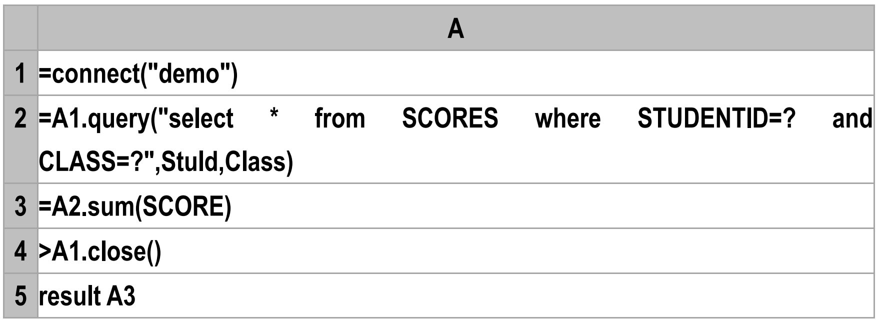
Parameter:

dfx Cellset file, with relative addressing path or absolute path

… Parameter. Use comma to separate if there are multiple parameters. The parameter names for receiving parameters will not be used. Instead, they will be assigned with value in the order of parameters.

Example:

C: \\test.dfx cellset file has the following contents, in which **StuId** and **Class** is the cellset parameters.



	A
1	=connect("demo")
2	=A1.query("select * from SCORES where STUDENTID=? and CLASS=?",StuId,Class)
3	=A2.sum(SCORE)
4	>A1.close()
5	result A3

Test code is shown below:

```
public void testDataServer() {
        Connection con = null;
        com.esproc.jdbc.InternalCStatement st;
        try{
            Class.forName("com.esproc.jdbc.InternalDriver");
            con= DriverManager.getConnection("jdbc:esproc:local://");
            // Call the stored procedure in which test is the file name of dfx
            st =(com.esproc.jdbc.InternalCStatement)con.prepareCall("test ?,?");
            // Set the first parameter
            st.setObject(1,"4");
           // Set the second parameter
            st.setObject(2,"Class one");
            // Execute stored procedure
            st.execute();
            // Get result set
            ResultSet set = st.getResultSet();
            // Print results
            printRs(set);
            // The following statement leads to the same result as the above-mentioned
method of calling
```

```
            st =(com. esproc.jdbc.InternalCStatement)con.prepareCall("test 4,\ "Class
one\"");
            st.execute();
            set = st.getResultSet();
            printRs(set);
        }
        catch(Exception e){
            System.out.println(e);
        }
        finally{
            // Close the connection
            if (con!=null) {
                try {
                    con.close();
                }
                catch(Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
```

diff()

A.diff()

Description:

To remove from the first sub-sequence the members that also exist in the other sub-sequences of the sequence

Syntax:

A **.diff()**

Remarks:

Generally sequence *A* contains multiple sub-sequences. The function creates a new sequence by getting the difference of *A*'s first sub-sequence and the other sub-sequences, ensuring that the new sequence doesn't include any member of the other sub-sequences.

The operation is to compute the difference of the first sub-sequence and the second one, then compute the difference of the result and the third sub-sequence, and so on and so forth.

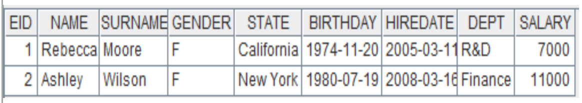
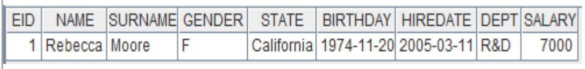
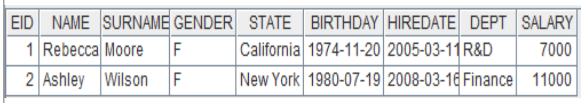
Parameters:

A	A sequence whose members are sequences

Return value:

The new sequence created by performing difference operation on A's first sub-sequence and the other sub-sequences.

Example:

	A	
1	=[[1,2,3,4,5],[3,7,8]].diff()	[1,2,4,5]
2	=[[1,2,3],[3,2],1].diff()	[]
3	=[[1,2,2,3],2].diff()	[1,2,3] remove only one of the duplicate members
4	=demo.query("select top 2 * from EMPLOYEE")	
5	=demo.query("select top 1 * from EMPLOYEE")	
6	=[A4,A5].diff()	 Since A4 and A5 come from different TSeqs and have different store addresses, so the same records are regarded as different members

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000

Note:

If the sub-sequences are **RSeqs**, we can judge if they are duplicate by their store addresses.

Related concepts:

A.diff(x)

A.union()

A.conj()

A.isect()

A.diff(*x*)

Description:

Perform a certain operation on a sequence so as to remove from the first sub-sequence the members that also exist in the other sub-sequences.

Syntax:

A.**diff(x)**

Remark:

Generally sequence *A* contains multiple sub-sequences. Compute x against each sub-sequence of *A* by loop to create a new sequence, get the difference of its first sub-sequence and the other sub-sequences, ensuring that the result doesn’t include any member of the other sub-sequences.

The operation is to compute the difference of the first sub-sequence and the second one, then compute the difference of the result and the third sub-sequence, and so on and so forth.

Parameters:

A	A sequence whose members are sequences
x	An expression that returns a sequence

Return value:

A sequence

Example:

	A	
1	=demo.query("select * from EMPLOYEE where GENDER = 'M' order by NAME")	
2	=demo.query("select * from EMPLOYEE where GENDER = 'F' order by NAME")	
3	=[A1,A2].diff(~.(NAME))	Remove only one of the duplicate members

Related concepts:

A.diff()

dims()

Description:

Generate a piecewise dimension table space.

Syntax:

dims(*h*;*V*,*g*;…**)**

Remark:

Generate a piecewise dimension table space. Divide the dimension table *V* into several sections according to the piecewise function *g*. Each node machine will be used to store a part of dimension table. Users can specify multiple dimension tables of V. In the parallel computing, all used dimension tables must be stored in the dimension table space. They can all be generated using dims, or can be appended to the dimension table space with *gf()* function as required.

Parameters:

h	Node machine sequence
V	Name of dimension table. Typically, it is a global variable name, and the variable value is just this dimension table.
g	The piecewise function expression taking the primary key of dimension table as the parameter. According to the primary key value, the node machine holding the corresponding section of the piecewise dimension table will be found, and the sequence number of this node machine in the node machine sequence *h* will be returned. In the conditional expression, the primary key is represented by ?. For multiple primary keys, use ?1,?2 to represent them. If there are altogether *k* node machines in the node machine list *h*, then the computational result of piecewise function *g* should be any integer bewtween 1 and *k*.

Return value:

Dimension table space

Example:

DimEmployee.dfx(**ID1** is the cellset parameter)

	A
1	=connect("demo")

2	=@DimEmployee=A1.query("SELECT * FROM EMPLOYEE where EID in (?)",ID1)
3	=A1.close()
4	result A2

DimGymscore.dfx(**ID1** is the cellset parameter)

	A
1	=connect("demo")
2	=A1.query("SELECT * FROM GYMSCORE ID where ID in (?)",ID1)
3	=A2.primary(NAME,EVENT)
4	=@DimGymscore=A2
5	=A1.close()
6	result A4

	A	
1	=["192.168.0.232:8281","192.168.0.147:8281"]	Get the idle machine
2	=to(200)	
3	=to(201,300)	
4	=callx@a("DimEmployee.dfx",[A2:A3];A1)	
5	=callx@x("DimGymscore.dfx",[[1,2,3,4,8,11],[5,6,7,9,10,12]];A1)	
6	=dims(A1;@DimEmployee,if(?<=200,1,2);@DimGymscore,if(?1=="Ana Silva" || ?2=="Vault",1,2))	On the node machine in A1, create the piecewise dimension table **@ DimEmployee** and **@DimGymscore**. The piecewise formula is **" if(? <36,1,2)"** and **"if(? 1=="Ana Silva" || ? 2=="Vault",1,2) "**. When a certain TSeq requires that the primary key of dimension table is used for referencing the data of this table, the records will be queried according to the primary key of the dimension table
7	=create(name,event).record(["Ana Silva","Vault"])	
8	=A6.fetch(A7:name:event,@DimGymscore,NAME,SCORE)	According to the defined relation in the dimension table space **A6**, make the values of fields **name** and **event** in the sequence **A7** correspond to the two primary keys of the piecewise dimension table in order to get the data from piecewise

		dimension table **@DimGymscore**
9	**=A6.fetch(to(101),@DimEmployee,EID,NAME+" "+SURNAME:Name,DEPT)**	Get the records according to the query criterion of matching sequence **to(101)** to the primary key of piecewise dimension table **@DimEmployee**,

directory()

directory(*path*)

Description:

List the file name(s) satisfying a wildcard path

Syntax:

directory(*path***)**

Remark:

List the file name(s) satisfying the wildcard path *path*, excluding the path names.

Parameters:

path	Wildcard path, * represents 0 or multiple characters, and ? represents a single character

Options:

@d	List the subdirectory of *path*
@p	List the returned file name, including the name of full path
@m	Create a directory
@r	Delete a directory, which must be empty.

Return value:

A sequence

Example:

	A	
1	**=directory("D://*.txt")**	Return the txt file list under the root directory on driver D
2	**=directory@d("D://tomcat5")**	List the subdirectory under the **tomcat5** directory
3	**=directory@m("D://test")**	Create test folder under the root directory on driver D
4	**=directory@r("D://test")**	Delete **D://test** directory, but the test folder must be empty

Related concepts:

f. exists()

f. size()

movefile()

directory(*path*,*z*)

Description:

List the file name(s) in the data section matching the wild card path name

Syntax:

directory(*path*,*z***)**

Remark:

List file name(s) in data section *z* satisfying the wild card path *path* in the Data section *z*, excluding the path name. The function can only be used in the node machine.

Parameter:

path	Wild card path
z	Data section name

Options:

@d	List the subdirectory of *path*
@p	Return the file name, and the returned result still have the full path name
@m	Create a directory
@r	Delete a directory, which must be empty

Return value:

A sequence

Example:

On the node machine "**192.168.0.99:9282**", the contents of cellset file **directory**.dfx is shown below. Set the cellset parameter arg1:

	A	
1	=directory("/","3")	Return the file list under the root directory of section **3**
2	=directory@d("/","3")	List the subdirectory under the root directory of section **3**
3	=directory@p(“/”,”3”)	Return the file list under the root directory of section **3**, keeping the full path names.
4	=directory@m("//test//","3")	Under the root directory of section **3**, create the file folder test
5	=directory@r("//test//","3")	Under the root directory of partition **3**, remove the file folder test, which must be empty
6	result A1,A2,A3,A4,A5	

	A	
1	=callx("directory.dfx",1;"192.168.0.99:9282")	Call cellset file

Related concepts:

directory(path)

distinct()

mdb.distinct()

Description:

Query the data of a MongoDB, filter them and return the result in the form of a sequence

Syntax:

mdb.**distinct**(*c*, *s*, *f*)

Remark:

Query field *s* in table *c* of a MongoDB, select records satisfying *f*, remove the duplicate records and return the result as a sequence

Parameters:

mdb	Connection to *MongoDB*
c	The table name
s	A field
f	Filtering condition, which is written according to the syntax of MongoDB.

Return value:

A sequence

Example:

	A	
1	=mongodb("mongo://127.0.0.1:27017/myTest?user=root &password=sa")	
2	=A1.distinct("Score","name","{score:{$gt:80}}")	Select the students whose scores are greater than 80 from table Score and remove the duplicate names
3	=A1.close()	Close the MongoDB connection

Related concepts:

mongodb()

mdb.close()

mdb.count()

mdb.find()

mdb.aggregate()

dup()

A.dup()

Description:

Copy a sequence.

Syntax:

A.**dup()**

Remark:

Copy sequence *A*.

Parameters:

A	A sequence

Return value:

A sequence

Example:

	A	
1	=[1,2,3]	
2	=A1.dup()	[1,2,3]

Related concepts:

T.dup@t()

T.dup@t()

Description:

Generate a new table sequence by copying data structure of the first record of a table sequence, and all the records of it.

Syntax:

T.**dup@t()**

Remark:

Generate a new TSeq by copying the data structure of the first record of TSeq *T*, and then copy all the records from TSeq *T* to the new TSeq.

Parameters:

T	A pure record sequence

Return value:

The new table sequence

Example:

	A	
1	=demo.query("select * from SCORES")	
2	=A1.select(SCORE>90)	Having been filtered, **A2** is an RSeq instead of a TSeq
3	=A2.dup@t()	Convert **A2** to a table sequence again
4	=create(Stuid,Subject,Score).record([100,"Math",100])	
5	=A2|A4	The structure of **A2** differs to that of **A4**. It is not a pure record sequence
6	=A5.dup@t()	Error in cell **A6**: pure record sequence is required

Related concepts:

A.dup()

end *s*

Description:

Log the error message

Syntax:

end *s*

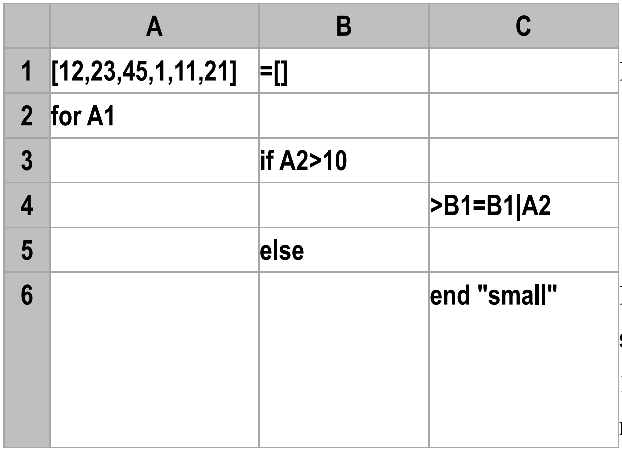
Remark:

Execute abnormal termination deliberately, throw and log the error message *s;* Without *s*, execute normal termination.

Parameters:

s Error message

Example:



	A	B	C
1	[12,23,45,1,11,21]	=[]	
2	for A1		
3		if A2>10	
4			>B1=B1|A2
5		else	
6			end "small"

B1 is **[12,23,45]**

Prompt and log an error message **"error small"** if value in A1 is not greater than 10. Terminate the program normally if no **"small"** is detected.

enum()

P.enum()

Description:

Generate a new RSeq byperforming enumeration grouping on an RSeq according to another RSeq.

Syntax:

P.**enum(***E*, *y***)**

Remark:

Generate a new RSeq by grouping the record sequence *P* according to the record sequence/sequence *E*

Parameters:

P	Record sequence to be grouped
E	Record sequence/sequence
y	A field name or an expression in *P*. *y* is allowed to be omitted

Options:

@r Repetitive enum, that is, allocate a certain record of *P* to multiple groups. This option and @n are mutually exclusive.

@p The return value is composed of the sequence numbers of members that exist in *P*

@n Group *P*’s records according to *E*’s members and return the result groups. In the result set, there is a group to store the unaligned member(s).

Function Keyword:

? It represents the value of *y*

Return value:

The new record sequence generated by grouping *P*

Example:

- *E* is a sequence

	A	
1	=["?<=60","?>=60 && ?<=90","?>=90"]	Use *y* to replace **?** in the group condition when computing
2	=demo.query("select * from SCORES")	
3	=A2.enum(A1,SCORE)	3 groups in total, and a member will not appear in multiple groups
4	=A2.enum@r(A1,SCORE)	3 groups in total; 60 and 90 are allocated to multiple groups
5	=A2.enum@p(A1,SCORE)	3 groups in total; return the sequence numbers of members that exist in A2 by group
6	=["?<=60","?>=60 && ?<=90"]	
7	=A2.enum@n(A6,SCORE)	3 groups in total, group 1 for those not greater than 60, group 2 for those greater than 60 and not greater than 90, and group 3 for the remaining values.

Note:

p.**enum()** is mainly used to conduct the enum grouping on a single record sequence. The grouping result is a sequence composed of multiple record sets, and each record set is a group. This sequence is in an order exactly equal to that of *E*. Therefore, you can retrieve the associated group information from *E* via the sequence numbers. The group result is not a record sequence, so no new data structure will be generated

Related concepts:

E.penum()

eq()

Description:

Judge if a sequence can be generated by swapping the positions of the members of another sequence.

Syntax:

A.**eq**(*B*)

Remark:

Judge if a sequence *A* can be generated by swapping the positions of the members of another sequence *B*.

Parameters:

A A sequence expression

B A sequence expression

Return value:

Boolean value.

Example:

	A	
1	= [1,2,5]	true
2	=[5,2,1]	
3	=A1.eq(A2)	true
4	=["a","b","c"]	
5	=["b","a","c"]	
6	=A4.eq(A5)	true
7	=[5,2,2]	
8	=A7.eq(A1)	false

error()

db.error()

Description:

To obtain the last error information from a database connection.

Syntax:

db.**error**()

Remark:

Return the error information corresponding to the previous command that is related to the database. **0** indicates error free; otherwise, the error code is returned. The error message can only be called for one time.

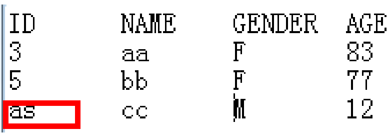
Parameters:

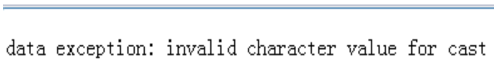
db Database connection

Options:

@m Use this option to return information in string.

Example:

	A	
1	=file("D://files//student.txt")	Data in the file **student.txt** is not correct  ID NAME GENDER AGE 3 aa F 83 5 bb F 77 as cc M 12
2	=A1.import@t()	
3	=connect@e("demo")	Create a connection and auto-control the commit and rollback operations
4	>A3.execute@k(A2,"update STUDENTS2 set NAME=?,GENDER=?,AGE=? where ID=?",NAME,GENDER,AGE,ID)	The transaction is not committed

5	=A3.error@m()	Return the error message corresponding to sql command execution
6	>file("D:\\log.txt").write@a(A5)	Append error message to the log  data exception: invalid character value for cast
7	>A3.close()	Close the connection

Related concepts:

db.close()

connect()

db.commit()

db.rollback()

eval()

Description:

Dynamically parse and compute the expression

Syntax:

eval(*StringExp* ,{*argExp*})

Remark:

The result of *StringExp* is an expression, which will be parsed and computed and whose value will be returned. In the expression that is evaluated with *argExp*, the value of the keyword **?** is the corresponding *argExp*. If there are more than one ?, then there may be more than one *argExp*. Generally there is a one-to-one correspondence between *argExp* and **?**.

If the number of **?** is more than that of *argExp*, then the extra ? will get assigned starting from the first *argExp* again.

And also, you can use the number to specify the parameter for **?**, such as **eval("?2/?1", 3, 6)**, which means the first **?** corresponds to the second parameter, the second **?** corresponds to the first parameter, so the result is **2**.

Parameters:

StringExp	An expression string to be computed
argExp	Parameter expression

Function keyword:

?	Used in *StringExp* to represent the value of *argExp*

Return value:

Result of expression. The data type will be determined by the expression

Example:

	A	
1	="1+3"	
2	=eval(A1)	4
3	=4	
4	=eval("?+5",A3)	The **?** is a key word, its value is **A3**, and the result is **9**

5	**=eval("(?+1)/?",3,4)**	The first **?** is **3**, the second **?** is **4**, the result is **1.0**.
6	**=eval("(?+?)*?",1,3)**	The first **?** is **1**, the second **?** is **3**, the third **?** is **1** again, the result is **4**.
7	**=eval("?+?",3)**	The result is **6** because the number of *argExp* is less than **?**, so *argExp* will be used repeatedly.
8	**=eval("?2/?1",3,6)**	The first **?** corresponds to the second parameter, the second **?** corresponds to the first parameter, so the result is **2.0**.

execute()

db.execute()

Description:

To execute SQL statements within the datasource.

Syntax:

db.**execute**(*sql* {,*args* …})

db.**execute**(*A*, *sql* {,*args* …}) Executions of SQL statements will be executed in BATCHSQL mode, which reduces the frequency of database access.

Remark:

Execute a SQL statement in the specified database connection. In general, this function is used to execute the SQL statements which change the records in a database (such as: update and insert).

Parameters:

db Database connection

sql A SQL statement, for example, **select * from table**

args If there are parameters in the SQL statement, they must get assigned; *args* can be either the constant value or the variable defined by expressions. Note: multiple *args* shall be separated by commas.

A A sequence; the SQL statement is executed for each member of *A*. In general, *args* is variable against each member of *A* and then passed to the SQL statement for use in execution.

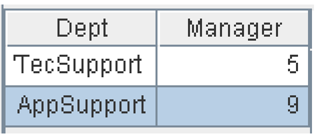
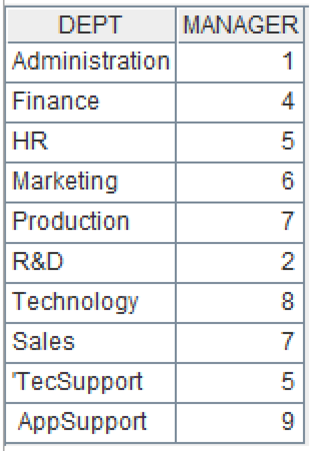
Options:

@k After the execution is completed, the transaction won't be committed. If this option is omitted, the transaction will be committed.

@s No preparation will be done for transaction processing.

Example:

	A	
1	**=demo.execute("insert into DEPARTMENT (DEPT, MANAGER)values(?,?)","TecSupport",5)**	Insert a record into the table
2	**=demo.execute("delete from DEPARTMENT**	Delete the records whose **DEPT** value equals

	where DEPT='TecSupport'")	"TecSupport"
3	=demo.execute("update DEPARTMENT set MANAGER = ? where DEPT='Sales'","7")	Modify the value of **MANAGER** of the records whose **DEPT** equals **'Sales'** into 7
4	=[["'TecSupport ",5],[" AppSupport",9]].new(~(1):Dept,~(2): Manager)	
5	=demo.execute(A4,"insert into DEPARTMENT (DEPT, MANAGER)values (?,?)", #1,#2)	**#1** and **#2** indicate the first and second columns respectively in **A4**
6	=demo.query("select * from DEPARTMENT")	

Dept	Manager
'TecSupport	5
AppSupport	9

DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6
Production	7
R&D	2
Technology	8
Sales	7
'TecSupport	5
AppSupport	9

Related concepts:

db.query()

db.proc()

exists()

f.exists()

Description:

Verify if this file exists

Syntax:

f.exists()

Remark:

Verify if file *f* exists

Parameter:

f	File object

Example:

	A	
1	=file("D://p1.dfx").exists()	Return **true** if the file exists

Related concept:

f. date()

f. size()

movefile()

exp()

Description:

***e* to the power of *n*Syntax:**

exp(*n*)

Remark:

Compute e^n

Parameters:

n	The exponent

Return value:

Numeric

Example:

- **exp(4.3)** **73.69979369959579**

Related concepts:

power(x,n)

export()

export()

Description:

Return a sequence as the string

Syntax:

export(*A*,*x*:*F*,…;*s*)

Remark:

Separate the selected fields of each record with the optional separator to generate a string and return. If no x is given, then export all fields.

Parameter:

A	A sequence
x	Fields to be exported. If omitted, then export all fields of *A*
F	Name of the result field in the string. If omitted, then use the original field name
s	Optional field separator. The default separator is tab

Options:

@t	The column name will be written to the string as the first record
@j	Import as json string, with *s* being ignored.
@x	Import as XML string, with *s* being ignored.

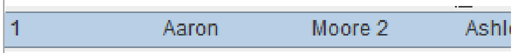
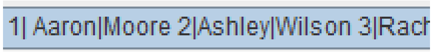
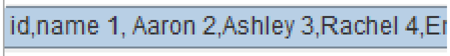
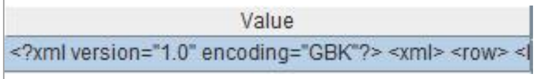
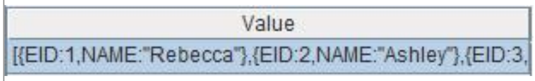
```
<xml>
    <row>
        <F>v</F>
        …
    </row>
```

```
        …
    </xml>
```

Return value:

String

Example:

	A	
1	=demo.query("select EID,NAME from EMPLOYEE")	
2	=export(A1)	 1 Aaron Moore 2 Ashl Parameters x, F, and s are omitted
3	=export(A1;"|")	 1| Aaron|Moore 2|Ashley|Wilson 3|Rach Specify the separator as "|"
4	=export@t(A1,EID:id,NAME:name;",")	 id,name 1, Aaron 2,Ashley 3,Rachel 4,Er Specify to-be-exported fields and the separator, inclusive of field names
5	=export@x(A1)	 Value <?xml version="1.0" encoding="GBK"?> <xml> <row> <I
6	=export@j(A1)	 Value [{EID:1,NAME:"Rebecca"},{EID:2,NAME:"Ashley"},{EID:3,

Note:

Format of the string: Separate the records with space, and the fields with the optional separator. The default separator is tab.

In this case, *A* must be a sequence composed of records of the same data structure

Related concepts:

f.import()

f.export()

f.export(*A*,*x*:*F*,…;*s*)

Description:

Write a sequence into a file object.

Syntax:

f.**export**(*A*, *x*:*F*,…;*s*)

Remark:

Write the sequence *A*, in the form of text, into the file object *f*. *A*'s fields that cannot be exported will be ignored. Export all fields to the file object if *x* is omitted. For the reference record r, export the r.v(). If the file object *f* doesn't exist, it will be created automatically (the directory path cannot be created automatically). The default export is a text file.

Parameters:

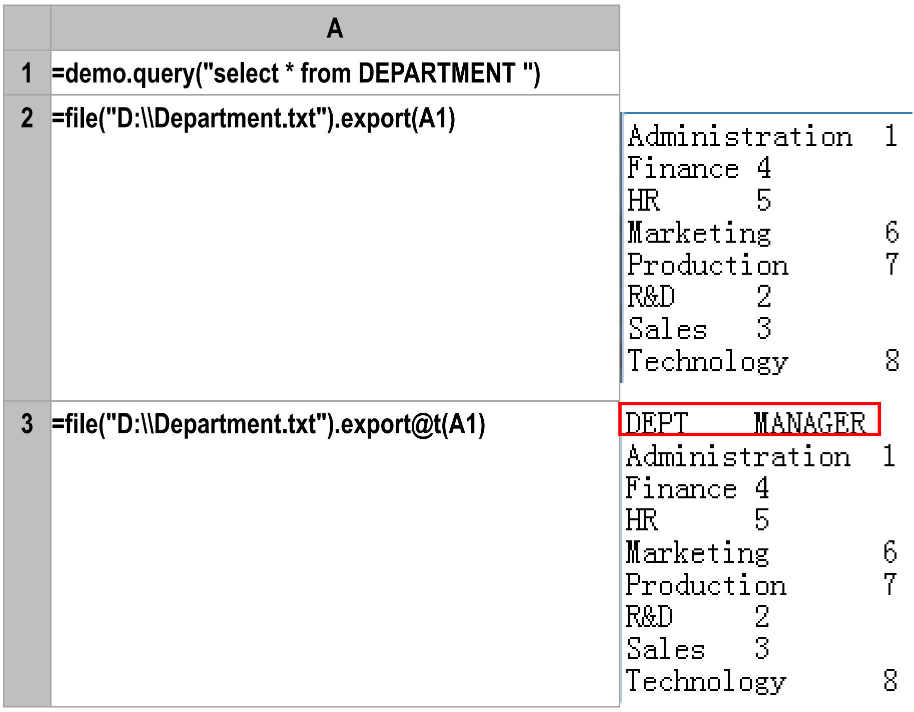
f	File object
A	The record sequence to be exported.
x	The field to be exported. If it is omitted, all the fields of *A* that can be textualized will be exported.
F	Result field name. If omitted, then use the original field name.
s	The optional separator used in the text file, and the default separator is tab.
x	The grouping expression in the **@g** option. With @g option, the *s* will be replaced.

Options:

@t	Export the first record to the file as the title.
@a	Append. If omitted, then overwrite the original file. It is exclusive to **@t**
@b	Convert into binary file to speed up the processing. The **@t** is ignored**.** It does not support *s*.
@g(…;*x*…)	*A* is ordered by the field *x*…. Data will be written into the binary file *f* in segmentation according to different groups, which is especially useful when there is a huge amount of parallel data to be exported by segments. With this option, during the exporting, the records from a same group will not be split apart.

Example:

Write a record sequence into a txt file.

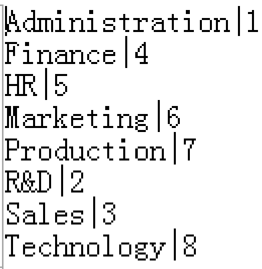
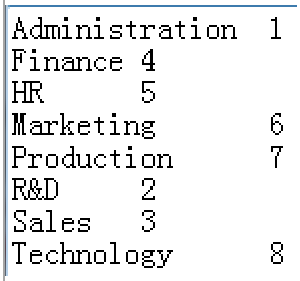
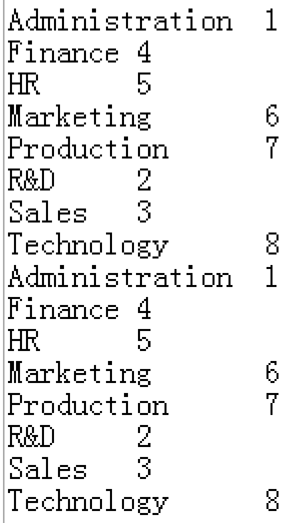
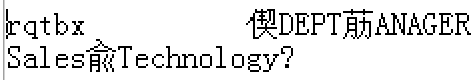
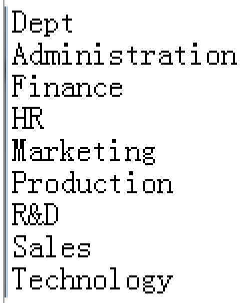


	A
1	=demo.query("select * from DEPARTMENT ")
2	=file("D:\\Department.txt").export(A1)
3	=file("D:\\Department.txt").export@t(A1)

```
Administration	1
Finance	4
HR	5
Marketing	6
Production	7
R&D	2
Sales	3
Technology	8
```

```
DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6
Production	7
R&D	2
Sales	3
Technology	8
```

Use **@t** to export the first record as the title

4	=file("D:\\Department2.txt").export(A1;"|")	 Administration|1 Finance|4 HR|5 Marketing|6 Production|7 R&D|2 Sales|3 Technology|8
5	=file("D:\\Department2.txt").export(A1;)	 Administration 1 Finance 4 HR 5 Marketing 6 Production 7 R&D 2 Sales 3 Technology 8
6	=file("D:\\Department2.txt").export@a(A1)	 Administration 1 Finance 4 HR 5 Marketing 6 Production 7 R&D 2 Sales 3 Technology 8 Administration 1 Finance 4 HR 5 Marketing 6 Production 7 R&D 2 Sales 3 Technology 8 Still the above example, append the contents from A1 to the contents of the file
7	=file("D:\\Department3.txt").export@b(A1)	 rqtbx 偰DEPT荕ANAGER Sales兪Technology?
8	=file("D:\\Department4.txt").export@t(A1,Dept:Dept1;"|")	 Dept Administration Finance HR Marketing Production R&D Sales Technology If not omitting x, then only export the specified field
9	=demo.query("select * from EMPLOYEE order by GENDER")	Sort by **GENDER**

10	=file("D:\\EMPLOYEE.txt").export@g(A9,EID,NAME,SURNAME,GENDER,SALARY;GENDER)	Export data to **EMPLOYEE.txt** file based on groups divided by **GENDER**

Note:

Format of text file: separate records with carriage return, and each fields with the optional separator. The default seperator is the tab.

In this case, the A should be a sequence comprising data of the same data structure

Related concepts:

f.import()

f.export(cs,x:F,…;s)

f.export(*cs*,*x*:*F*,…;*s*)

Description:

Retrieve data from the cursor *cs* and write them to the text file

Syntax:

f.**export**(*cs*,*x*:*F*,…;*s*)

Remark:

Write data of cursor *cs* to the file object *f* as text . If the file does not exist, create it automatically. Please note that the directory path cannot be created automatically.

Option:

@t	Export the first record to the file as the title.
@a	Append. If omitted , then overwrite the original file. It is exclusive to **@t**
@b	Use binary file to speed up the processing. The **@t** is ignored**.** It does not support *s*.
@g(…;*x*…)	*cs* is ordered by the field *x*…. The data will be written into the binary files *f* in segmentation according to different groups, which is especially useful when there is a huge amount of parallel data to be exported by segments. With this opion, during the exporting, the records from a same group will not be split apart.

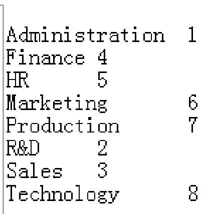
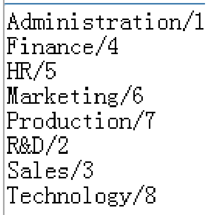
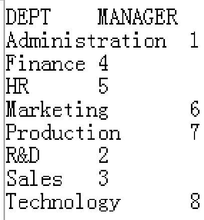
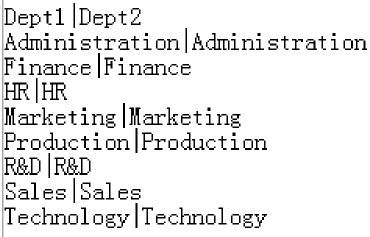
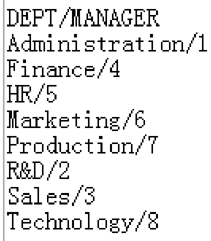
Parameters:

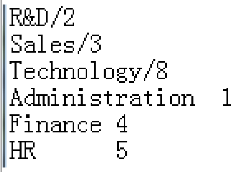
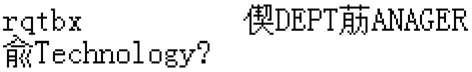
f	File object
cs	Cursor whose data to be exported
x	Fields to be exported. If omitted, then export all fields in the RSeq *A* that can be textualized
F	Result field name. If omitted, then use the original field name.
s	Optional seperator; default separator is tab
x	The grouping expression in the **@g** option. With @g option, the s will be replaced.

Example:

Write the cursor data to the text file.

	A
1	=demo.cursor("select * from DEPARTMENT")

2	=file("D:\\Department1.txt").export(A1)	 Administration 1 Finance 4 HR 5 Marketing 6 Production 7 R&D 2 Sales 3 Technology 8 tab separator
3	=demo.cursor("select * from DEPARTMENT")	
4	=file("D:\\Department2.txt").export(A3;"/")	 Administration/1 Finance/4 HR/5 Marketing/6 Production/7 R&D/2 Sales/3 Technology/8 Specify the separator “/”
5	=demo.cursor("select * from DEPARTMENT")	
6	=file("D:\\Department3.txt").export@t(A5)	 DEPT MANAGER Administration 1 Finance 4 HR 5 Marketing 6 Production 7 R&D 2 Sales 3 Technology 8 Write data to the file with the first record being the title
7	=demo.cursor("select * from DEPARTMENT")	
8	=file("D:\\Department4.txt").export@t(A7,DEPT:Dept1,DEPT:Dept2;"|")	 Dept1|Dept2 Administration|Administration Finance|Finance HR|HR Marketing|Marketing Production|Production R&D|R&D Sales|Sales Technology|Technology If x is not omitted, then export the specified fields
9	=demo.cursor("select * from DEPARTMENT")	
10	=file("D:\\Department5.txt").export@t(A9,DEPT,MANAGER;"/")	 DEPT/MANAGER Administration/1 Finance/4 HR/5 Marketing/6 Production/7 R&D/2 Sales/3 Technology/8 If F is omitted, then use the original field names
11	=demo.cursor("select * from DEPARTMENT")	

12	=file("D:\\Department5.txt").export@a(A11)	 R&D/2 Sales/3 Technology/8 Administration 1 Finance 4 HR 5 Still the above examlple, append the contents of A12 to the contents of **Department5.txt**
13	=demo.cursor("select * from DEPARTMENT")	
14	=file("D:\\Department6.txt").export@b(A13)	 rqtbx 偰DEPT莇ANAGER 俞Technology? Export data as the binary file with a higher speed.
15	=demo.cursor("select * from EMPLOYEE order by GENDER")	Sort by **GENDER**
16	=file("D:\\EMPLOYEE.txt").export@g(A15,EID,NAME,SURNAME,GENDER,SALARY;GENDER)	Export data to **EMPLOYEE.txt** file based on groups divided by **GENDER**

Related concepts:

f.export(A,x:F,…;s)

F.export()

Description:

Write the cursor fields into the files in *F* respectively, with one file for one field.

Syntax:

F. **export(***cs*,*x*:F_i,…**)**

Remarks:

Write the fields from cursor *cs* to the file object sequence *F*, and return binary files each of which for one field. The fields in the *cs* must be as many as the files in *F*. The file extension indicates the file type.

Parameters:

F	File object sequence
cs	Cursor data to be exported
x	Field(s) to be exported. If omitted, then export all fields that can be textualized from *cs*. In this case, the number of fields and that of files must be the same.If not omitted, the number of *x* must also be the same as that of files.
F_i	Result field name. If omitted, then use the original field name.

Option:

@a	Append. If omitted, then overwrite the original file.

Return value:

Multiple binary files in which each file holds a single field from cursor *cs*

Example:

	A	B	
1	=directory@p("D://txt//")	=[]	Retrieve all files under the directory

2	for A1.len()		
3		=file(A1(A2))	
4		>B1=B1|B3	Save the file objects obtained from the loops in cell B1
5	=demo.cursor("select * from STOCKRECORDS")		Return a cursor of retrieved data
6	=B1.export(A5, STOCKID:StockID,DATE:TradeDate, CLOSING)		Export the fields from cursor to the member file of B1

Related concepts:

db.cursor()

f.export()

f.export(cs,x:F,…;s)

exportxls()

f.exportxls()

Description:

Write a TSeq to an Excel file

Syntax:

f.**exportxls**(*A*,*x*:*F*,…;*s*)

Remark:

Write TSeq *A* to Excel file object *f*. Export all fields if no *x* is specified. Program will create the file *f* (cannot create path directory automatically) automatically; if there are files with the same name, then they will be overwritten.

Parameter:

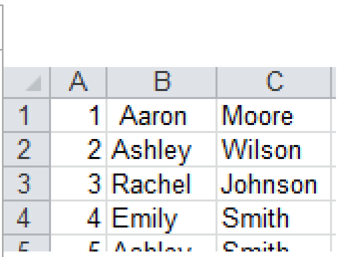
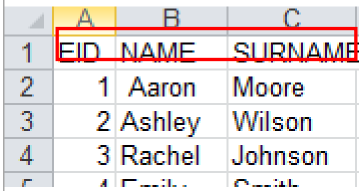
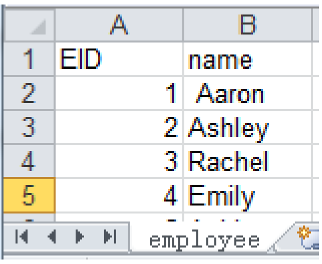
f	File object
A	TSeq to be exported
x	Fields to be exported. If omitted, then export all fields which can be textualized in the RSeq *A*
F	Result field name. If omitted, then use the original field name
s	Sheet name

Options:

@t	Write the first record as the header into a file
@x	In the xlsx format; use the file extension as a default way to determine the file type. If failed to determine it, then just use xls

Example:

	A
1	=demo.query("select EID,NAME,SURNAME

	from EMPLOYEE")	
2	**=file("D:\\EMPLOYEE1.xls").exportxls(A1)**	 A B C 1 1 Aaron Moore 2 2 Ashley Wilson 3 3 Rachel Johnson 4 4 Emily Smith
3	**=file("D:\\EMPLOYEE2.xls").exportxls@t(A1)**	 A B C 1 EID NAME SURNAME 2 1 Aaron Moore 3 2 Ashley Wilson 4 3 Rachel Johnson With **@t** option, write the 1st row into the file as the header
4	**=file("D:\\EMPLOYEE3.xlsx").exportxls@x(A1)**	With **@x** option, use xlsx format
5	**=file("D:\\ EMPLOYEE4.xls").exportxls@t(A1,EID,NAME: name;"employee")**	 A B 1 EID name 2 1 Aaron 3 2 Ashley 4 3 Rachel 5 4 Emily employee Specify and export fields to the sheet of ″**employee**″
6	**=file("D:\\EMPLOYEE4.xl").export@t(A1)**	Cannot determine the file extension, so export data as xls format to″EMPLOYEE4.xl″ sheet

Related concepts:

[f.export()]

f@o(…)

Description:

Syntax of complex functions.

Syntax:

f@*o*(…) With various function options, a function can implement various functions. The basic format of function options is *f@o*(…), and "*o*" is the option of function *f*.

Parameters:

f Function name

@*o* Function option. Different options support different functions, and a same option has almost the same meaning in various functions.

(…) The input parameters of a function. The separator of parameters can be colon **":"**, comma **","**, and semicolon **";"** and their priorities decrease from left to right just as the introduction order above.

{...} The bracket for changing the level to which the parameters belong

fact()

Description:

Compute the factorial of parameters

Syntax:

fact(*nExp*)

Remark:

Compute the factorial of *nExp*

Parameters:

nExp Data for which you want to compute the factorial

Return value:

Long integer (64 bit)

Example:

- fact(10) 3628800

false

Description:

False value – a logical constant.

Syntax:

false

Remark:

It can be used directly in the constant cell or expression.

Example:

	A	B	
1	=null		Assign **null** to **A1**
2	=A1==null		Judge if **A1** is **null**
3	if A2==false	>a=4	If **A2** is false, assign **4** to **a**
4	else	>a=3	Otherwise assign **3** to **a**

Related concepts:

true

null

fetch()

cs.fetch()

Description:

Fetch one or more records from a cursor.

Syntax:

cs.**fetch**(*n*;*x*)

Remark:

From a cursor, fetch *n* records or fetch them continuously till *x* has changed. Then, organize them in a TSeq and return the TSeq. If the records in the cursor has been fetched out, return null after trying to fetch again. This function is used to fetch records in batch where there is a great many of them. Return the remaining records and close the cursor when omitting *n* and *x*. Only one of the two parameters *n* and *x* is valid.

Parameters:

cs	a cursor
n	An integer
x	Grouping expression, and *cs* is sorted by *x*. With *x*, *n* will be ignored.

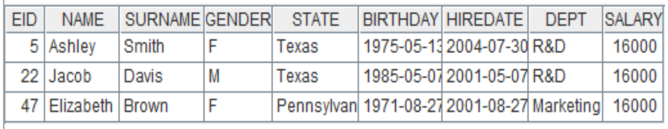
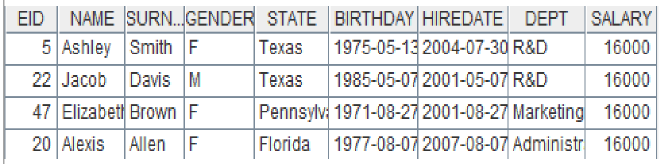
Options:

@0	The selected data will not be fetched out from cursor actually. The action equals copying data from cursor. The option doesn't support *x*.
@x	Close the cursor after data are fetched

Return value:

A table sequence

Example:

	A	
1	**=demo.cursor("select * from EMPLOYEE order by SALARY desc")**	Return a cursor of retrieved data, and sort by **SALARY**
2	**=A1.fetch@0(3)**	 Select top 3 highest-paid employees
3	**=A1.fetch(;SALARY)**	 Select a group of equally-paid employees. Because @0 option is used by A2, A3 retrieves data from a complete cursor.
4	**=A1.fetch()**	Return the remaining cursor
5	**=demo.cursor("select * from EMPLOYEE order by SALARY desc")**	
6	**=A5.fetch@x(3)**	Close the cursor after data are fetched

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000
22	Jacob	Davis	M	Texas	1985-05-07	2001-05-07	R&D	16000
47	Elizabeth	Brown	F	Pennsylvan	1971-08-27	2001-08-27	Marketing	16000

EID	NAME	SURN...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000
22	Jacob	Davis	M	Texas	1985-05-07	2001-05-07	R&D	16000
47	Elizabeth	Brown	F	Pennsylv	1971-08-27	2001-08-27	Marketing	16000
20	Alexis	Allen	F	Florida	1977-08-07	2007-08-07	Administr	16000

Related concepts:

db.cursor()

cs.skip()

ds.fetch()

Description:

From the dimension table space, get the record contents from the specified dimension table

Syntax:

ds.**fetch**(*A*:*K*:…,*V*,*x*:*F*,…)

Remarks:

By matching field expression in the sequence A with the primary key of corresponding segmental dimension table, get the record data from the segmental dimension table *V*. Then, fill the results of computing expression *x* against the segmental dimension table into the field *F*. Lastly, return a RSeq comprising the field F_1 and F_2.

Parameters:

ds	Dimension table space
A	A sequence
K	Expression that is evaluated by computing the sequence *A*
V	Dimension table name. Normally, it is the global variable name, and the variable value is just this dimension table
x	Function expression that gets evaluated by computing the dimension table
F	Field name

Return value:

RSeq

Example:

	A	
1	=hosts(["192.168.0.232:8281","192.168.0.147:8281"],2)	
2	=file("D:/2010 year Stock.txt").import@t()	
3	=callx@a("Product.dfx",[1,299],[298,399];A1)	
4	=dims(A1;@DimProduct,if(?<36,1,2))	
5	=A4.fetch(A2:StockID,@DimProduct,StockID:StockID1, StockNum:StockTotalNum)	From the dimension table **@DimProduct**, retrieve a TSeq composed ofstock record 2010'sStockID field and StockNum field.
6	=A4.fetch(to(100),@DimProduct,StockID,StockNum)	From the dimension table **@DimProduct**, get the records whose stock number is within 100.

field()

r.field()

Description:

Get the value of a field in a record.

Syntax:

r.**field**(*i*)

Remark:

Get value of the *i*th field in record *r*.

Parameters:

r	A record
i	Sequence number of the field

Return value:

A field value or a sequence composed of multiple field values.

Example:

- **demo.query("select * from EMPLOYEE")(1).field(2)**

  Return value of the second field of the first record in TSeq **demo.query ("select * from EMPLOYEE").**

Related concepts:

[A.field(i)](#)

[A.field(i,x)](#)

[r.field(i,x)](#)

r.field(*i*,*x*)

Description:

Modify values in the specified field of record r

Syntax:

r. **field**(*i*,*x*)

Remarks:

Modify the value of the *i*th field of the record *r* into *x*.

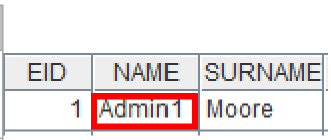
Parameters:

r	Record
i	Sequence number of a field
x	Expression whose computational result is the value of the *i*th field

Return value:

Field value

Example:

	A	
1	**=demo.query("select * from EMPLOYEE")**	 EID NAME SURNAME 1 Admin1 Moore
2	**=A1(1).field(2,"Admin1")**	Change the value of the 2nd field of the record 1 to **Admin1**

Related concepts:

[A.field(i)](#)

A.field(i,x)

r.field(i)

A.field(*i*)

Description:

Get the values in the *i*th field of a sequence.

Syntax:

A.**field**(*i*)

Remark:

Generate a new sequence composed of the values in the *i*th field of each record in *A*.

Parameters:

A	A sequence
i	The sequence number of a field

Return value:

The new sequence composed of the values of the *i*th field of each record in *A*.

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.field(2)	Return a sequence composed of values of the second field.

Related concepts:

r.field(i)

r.field(i,x)

A.field(i,x)

A.field(*i*,*x*)

Description:

Get the modified value of the *i*th field in a sequence

Syntax:

A. **field**(*i*,*x*)

Remarks:

Get the value of the *i*th field of each record in *A*, reassign *x* to it, and return a sequence consisting of the new values

Parameters:

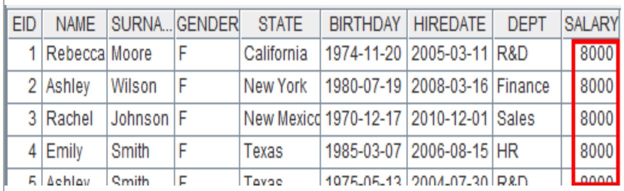
A	A sequence
i	Sequence number of a field
x	Expression whose computational result is the value of the *i*th field

Return value:

A new sequence composed of the result values of the *i*th field in *A* by computing expression *x*.

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.field(9,8000)	Modify the value of the ninth field



EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	8000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	8000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	8000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	8000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	8000

Related concepts:

r.field(i)

A.field(i)

r.field(i,x)

file()

file(*fn*:*cs*)

Description:

Open a file object with the specified file name.

Syntax:

file(*fn*{:cs})

Remark:

Open a file object with the file name of *fn*.

Parameters:

fn Name of the file to load. Both the absolute path and the relative path are acceptable. If using the relative path, then it is relative to the main path. The main path is the one configured on the option menu of esProc, and by default it is the current directory (the directory opened on running the dfx file).

cs Character set. It supports jvm's built-in character set. It is the default value of virtual machine by default.

Option:

@s Search the file names on the non-absolute path in a specified order: Class path -> Searching path-> Main path. The Searching path is the one configured on the option menu of esProc. The default main path is the current directory.
The return result is the name of a read-only file.

Return value:

A file object

Example:

- **file("D:\\Area.txt":"UTF-8")** Load the file Area.txt from the directory "**D:\**" . Character set is **UTF-8**
- **file("\\Area.txt")** Load the file Area.txt under the Main path.

- **file@s("data\\ Area.txt")** Search the class path first. If not found, then search throughout the list of Searching paths. Lastly, search the Main path if not found in the Searching paths.

file(*fn*:*cs*,*z*,*h*)

Description:

Open the file object of the remote file with the specified name.

Syntax:

file(*fn*:*cs*,*z*,*h*)

Remark:

Select a sub-machine from the sequence *h*, then open the file *fn* from the partition *z* of the sub-machine. Firstly, search for the file whose name is *fn* locally. If not found, then search for it among files on the relatively idle sub-machine of the remote machine. The remote file cannot be written.

Parameter:

fn Name of the file to load. Both the absolute path and the relative path are acceptable. If using the relative path, then it is relative to the Main path. In the IDE, the Main path is the one configured on the option menu of esProc and by default it is the current directory. On server, the Main path is the value of main Path in the configuration file config.xml, and this value must not be null.

cs Character set. It supports jvm's built-in character set. It is the default value of virtual machine by default.

z Partition name

h Sequence of sub-machines

Return value:

File object

Example:

- **=file("D:\\Scores.txt":"UTF-8","2","192.168.0.20:9282")** Load the file “Scores.txt” from the D:\ of the native machine and the character set is **UTF-8**
- **=file("Scores.txt":"UTF-8","2","192.168.0.20:9282")** Firstly, load the file Scores.txt under the Main path of the native machine.
- **=file("Scores1.txt":"UTF-8","2","192.168.0.20:9282")** If **Scores1.txt** is not found on this machine, then load the file "Scores1.txt" on partition "2" of the sub-machine"**192.168.0.20:9282**".

filename()

Description:

Split up the full path, and get the file name and suffix

Syntax:

filename(*fn*)

Remark:

Split the file name and suffix contents from the full path *fn*

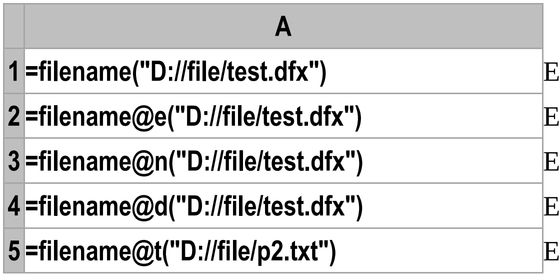
Parameter:

fn The full path with the file name

Options:

@e	Only split the extension part from the full path
@n	Split out the file name without the extension
@d	Split out the directory under which the file is located, inclusive of address string
@t	Under *fn* directory, generate and return the full path names of temporary files of the same type. If *fn* is omitted, they will be generated under the system temporary directory, and the file name relative to the main path will be returned.

Example:



	A	
1	=filename("D://file/test.dfx")	Export "**test.dfx**"
2	=filename@e("D://file/test.dfx")	Export "**dfx**"
3	=filename@n("D://file/test.dfx")	Export "**test**"
4	=filename@d("D://file/test.dfx")	Export " **D://file**"
5	=filename@t("D://file/p2.txt")	Export "**D:\file\tmpdata3665839327103632105.txt**"

Related concept:

file()

fill()

Description:

To create a string by concatenating strings together.

Syntax:

fill(*s*, *n*)

Remark:

Get a new string by concatenating *n* strings (*s*) together

Parameters:

s Source strings for making up a new string

n Number of source strings in the new string

Return value:

Character

Example:

fill("1 ",10)	"1 1 1 1 1 1 1 1 1 1 "
fill("a b",10)	"a ba ba ba ba ba ba ba ba ba b"

find()

A.find()

Description:

To find a record whose primary key value is *v*.

Syntax:

A.**find**(*v*)

Remark:

If the sequence *A* is composed of non-record members, *A*.**find()** is used to get the first member equal to *v* from *A*.

If *A* is a record sequence, *A*.**find()** is used to get the member from *A* whose primary key values are equal to *v*. Use the index table if there is one.

Parameters:

A	A sequence
v	A sequence member or a primary key value; a sequence shall be used if there are multiple primary key values.

Option:

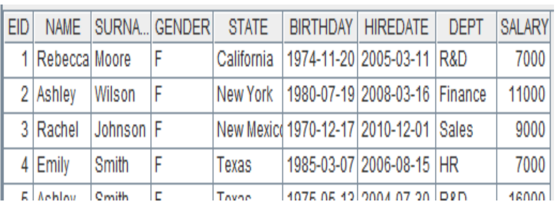
@b	Enable the binary search. Here, *A* must be ordered by the primary key; otherwise, the result will be wrong. The index table will be ignored.

Return value:

The member found

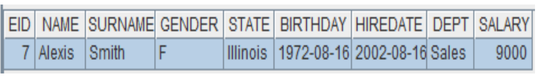
Example:

	A
1	=demo.query("select * from EMPLOYEE")



EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexic	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000

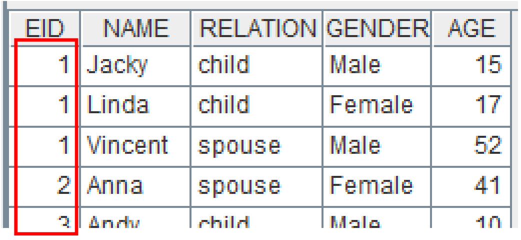
	A
2	>A1.primary(NAME,DEPT)
3	=A1.find(["Alexis","Sales"])



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
7	Alexis	Smith	F	Illinois	1972-08-16	2002-08-16	Sales	9000

A sequence is used because there are multiple fields for forming the primary key

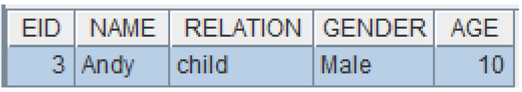
	A
4	=demo.query("select * from FAMILY")



EID	NAME	RELATION	GENDER	AGE
1	Jacky	child	Male	15
1	Linda	child	Female	17
1	Vincent	spouse	Male	52
2	Anna	spouse	Female	41
3	Andy	child	Male	10

This table sequence is ordered by the **EID** field

	A
5	>A4.primary(EID)
6	=A4.find@b(3)



EID	NAME	RELATION	GENDER	AGE
3	Andy	child	Male	10

@b is used to enable the binary search in order to speed up the computation

	A
7	=[1,3,5,7]
8	=A7.find(3)

3; because **A7** is neither a record sequence nor a table sequence, the search operation is conducted according to members' values

Related concepts:

A.pfind()

mdb.find()

Description:

Query the data of a MongoDB and return the result in the form of a cursor

Syntax:

mdb.**find**(*c*,*f*,*s*)

Remark:

Query field *s* in table *c* of a MongoDB, select records satisfying *f* and return the result as a cursor

Parameters:

mdb	Connection to MongoDB
c	The table name
f	Filtering condition, whose syntax is the same as that of MongoDB
s	A field, whose syntax is the same as that of MongoDB. All fields will be returned by default.

Return value:

A cursor

Example:

	A	
1	=mongodb("mongo://127.0.0.1:27017/myTest?user=root &password=sa")	
2	=A1.find("Score","{name:'Sean'}","{id:1,score:1,_id:0}")	Find records of id and score from table Score according to the condition that the name is Sean
3	=A1.close()	Close the MongoDB connection

Related concepts:

mongodb()

mdb.close()

mdb.count()

mdb.distinct()

mdb.aggregate()

float()

Description:

Convert a string or a number into a 64-bit double-precision floating-point number.

Syntax:

float(*stringExp***)**

float(*numberExp***)**

Remark:

The computation of *stringExp* must be a string that consists of a number which is less than or equal to 64 bits. For a value of more than 64 bits, the result of **float(***stringExp***)** will be imprecise.

The computation of *numberExp* must be a numeric value which is less than or equal to 64 bits. For a value of more than 64 bits, the result of **float(***numberExp***)** will be imprecise.

Parameters:

stringExp	The string expression you want to return as a double-precision floating-point number.
numberExp	The number you want to return as a double-precision floating-point number.

Return value:

64-bit double-precision floating-point number

Example:

–	**float("1234567")**	**1234567.0**
–	**float(1234567)**	**1234567.0**

Related concepts:

decimal()

int()

long()

number()

string()

floor()

Description:

Truncate data at the specified position, and reject all the remaining part if any

Syntax:

floor(*numberExp*, {*nExp*}**)**

Remark:

Truncate the data *numberExp* at the specified position *nExp*, and reject all the remaining part if any

Parameters:

numberExp	Data to be truncated
nExp	Integer to specify the position from which data are to be truncated, **>0**: Move the decimal point to the right for *nExp* places **<0**: Move the decimal point to the left for *nExp* places **=0**: Indicate the current decimal place.

Return value:

Numeric

Example:

– **floor(3451231.234,0)**	**3451231.0**
– **floor(3451231.234,-1)**	**3451230.0**
– **floor(3451231.234,-2)**	**3451200.0**
– **floor(3451231.234,1)**	**3451231.2**
– **floor(3451231.234,2)**	**3451231.23**

Related concepts:

ceil()

round()

fname()

T.fname(*i*)

Description:

Get the names of fields of the TSeq according to the serial numbers of these fields

Syntax:

T. **fname(***i***)**

Remarks:

From TSeq *T*, get the name of the field whose serial number is *i*. If no parameter, then return all field names in *T*

Parameters:

T	TSeq

i Serial number of a field

Return value:

A string or a sequence

Example:

	A	
1	=demo. query("select * from EMPLOYEE")	
2	=A1.fname(2)	"NAME", the name of the 2nd field is NAME
3	=A1.fname()	Return a sequence comprising all field names in A1

Related concepts:

r.fname()

r.fname(*i*)

Description:

Get the field names from records according to the serial numbers of these fields

Syntax:

r. fname(*i*)

Remarks:

From the record *r*, get the name of the field whose serial number is *i*. If no parameter, then return all field names in *r*

Parameters:

r Record

i Serial number of a field

Return value:

Character strings or sequences

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1(1).fname(2)	"NAME", the name of the 2nd field in the first record is NAME
3	=A1(1).fname()	Return a sequence comprising all field names of the 1st record in A1

Related concepts:

T.fname()

fno()

T.fno(*F*)

Description:

Get the serial number of a field in a table sequence.

Syntax:

T.fno(*F*)

Remark:

Get the serial number of field *F* in *T*, if there are no parameters, the number of fields in *T* is returned

Parameters:

T	TSeq
F	Field name

Return value:

The serial number of field *F* in *T*

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.fno(EID)	**1**, which indicates that the serial number of field **"EID"** is **1**
3	=A1.fno()	**9**, which indicates that there are 9 fields in the table sequence

Related concepts:

[r.fno()]

r.fno(*F*)

Description:

Get the serial number of a field of a record in a table sequence .

Syntax:

r.fno(*F*)

Remark:

Get the serial number of field *F* of record *r* in the table sequence; if there is no parameter, return the number of fields in the table sequence in which *r* locates.

Parameters:

r	A record
F	Field name

Return value:

The serial number of field *F*

Example:

	A
1	=demo.query("select * from EMPLOYEE")

2	=A1(2).fno(EID)	**1**, which indicates that the serial number of the field "**EID**" is **1**
3	=A1(2).fno()	**9**, which indicates that there are 9 fields in the table "**Employee**"

Related concepts:

T.fno()

for

for *a,b,s*

Description:

Execute a loop according to the specified scope and step

Syntax:

for *a,b,s*

Remark:

Execute the code block in loops from *a* to *b* with a step of *s*

Parameters:

a	An integer
b	An integer
s	An integer to indicate the step. Its value is 1 by default.

Example:

	A	B	
1			Save the cumulative total -18 - in A1
2	for 1,11,5		
3		>A1=A1+A2	Add up integers within the scope from 1 to 11 with a step of 5, i.e. every 4 integers

for *cs,n;x*

Description:

Loop through the cursor

Syntax:

for *cs,n;x*

Remark:

Retrieve *n* records from the cursor and return or retrieve records till *x*… has changed. Close the cursor once the retrieval is completed. This function is often used to fetch a large volume of records by group. Return all the remaining records and close the cursor when omitting *n* and *x*.

Parameters:

cs Cursor

n Number of records

x Grouping expression, and *cs* is sorted by *x*. With *x*, *n* can be ignored.

Example:

	A	B	
1	=demo.cursor("select * from EMPLOYEE order by SALARY desc")	=[]	
2	for A1,10;SALARY		
3		>B1.insert(0,A2)	Loop through the cursor, selecting records with the same **SALARY** value and inserting them into B1.

for *x*

Description:

To start a loop.

Syntax:

for *x*

Remark:

Start a loop executing the code block

Parameter:

x A sequence, an integer, or a logic expression. Loop through the code block against *x* or **to(x)**, or if *x* is true. If *x* is empty, then the loop will be endless.

Example:

When *x* is an integer

	A	B	C	
1				Save the cumulative total **55** in **A1**
2	for 10			
3		>A1=A1+A2		Add up the integers from **1** to **10**

When *x* is empty

	A	B	C	
1	=0			**5050**
2	for			Loop will be endless if *x* is empty. When the loop count reaches **100**, then break the loop
3		>A1=A1+#A		
4		if #A2==100	break	

When *x* is a Boolean expression

	A	B
1	=15	

2	for A1>10	
3		>B1=B1+#A2
		>A1=A1-1

If **A1 >10**, then start a loop through the code block; otherwise, quit the loop

fork

Description:

Multithread is used to execute the code block in this cellset.

Syntax:

fork *Ai,…*

Remark:

Ai is the sequence parameter, whose length determines the number of parallel threads and in which the single-value parameters will be duplicated to compose a sequence. Make multiple duplicates of the cellset and the context being executed currently , and then use multithreads to execute them respectively. The parameters will be separated and used for execution respectively and the result of each thread becomes the value of its corresponding cell. The result of *result* statement in code block will be combined into a sequence which will act as the cell value of the main thread to call the subroutine.

Parameters:

Ai A sequence

Example:

1) Single Parameter

	A	B
1	fork [[1,20,6,14,5],[32,8]]	// return the result of *result* statement of multithread
2		=connect("demo")
3		=B2.query("select * from EMPLOYEE where EID in (?)",A1)
4		=B2.close()
5		result B3

2) Multiple Parameters

	A	B
1	fork [[1,20,6,14,5],[32,8]],"F"	// return the result of *result* statement of multithread
2		=connect("demo")
3		=B2.query("select * from EMPLOYEE where EID in (?) and GENDER=?",A1(1),A1(2))
4		=B2.close()
5		result B3

Related concepts:

callx()

call()

func

func()

Description:

To call a subroutine.

Syntax:

func (*a*,*arg*)

Remark:

A subroutine can be called in any cell as required once defined in a program cellset. The **func()** function is used to call the subroutine starting with master cell *a* and, in the meantime, the parameter *arg* will be passed in as the cell value of *a*. After the call is completed, the return value defined in the subroutine will be returned.

Parameters:

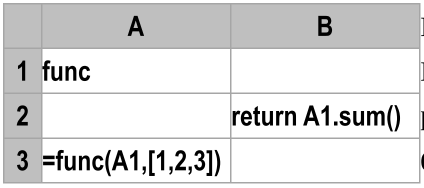
a The master cell of a sub routine; usually, it is the cell where the **func** is located

arg The parameter used in a subroutine. It can be either a single value or a sequence.

Return value:

The return value of the subroutine.

Example:



	A	B
1	func	
2		return A1.sum()
3	=func(A1,[1,2,3])	

Name the program cellset as **p1**

Define a subroutine and return the sum of the members of the passed parameter – whose the value is a sequence

6. **A3** is a method of calling and the return value is **6**

func …{return x_i}

Description:

Define the function block of a function.

Syntax:

```
func
    …
    {return xi}
```

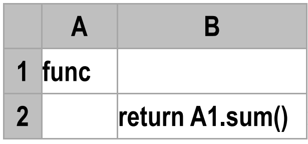
Remark:

Define a function block. The result of computation may or may not be returned. A function block (subroutine) begins with the start symbol **func**. The content of a subroutine is a code block taking the cell containing **func** as the master cell. The **return** statement is used to return the function result and terminate the function's code block.

Return value:

The result of the function block.

Example:



	A	B	
1	func		Define a subroutine; the parameter passed in is a sequence; the sum of
2		return A1.sum()	members of the sequence is returned

Related concepts:

func()

gf()

ds.gf()

Description:

In the dimension table space *ds*, append dimension table *V* and the segmental function *g*

Syntax:

ds. **gf**(*V*,*g*;…)

Remarks:

Use *dims()* to generate dimension table space *ds*. If the dimension table space ds does not have the dimension table *V*, use *ds*.**gf** () to append dimension table *V* and the segmental function *g* to the dimension table space *ds*.

Parameters:

ds Dimension table space

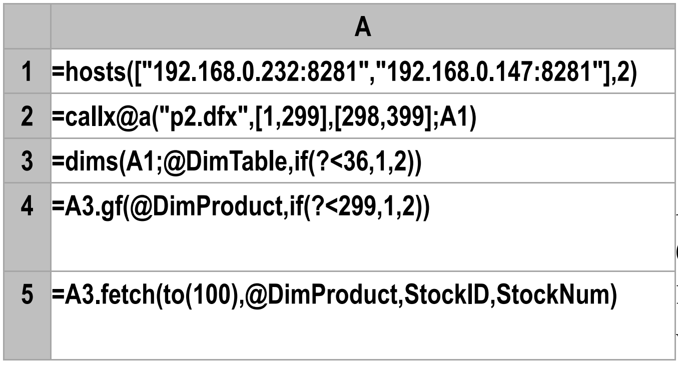
V Dimension table name. Typically, it is a global variable name, and the variable value is just this dimension table

g Segmental function expression, which takes the primary key of the dimension table as the parameter, and returns a value of to(*n*)

Return value:

Dimension table space

Example:



	A	
1	=hosts(["192.168.0.232:8281","192.168.0.147:8281"],2)	
2	=callx@a("p2.dfx",[1,299],[298,399];A1)	
3	=dims(A1;@DimTable,if(?<36,1,2))	
4	=A3.gf(@DimProduct,if(?<299,1,2))	Append dimension table @DimProduct
5	=A3.fetch(to(100),@DimProduct,StockID,StockNum)	Fetch records whose stock ID is within 100 in the dimension table

group()

A.group(*x*$_i$,…)

Description:

Perform equal grouping according to *x*$_i$,…

Syntax:

A.**group**(*x*$_i$,…)

Remarks:

Perform equal grouping on a sequence or an RSeq according to a single or multiple fields/expressions, the result is a sequence consisting of groups

Options:

@o Group records by comparing adjacent ones, which is equal to the merging operation, and the result set won't be sorted.

@1 Get the first record of each group to form an RSeq and return it (Please note that 1 is a number instead of a letter)

@n *x* gets assigned with group numbers which can be used to define the groups. @n and @0 are mutually exclusive.

Parameters:

A A sequence

x$_i$ Grouping expression. When grouping data by multiple fields or expressions, separate grouping expressions from each other by commas

Return value:

A sequence or an RSeq

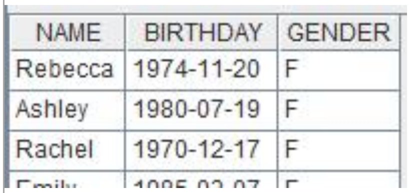
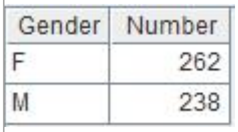
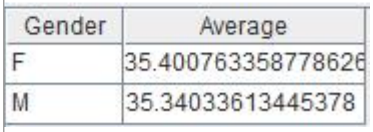
Example:

- Group a sequence

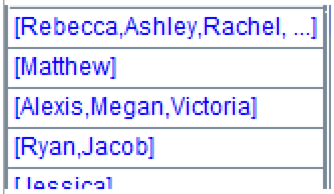
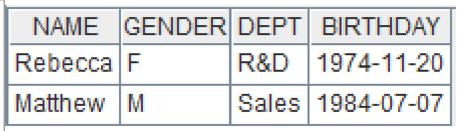
	A	
1	[6,9,12,15,16,5,1,7,8]	
2	=A1.group(~%2)	[[6,12,16,8],[9,15,5,1,7]] The series is divided into two groups. Divide the number of members in both groups by 2, one of the remainders is 0 and the other is 1
3	=A1.group(~%2,~%3)	[[6,12],[16],[8],[9,15],[1,7],[5]] Group the series according to multiple expressions.

- Reuse the grouping result

	A	
1	=demo.query("select NAME,BIRTHDAY,GENDER from EMPLOYEE")	
2	=A1.group(GENDER)	[[Rebecca,Ashley,Rachel,…],[Matthew,Ryan,Jacob,

<table>
<tr><td></td><td></td><td>...]], each group is a sequence

NAME BIRTHDAY GENDER
Rebecca 1974-11-20 F
Ashley 1980-07-19 F
Rachel 1970-12-17 F
Emily 1985-03-07 F</td></tr>
<tr><td>3</td><td>=A2.new(GENDER:Gender,~.count():Number)</td><td>
Gender Number
F 262
M 238
Count members of each group</td></tr>
<tr><td>4</td><td>=A2.new(GENDER:Gender,~.avg(age(BIRTHDAY)):Average)</td><td>
Gender Average
F 35.40076335877862
M 35.34033613445378
Get different statistical results using the same grouping result</td></tr>
</table>

- ➢ Group data by multiple fields

<table>
<tr><th></th><th>A</th><th></th></tr>
<tr><td>1</td><td>=demo.query("select NAME,GENDER,DEPT,BIRTHDAY from EMPLOYEE")</td><td></td></tr>
<tr><td>2</td><td>=A1.group(GENDER,DEPT)</td><td>Group records by multiple fields</td></tr>
<tr><td>3</td><td>=A1.group@o(GENDER)</td><td>Records won’t be sorted; only the adjacent ones will be compared and group them together if they are the same. Same records that are not adjacent may be put into different groups, so there may be overlapped groups and the return result is a set of sequences.

[Rebecca,Ashley,Rachel, ...]
[Matthew]
[Alexis,Megan,Victoria]
[Ryan,Jacob]
[Jessica]</td></tr>
<tr><td>4</td><td>=A1.group@1(GENDER)</td><td>
NAME GENDER DEPT BIRTHDAY
Rebecca F R&D 1974-11-20
Matthew M Sales 1984-07-07
Return the first record of each group</td></tr>
<tr><td>5</td><td>=A1.group@n(if(GENDER=="F",1,2))</td><td><i>x</i> gets assigned with group numbers which can be used to define the groups directly.</td></tr>
</table>

[[Rebecca,Ashley,Rachel,...],[Matthew,Ryan,Jacob,...]]

Related concepts:

A.id()

A.group(x:F;y:G…)

A.group(*x*:*F*;*y*:*G*…)

Description:

Group a sequence and then perform aggregate operations

Syntax:

A.**group**(*x*:*F*;*y*:*G*…)

Remarks:

First group the sequence according to a single or multiple fields/expressions and then aggregate each group of data. After the sequence is grouped according to *x*, a new TSeq with fields being *F*,... *G*,… will be created. Values of *F* field are the value of *x* field of the first record in each group and *G* field gets values by computing *y* with regard to each group.

Options:

@o Group records by comparing adjacent ones, which is equal to the merging operation, and the result set won't be sorted.

@n *x* gets assigned with group numbers which can be used to define the groups. @n and @o are mutually exclusive.

Parameters:

A A sequence

x Grouping expression. If omitting *x*:*F*, aggregate the whole set without grouping; in this case ;must not be omitted

F Field name of the result TSeq

G Names of summary fields in the result TSeq

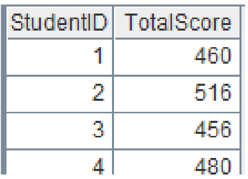
y Aggregate expression in which ~ is used to reference a group

Return value:

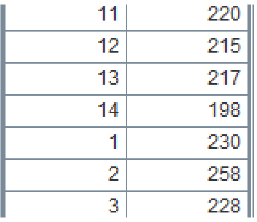
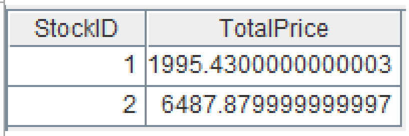
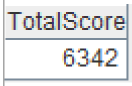
A sequence

Example:

	A
1	=demo.query("select * from SCORES")
2	=A1.group(STUDENTID:StudentID;~.sum(SCORE):TotalScore)



StudentID	TotalScore
1	460
2	516
3	456
4	480

3	=A1.group@o(STUDENTID:StudentID;~.sum(SCORE):TotalScore)	 11 220 12 215 13 217 14 198 1 230 2 258 3 228 Only the adjacent ones will be compared and group them together if they are the same. The result set won't be sorted.
4	=demo.query("select * from STOCKRECORDS where STOCKID<'002242'")	
5	=A4.group@n(if(STOCKID=="000062",1,2):StockID;~.sum(CLOSING):TotalPrice)	 StockID TotalPrice 1 1995.4300000000003 2 6487.879999999997 *x* gets assigned with group numbers
6	=A1.group(;~.sum(SCORE):TotalScore)	 TotalScore 6342 x:F is omitted, so compute the total score of all the students

Related concepts:

A.id()

A.group(x_i,…)

groupn()

cs.groupn()

Description:

Group cursor records by the group sequence number and return a cursor sequence

Syntax:

cs.**groupn**(*x*)

Remark:

Group the records in cursor *cs* by *x*, and *x* is the group sequence number for direct locating. Return the result as a cursor sequence.

Parameters:

cs	Cursor records
x	Group sequence number

Option:

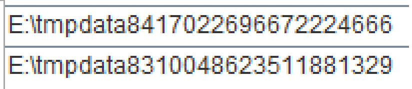
@f	Return a sequence of file names

Return value:

Cursor sequence

Example:

	A

1	=demo.cursor("select * from SCORES")	
2	=A1. groupn(if(CLASS=="Class one",1,2))	Return as a sequence cursor
3	=demo.cursor("select * from SCORES")	
4	=A3.groupn@f(if(CLASS=="Class one",1,2))	 E:\tmpdata8417022696672224666 E:\tmpdata8310048623511881329 Files corresponding to each group of data

Related concepts:

cs.groups()

cs.groupx()

groups()

A.groups()

Description:

Group a TSeq and then get the aggregating result cumulatively.

Syntax:

A.**groups**(*x*:*F*…;*y*:*G*…)

Remark:

Group and aggregate the TSeq simultaneously by one or multiple fields/expressions to generate a new TSeq with *F*,.. *G*… as the fields. Namely, during the traversal through members of *A*, they will be placed to the corresponding result set one by one; in the meanwhile, the aggregating result of each result set will be computed cumulatively. Compared with the method of grouping first then aggregating, the function has a better performance.

Options:

@o Group records by comparing adjacent ones, which is equal to the merging operation, and the result set won't be sorted.

@n *x* gets assigned with group numbers which can be used to define the groups. @n and @o are mutually exclusive.

Parameters:

A A sequence

x Grouping expression

F Field name of the result TSeq

y *y* is the aggregate function with which *A* is traversed, and which only supports sum/count/max/min/topx/avg

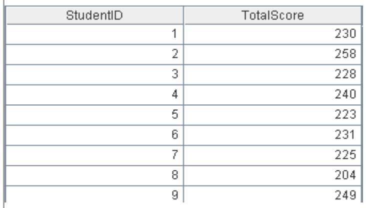
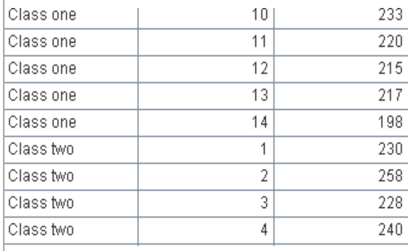
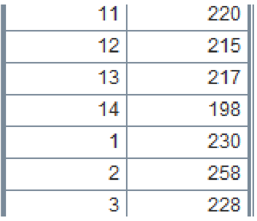
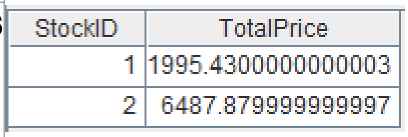
G Summary field name in the result TSeq

Return value:

Post-grouping TSeq

Example:

	A
1	=demo.query("select * from SCORES where

	CLASS = 'Class one'")	
2	=A1.groups(STUDENTID:StudentID;sum(SCORE):TotalScore)	 StudentID TotalScore 1 230 2 258 3 228 4 240 5 223 6 231 7 225 8 204 9 249 group by a single field
3	=demo.query("select * from SCORES")	
4	=A3.groups(CLASS:Class,STUDENTID:StudentID;sum(SCORE):TotalScore)	 Class one 10 233 Class one 11 220 Class one 12 215 Class one 13 217 Class one 14 198 Class two 1 230 Class two 2 258 Class two 3 228 Class two 4 240 group by multiple fields
5	=demo.query("select * from SCORES")	
6	=A5.groups@o(STUDENTID:StudentID;sum(SCORE):TotalScore)	 11 220 12 215 13 217 14 198 1 230 2 258 3 228 Only compare and merge with the neighboring element, and the result set is not sorted.
7	=demo.query("select * from STOCKRECORDS where STOCKID<'002242'")	
8	=A7.groups@n(if(STOCKID=="000062",1,2):StockID;sum(CLOSING):TotalPrice)	 StockID TotalPrice 1 1995.4300000000003 2 6487.879999999997 The value of *x* is the group sequence number

Note:

Compared with *A*.**group**(*x*:*F*;*y*:*G*…), *A*.**groups**(*x*:*F*…;*y*:*G*…) computes in a accumulative way and thus gets a better performance.

Related concepts:

A.group(xi,…)

A.group(x:F;y:G…)

cs.groups()

Description:

Group the records in the cursor and sort the grouping result by the grouping field. Then get the final aggregating result set cumulatively.

Syntax:

cs.**groups**(*x*:*F*,…;*y*:*G*…)

Remark:

After the cursor records are grouped by *x*, a new TSeq with the fields of *F*,...*G*,… is generated. Then sort the new TSeq by grouping field *x*. The values of *F* field are the value of *x* field of the first record in each group and *G* field gets values by computing *y* with regard to each group. The *y* is the aggregate function with which records of *cs* is aggregated.

Options:

@o Group records by comparing adjacent ones, which is equal to the merging operation, and the result set won't be sorted.

@n *x* gets assigned with group numbers which can be used to define the groups. @n and @o are mutually exclusive.

Parameters:

cs Cursor records

x Group expression. If omitting *x* :*F*, then aggregate the whole set; in this case, ;must not be omitted

F Field name in the result TSeq

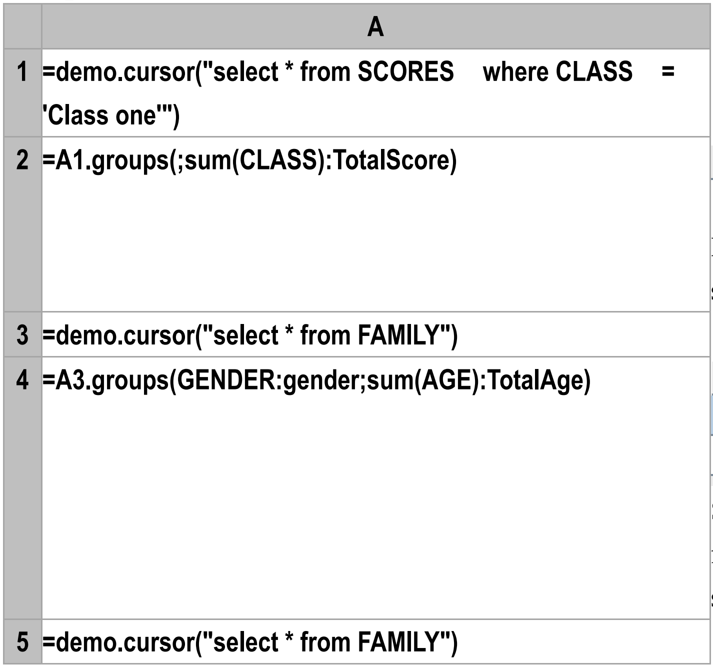
y Aggregate function that only supports sum/count/max/min/topx/avg

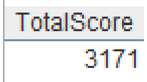
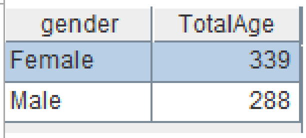
G Summary field name in the result TSeq

Return value:

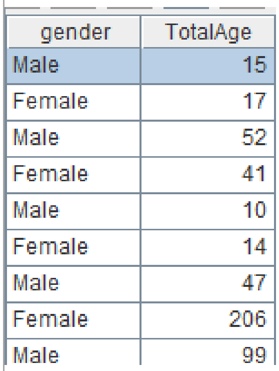
Post-grouping TSeq

Example:



	A	
1	=demo.cursor("select * from SCORES where CLASS = 'Class one'")	
2	=A1.groups(;sum(CLASS):TotalScore)	 TotalScore 3171 If omitting x :F, then the total scores of all students will be computed
3	=demo.cursor("select * from FAMILY")	
4	=A3.groups(GENDER:gender;sum(AGE):TotalAge)	 gender TotalAge Female 339 Male 288 Specify the field group, and sort by the given fieldGroup records by specified field and then sort the result by the grouping field
5	=demo.cursor("select * from FAMILY")	

6 =A5.groups@o(GENDER:gender;sum(AGE):TotalAge)

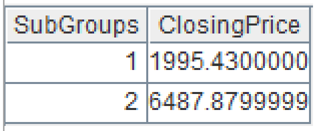


gender	TotalAge
Male	15
Female	17
Male	52
Female	41
Male	10
Female	14
Male	47
Female	206
Male	99

Merge the neighbors by **Gender** field without sorting

7 =demo.cursor("select * from STOCKRECORDS where STOCKID <'002242'")

8 =A7.groups@n(if(STOCKID=="000062",1,2):SubGroups;sum(CLOSING):ClosingPrice)



SubGroups	ClosingPrice
1	1995.4300000
2	6487.8799999

The value of *x* is the group sequence number. For those records whose **STOCKID** is equal to”**000062**”,allocate them to the first group. Otherwise, allocate them to the second group. Meanwhile, aggregate each group of records.

Related concepts:

A.group(xi,…)

A.group(x:F;y:G…)

A.groups()

cs.groupn()

cs.groupx()

groupx()

cs.groupx()

Description:

Group the ordered records in the cursor and return result as a cursor

Syntax:

cs.**groupx**(*x*:*F*,…;*y*:*G*…;*n*)

Remark:

After the cursor records are grouped by *x,* a new TSeq is generated with the fields *F*,...*G*,… Then sort the new TSeq by the grouping field. The values *F* field are the value of *x* field of the first record in each groupand *G* field gets values by computing *y* with regard to each group. The *y* is the aggregate function with which records of *cs*is aggregated. The *n* is the number of records in the grouping result stored in the memory. Once *n* is reached, the records will be cached to the file in the temporary directory, as indicated in the option.

Options:

@o Group records by comparing adjacent ones, which is equal to the merging operation, and the result set won't be sorted.

@n *x* gets assigned with group numbers which can be used to define the groups. @n and @o are mutually exclusive.

Parameters:

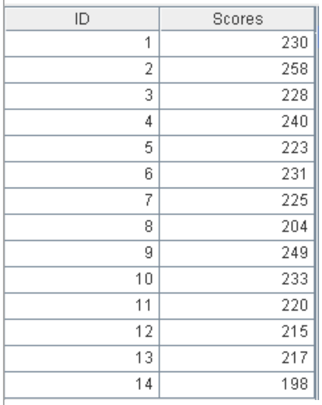
cs	Cursor records
x	Grouping expression
F	Field name of the result TSeq
y	Aggregate function which only supports sum/count/max/min/topx/avg
G	Summary field name in the result TSeq
n	Number of records in buffer; use the system value by default

Return value:

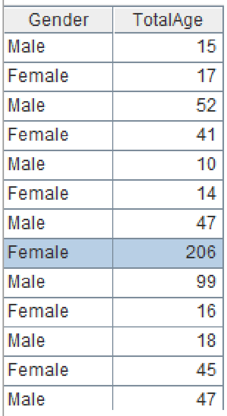
Cursor

Example:

	A
1	=demo.cursor("select * from SCORES")
2	=A1.groupx(STUDENTID:ID;sum(SCORE):Scores;300000).fetch()
3	=demo.cursor("select * from FAMILY")
4	=A3.groupx@o(GENDER:Gender;sum(AGE):TotalAge;300000).fetch()

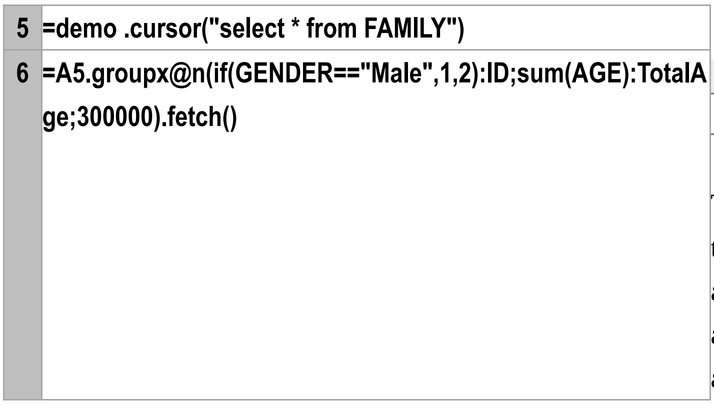


ID	Scores
1	230
2	258
3	228
4	240
5	223
6	231
7	225
8	204
9	249
10	233
11	220
12	215
13	217
14	198

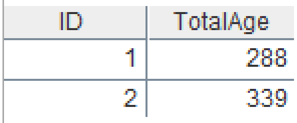


Gender	TotalAge
Male	15
Female	17
Male	52
Female	41
Male	10
Female	14
Male	47
Female	206
Male	99
Female	16
Male	18
Female	45
Male	47

Compare and merge the neighbors only, and the result will not be sorted



```
5 =demo .cursor("select * from FAMILY")
6 =A5.groupx@n(if(GENDER=="Male",1,2):ID;sum(AGE):TotalAge;300000).fetch()
```



ID	TotalAge
1	288
2	339

The value of x is the group sequence number. For the records whose **GENDER** is equal to "**Male**", allocate them to the first group. Otherwise, allocate them to the second group. Meanwhile, aggregate every group of records

Related concepts:

A.group(xi,…)

A.group(x:F;y:G…)

A.groups()

cs.groupn()

cs.groups()

hdfsfile()

Description:

Return HDFS file flow

Syntax:

hdfsfile(*url*,*cs*)

Remark:

Return HDFS file flow. The compression mode can be determined by extension

Parameter:

url	Load Hadoop file in URL format
cs	Character set. It supports JVM's built-in character set. It is the default value of virtual machine by default.

Return value:

File flow

Example:

- **=hdfsfile("hdfs://192.168.0.204:9000/user/root/student.txt","iso-8859-1")** Load Hadoop file "**student.txt**". The character set is **iso-8859-1**.

hosts()

Description:

From the node machine sequence, find the idle machines of a specified number.

Syntax:

hosts(*h*,*n*)

Remarks:

From the node machine sequence *h*, find *n* idle machines.

Parameters:

h	Sequence of node machines
n	Number of idle machines

Return value:

Sequence of character strings

Example:

–	=hosts(["192.168.0.232:8281","192.168.0.86:4001"],1)	["192.168.0.86:4001"]

hour()

Description:

Get the hour from a specified datetime

Syntax:

hour(*datetimeExp*)

Remark:

Get the hour from the specified time *datetimeExp*

Parameter:

datetimeExp	Date or standard datetime format string

Return value:

Integer

Example:

–	hour("1983-12-15")	0
–	hour("1983-12-15 10:30:25")	10
–	hour(datetime("2006-01-15 13:20:30"))	13

Related concepts:

year()

month()

day()

minute()

second()

millisecond()

httpfile()

Description:

Package the returned result of URL as the file flow and return it

Syntax:

httpfile(*url*,*cs*)

Remark:

Package the returned result of URL string of HTTP service into a file flow and return it

Parameters:

url HTTP service in the URL string format

cs Character set. It supports JVM's built-in character set. It is the default value of virtual machine by default.

Return value:

File flow

Example:

httpfile("http://192.168.0.99:9281/p3.dfx()","iso-8859-1") Load the file on the http service "**p3. dfx**", with character set of **iso-8859-1**

icount()

fi.icount()

Description:

Return the total number of records in an index file

Syntax:

***fi*.icount()**

Remark:

Return the total number of records in the index file *fi*

Parameter:

fi The index file

Return value:

Numeric

Example:

	A	
1	**=file("D:\\EMPLOYEE.txt").index(file("D:\\e"),EID)**	Create an index file for the binary file **EMPLOYEE.txt** *e*; the index field is EID. Return **true** after the index file is created
2	**=file ("D:\\e").icount()**	Return the total number of records in the index file

Related concept:

[*f*. **icursor()**]

icursor()

f.icursor()

Description:

Use the index file to retrieve the file content

Syntax:

f.**icursor**(*fi*,*x*;F_i,…)

Remark:

Use the index file *fi* to retrieve records which have the index field values meet the condition *x* from *f*and then return them as a cursor. *x* is optional. Please notice that the comma preceding *x* must not be omitted if omitting *x*. The condition specified by *x* can be automatically identified and according to it the index file is employed. In this case *x* cannot be omitted.

Parameters:

f	Binary file object
fi	Index file
x	Conditional expression of the index field
F_i	Fields that are picked out

Return value:

Cursor

Example:

	A	
1	=file("D:\\EMPLOYEE1.txt").icursor(file("D:\\e"),EID<100;EID,NAME,DEPT)	Use the index file **e** to retrieve records whose **EID** is less than 100 from the file **EMPLOYEE1.txt,** and select out the fields **EID**, **NAME,** and **DEPT.**
2	=file("D:\\EMPLOYEE.txt").icursor(file("D:\\es"),SALARY>=3000&&SALARY<=10000;EID,NAME,DEPT,SALARY)	Multi-field index. Retrieve records whose **SALARY** is not less than 3,000 and not greater than 10,000. Create index file **es** in the order of first **EID** and then **SALARY**.

Related concept:

[*f*. **index()**]

id()

A.id()

Description:

Obtain distinct values by performing *distinct* operation on a sequence.

Syntax:

A.**id**(*x*)

A.**id**([x_1,x_2,......x_i]) Compute the composite distinct values according to multiple fields [x_1,x_2,......x_i], , the return value is a sequence that is composed of these composite distinct values

Remark:

Compute the value of expression *x* against each member of *A* and sort the result, then generate a new sequence composed of the **distinct** values of *x*.

Parameters:

x_i	*distinct* expression
A	*n* sequence

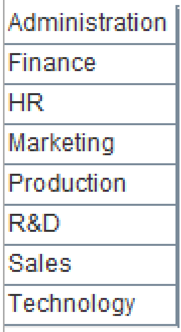
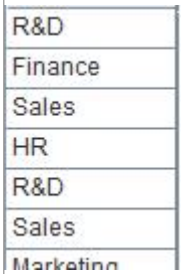
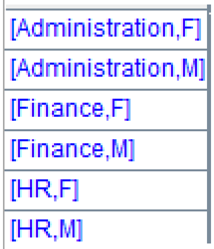
Option:

@o	Without sorting, remove the neighboring duplicate members only

Return value:

The new sequence composed of the **distinct** values of *x*

Example:

	A		
1	=demo.query("select * from EMPLOYEE")		
2	=A1.id(DEPT)	 Administration Finance HR Marketing Production R&D Sales Technology	Sort members in ascending order
3	=A1.id@o(DEPT)	 R&D Finance Sales HR R&D Sales Marketing	No sorting
4	=A1.id([DEPT,GENDER])	 [Administration,F] [Administration,M] [Finance,F] [Finance,M] [HR,F] [HR,M]	Get the distinct values after sorting by **DEPT & GENDER**

5	**=["a","b","c","a"].id()**	Omitting *x* indicates it is the sequence members themselves that will be computed to get the distinct values. Return **["a","b","c"]** as with this case

Related concepts:

A.group(xi,…)

if

if *x*

Description:

The introduction of **if** statement

Syntax:

if *x*

Remark:

Execute the **if** code block if *x* is true and skip **if** code block if *x* is false

Parameter:

x A Boolean expression. If the value is **true**, then execute **if** code block. Otherwise, skip it.

Example:

	A	B	
1	**=6**		**16**
2	**if A1>5**		**if** code block
3		**>A1=A1+10**	

Related Concepts:

if x …{ else { if x}… }

if x … else …

if x else

if *x* …{ else { if *x*}… }

Description:

The introduction of **if** statement

Syntax:

if *x*

… The **if** code block

{else {if *x***}}**

… The **else** or **else if** code block. When **else** or **else if** is not found, then **else** or **else if** code block does not exist.

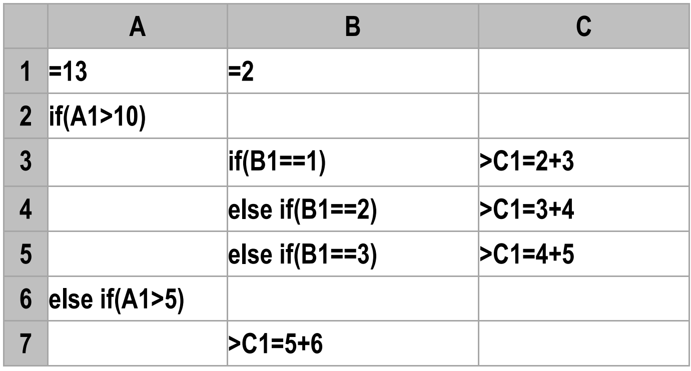
Remark:

If the conditional expression *x* is evaluated as true, then execute the **if** code block, otherwise, skip the **if** code block and execute the code block of **else** or **else if**.

Parameter:

x A Boolean expression. If the result is true, then execute **if** code block, otherwise, execute the **else** or **else if** code block, if any. If no **else** or **else if** code block is available, then the **if** statement terminates.

Example:



	A	B	C
1	=13	=2	
2	if(A1>10)		
3		if(B1==1)	>C1=2+3
4		else if(B1==2)	>C1=3+4
5		else if(B1==3)	>C1=4+5
6	else if(A1>5)		
7		>C1=5+6	

C1 returns **7**

Related Concepts:

if x … else …

if x else

if x

if *x*…else …

Description:

The introduction of **if** statement

Syntax:

if *x* … **else** …

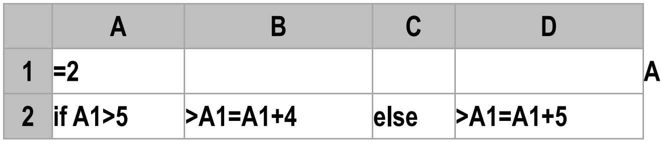
Remark:

Since **if** and **else** are in the same line, if *x* is true, then execute the code between **if** and **else**; if *x* is false, then only execute the code after **else** till the end of thisline.

Parameter:

x A Boolean expression

Example:



	A	B	C	D
1	=2			
2	if A1>5	>A1=A1+4	else	>A1=A1+5

A2 returns **7**

Related Concepts:

if x …{ else { if x}… }

if x else

if x

if *x* else

Description:

The introduction of **if** statement

Syntax:

if *x*

…

else

…

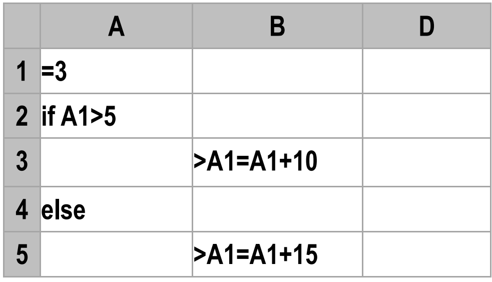
Remark:

Execute **if** code block if *x* is true and execute **else** code block if *x* is false.

Parameter:

x A Boolean expression. If evaluated to be true, then execute **if** code block, otherwise, execute **else** code block.

Example:



	A	B	D
1	=3		
2	if A1>5		
3		>A1=A1+10	
4	else		
5		>A1=A1+15	

if code block

else code block

Related Concepts:

[if x …{ else { if x}… }]

[if x … else …]

[if x]

if()

Description:

Calculate the boolean expression from left to right. if the result is true, return true value; otherwise return default value or false value.

Syntax:

if(*a*) If *a* is true, then return *true*. Otherwise, return *false*

if(*a*,*b*,*c*) If *a* is true, then return *b;* otherwise return *c*, which is *null* by default.

if(x_1:y_1,…,x_k:y_k;*y*) **if**(x_1,y_1,**if**(x_2,y_2,…,**if**(x_k,y_k,*y*)))

Remark:

This function calculates from the left to the right, and returns different values based on the different results of computing the boolean expression. If the result is true, then return *true* and ignore the subsequent computation. If none of the boolean expression results is true, but there is a default value expression, then return the default value; if there is no such a default value expression, return null.

Parameters:

a Boolean expression

b Value expression. If the result of *a* is true, then return the computation of *b*

c Value expresssion. If the result of *a* is false, then return the computation of *c*

x_k Boolean expression

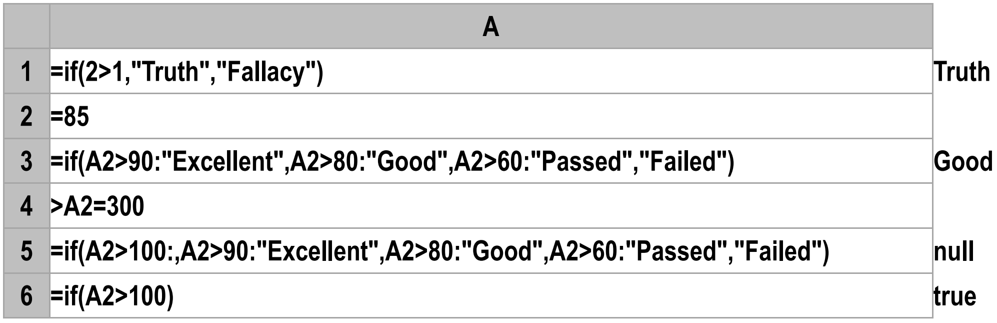
y_k True valueValue expression. If the result of x_k is true, then return the computation of y_k.

y Default value expression. If all the results of boolean expressions are false, then return the computation of this expression.

Return value:

The data type is dynamic, and is determined by the result of value expression. If the corresponding value expression is absent, then return null.

Example:



	A	
1	=if(2>1,"Truth","Fallacy")	Truth
2	=85	
3	=if(A2>90:"Excellent",A2>80:"Good",A2>60:"Passed","Failed")	Good
4	>A2=300	
5	=if(A2>100:,A2>90:"Excellent",A2>80:"Good",A2>60:"Passed","Failed")	null
6	=if(A2>100)	true

Related concepts:

case()

in()

ifa()

Description:

To judge if an object is a sequence.

Syntax:

ifa(*x*)

Remark:

Judge if *x* is a sequence

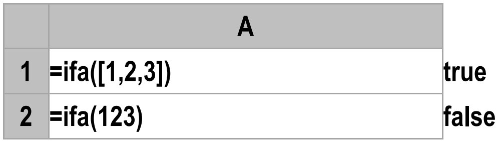
Parameters:

x The object to be judged

Return value:

A boolean value

Example:



	A	
1	=ifa([1,2,3])	true
2	=ifa(123)	false

Related concepts:

ift()

ifr()

ifv()

ifdate()

Description:

Judge if the parameter is a date or a datetime.

Syntax:

ifdate(*exp*)

Remark:

Judge if the parameter *exp* is a date or a datetime.

Parameters:

exp Expression of any data type

Return value:

Boolean

Example:

–	ifdate("2006-10-10")	false
–	ifdate(date("2006-10-10"))	true
–	ifdate(date("2006-10-10 10:20:30"))	true
–	ifdate("20061010")	false
–	ifdate("10:20:30")	false

Related concepts:

iftime()

ifn()

A.ifn()

Description:

Get the first non-null member of a sequence

Syntax:

A.**ifn()** Equivalent to **ifn**(x_1,…,x_n)

Remark:

Get the first non-null member of sequence *A*

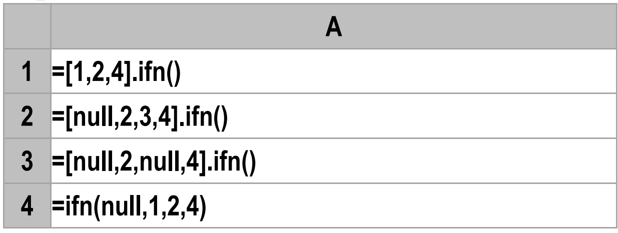
Parameter:

A *n* sequence

Return value:

The first non-null member of the sequence

Example:



	A	
1	=[1,2,4].ifn()	1
2	=[null,2,3,4].ifn()	2
3	=[null,2,null,4].ifn()	2
4	=ifn(null,1,2,4)	1

Related concepts:

A.ifn(x)

A.ifn(*x*)

Description:

Compute *x* with each member of the sequence and return the first non-null member of the new sequence

Syntax:

A.**ifn**(*x*) Equivalent to *A*.(*x*).**ifn**()

Remark:

Compute *x* against sequence *A* by loop and return the first non-null member of the new sequence

Parameters:

A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names

Return value:

The first non-null member of the new sequence

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.ifn(SALARY)	Return the first non-null value of SALARY
3	=A1.(SALARY+100).ifn()	Add 100 to the salary of each employee and return the salary of the first employee

Related concepts:

A.ifn()

ifnumber()

Description:

Judge if the parameter is of numeric data type.

Syntax:

ifnumber(*Exp*)

Remark:

Judge if the parameter *Exp* is of numeric data type

Parameters:

Exp Expression of any data type

Return value:

Boolean

Example:

– ifnumber("abc")	false
– ifnumber("1234")	false
– ifnumber(1234)	true
– ifnumber("1234sss")	false

Related concepts:

ifstring()

ifr()

Description:

Judge if an object is a record.

Syntax:

ifr(*x*)

Remark:

Judge if *x* is a record

Parameter:

x Any data object, for example a constant, an expression, a record, a record sequence, a sequence or a table sequence.

Return value:

true/false

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=ifr(A1(1))	true
3	=ifr(A1)	false
4	=ifr(A1(1).BIRTHDAY)	false

Related concepts:

ift()

ifa()

ifv()

ifstring()

Description:

Judge if the parameter is of string data type

Syntax:

ifstring(*Exp*)

Remark:

Judge if the parameter *Exp* is of string data type

Parameter:

Exp Expression of any data type

Return value:

Boolean

Example:

–	ifstring("abc")	true
–	ifstring(1234)	false
–	ifstring("1980-01-01")	true
–	ifstring(date("1980-01-01"))	false

Related concepts:

ifnumber()

ift()

Description:

Judge if an object is a table sequence.

Syntax:

ift(*x*)

Remark:

Judge if *x* is a table sequence

Parameter:

x Any object, for example a constant, an expression, a record, a record sequence, a sequence or a table sequence.

Return value:

true/false

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=ift(A1(1))	false
3	=ift(A1)	true
4	=ift([1,2,3])	false

Related concepts:

ifr()

ifa()

ifv()

iftime()

Description:

Judge if the parameter is of time data type.

Syntax:

iftime(*exp*)

Remark:

Judge if the parameter *exp* is of time data type.

Parameter:

exp Expression of any data type

Return value:

Boolean

Example:

- iftime("10:20:30") false

– iftime(time("10:20:30"))	true
– iftime("2006-10-10")	false
– iftime("2006-10-10 10:20:30")	false
– iftime("20061010")	false

Related concepts:

ifdate()

ifv()

Description:

Judge if a variable exists

Syntax:

ifv(*v*)

Remark:

This function is mainly used to judge if any variable is added to the current execution environment by the external program when the external program calls the program cellset. Generally, there is no need to judge the defined variable in the cellset script. They are there for sure if you've defined them.

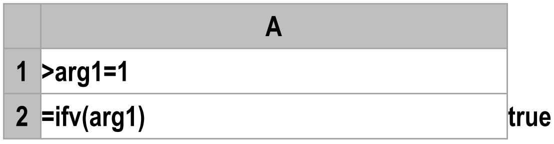
Parameter:

v Parameter name

Return value:

Boolean value

Example:



	A	
1	>arg1=1	
2	=ifv(arg1)	true

Related concepts:

ift()

ifa()

ifr()

import()

S.import()

Description:

Retrieve contents from a string as records and return them in the form of TSeq

Syntax:

S.import()

S.import(F_i:*type*,…;*s*) F_i is the retrieved fields. By default, all fields will be retrieved. *s* is the optional separator. By default, the separator is tab.

Remark:

Retrieve specified or all fields from string *S* and return them as a TSeq.

Parameters:

S The string. Its format: Separate the records with space, and the fields with optional separators. The default separator is tab.

F_i Fields retrieved. By default, all fields will be retrieved.

type Field types include bool, int, long, float, decimal, number, string, date, time and datetime. Data type of the first row will be used by default.

s Optional separator. The default separator is tab.

Options:

@t Take the first row in *f* as the field name. If not using this option, then use _1, and _2,… as the field name.

@j Import records from json strings and resolve them into a TSeq, with *s* being ignored. Of the [{*F*:*v*,…},…], *v* is the value of *F*; it will be quoted if being the string constant, and represented reclusively when it is a sequence or a TSeq.

@x Import records from XML strings and resolve them into a TSeq, with *s* being ignored.

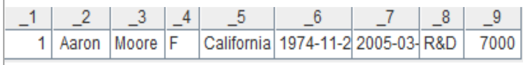
```
<xml>
    <row>
        <F>v</F>
        …
    </row>
    …
</xml>
```

v is represented in the form of […] when it is a sequence. If *v* is a TSeq, then it is represented in such forms recursively.

Return value:

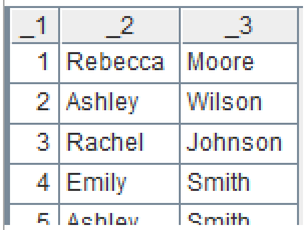
TSeq

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.(~.string())	Convert to the sequence of strings
3	=A2(1).import(;",")	 Select all fields from a specified string in the sequence. Specify comma as the separator, and return a TSeq as the result.
4	=demo.query("select EID,NAME,SURNAME from EMPLOYEE")	
5	=export(A4)	
6	=export@t(A4,EID:id,NAME:name,SURNAME:surname;"|")	

_1	_2	_3	_4	_5	_6	_7	_8	_9
1	Aaron	Moore	F	California	1974-11-2	2005-03-	R&D	7000

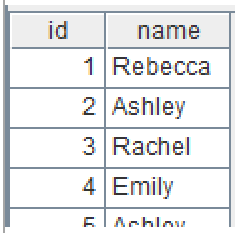
7 =A5.import()



_1	_2	_3
1	Rebecca	Moore
2	Ashley	Wilson
3	Rachel	Johnson
4	Emily	Smith
5	Ashley	Smith

No parameters are given. The default separator will be tab and _1 and _2,… will be taken as field names

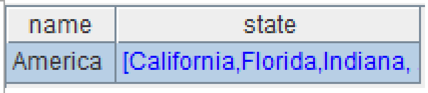
8 =A6.import@t(id:int,name;"|")



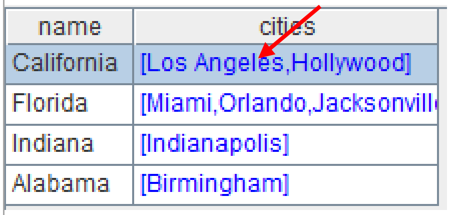
id	name
1	Rebecca
2	Ashley
3	Rachel
4	Emily
5	Ashley

Select fields id and name, with separator “|”

9 ="{name:'America',state:[{name:'California',cities:{city:['Los Angeles','Hollywood']}},{name:'Florida',cities:{city:['Miami','Orlando','Jacksonville']}},{name:'Indiana',cities:{city:['Indianapolis']}},{name:'Alabama',cities:{city:['Birmingham']}}]}".import@j()

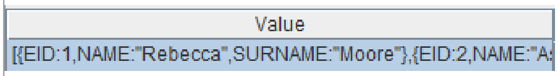


name	state
America	[California,Florida,Indiana,



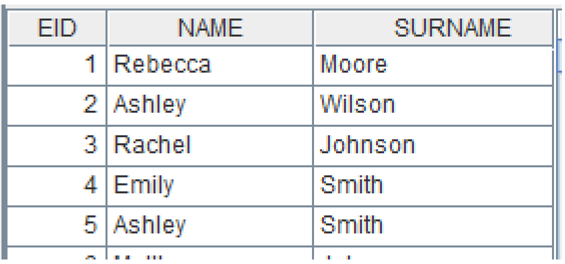
name	cities
California	[Los Angeles,Hollywood]
Florida	[Miami,Orlando,Jacksonvill
Indiana	[Indianapolis]
Alabama	[Birmingham]

10 =export@j(A4)



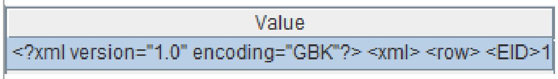
Value
[{EID:1,NAME:"Rebecca",SURNAME:"Moore"},{EID:2,NAME:"A

11 =A10.import@j()



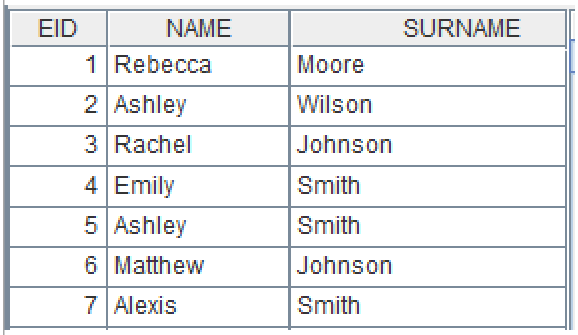
EID	NAME	SURNAME
1	Rebecca	Moore
2	Ashley	Wilson
3	Rachel	Johnson
4	Emily	Smith
5	Ashley	Smith

12 =export@x(A4)



Value
<?xml version="1.0" encoding="GBK"?> <xml> <row> <EID>1

13 =A12.import@x()



EID	NAME	SURNAME
1	Rebecca	Moore
2	Ashley	Wilson
3	Rachel	Johnson
4	Emily	Smith
5	Ashley	Smith
6	Matthew	Johnson
7	Alexis	Smith

Related concepts:

f.export()

f.import()

export()

f.import()

Description:

Read contents from a file object to form a TSeq.

Syntax:

f.**import**()

f.**import**(F_i:*type*,…;*s*,*b*:*e*) For the text file, *b* and *e* are the byte positions. F_i represents the retrieved field, and all fields will be retrieved by default. For the text file, *s* is the optional separator, the default separator is tab.

Remark:

Generate and return a new TSeq composed of records, each of which is composed of the contents of a row of *f*.

Parameters:

f A file object

F_i Fields retrieved. All fields will be retrieved by default.

type Field types include bool, int, long, float, decimal, number, string, date, time and datetime. Data type of the first row will be used by default.

s For text files, it is the optional separator. The default separator is tab.

b If *f* is a text file, then it represents the starting character. Omitting *b* indicates retrieving records from the first character to the e^{th} character. The row, which is not filled with characters to the full, shall still be regarded as one row. If not using @t option, the value of *b* must be 0 or 1 while fetching the first row.

e If *f* is a text file, then it represents the ending character. Omitting *e* indicates retrieving records from the b^{th} character to the last character. The row, which is not filled with characters to the full, shall still be regarded as one row. The “:” cannot be omitted. If *e* is greater than the actual number of rows, then the actual number of rows shall prevail.

If omitting both *b* and *e*, retrieve from the first byte to the last byte of the text file. The “,” can be omitted.

Options:

@t Use the first row of *f* as the field name. If this option is not used, **_1**, **_2**,**…** will be used as field names.

@b Retrieve data from the exported binary file, with the support for parameter F_i, *b* and *e*, and with no support available for parameters *type* and *s*. *@t* will be ignored. Here the binary file is the one stored in the unit of data block and all data of the block where both *b* - the beginning character and *e* - the ending character - falls in, will be retrieved.

@z For the text file, the file will be roughly divided into *e* parts and the part *b* will be retrieved. For the binary file, the *b* and *e* respectively represent the block number and total blocks, and the starting/ending position will be auto-computed programmatically. It is mostly applied to the binary file exported with f.export@g(A,x:F,…;x...). The contents of such a binary file are constructed in the form of one block for one group. In this case, with *b* and *e*, it is ensured that the exported records in one group will not be split apart.

@s Do not split the fields. Read the file as a TSeq composed of single-field string and ignore parameters.

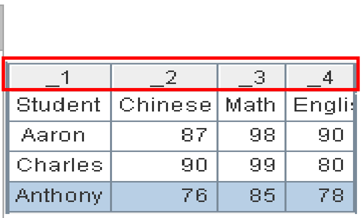
@m Using multithreads will increase data retrieval speed. Members of the result set haven't an definite order and this option can be ignored when parameters *b* and *e* exist. The option is often used to retrieve data from big files. There should be more than one parallel for registration code option and configuration.

Return value:

The new table sequence composed of records, each of which is composed of the contents of a row of file object *f*.

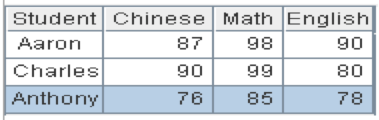
Example:

	A
1	=file("D:\\score.txt").import()



_1	_2	_3	_4
Student	Chinese	Math	Engli
Aaron	87	98	90
Charles	90	99	80
Anthony	76	85	78

	A
2	=file("D:\\score.txt").import@t()

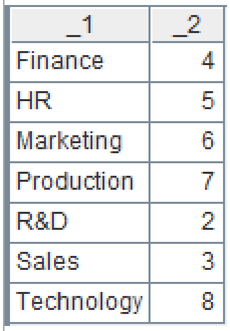


Student	Chinese	Math	English
Aaron	87	98	90
Charles	90	99	80
Anthony	76	85	78

	A
3	=file("D:\\score.txt").import(;,1:72)

With the fields to be imported and the seperator *s* being omitted, retrieve the records from the 1st byte to the 72nd byte.

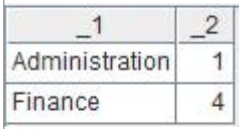
	A
4	=file("D:\\Department2.txt").import(;"|",2)



_1	_2
Finance	4
HR	5
Marketing	6
Production	7
R&D	2
Sales	3
Technology	8

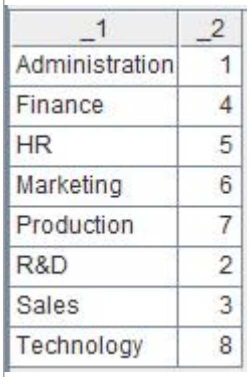
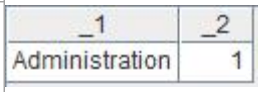
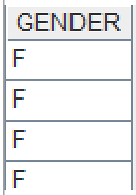
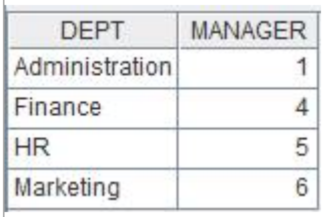
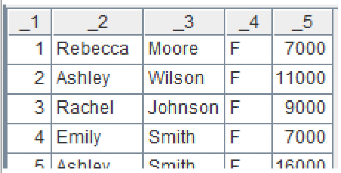
With e being omitted, retreive the records from the 2nd byte to the last byte. Do not retrieve the 1st row since the *b* is not 0

	A
5	=file("D:\\Department2.txt").import(;"|",:20)

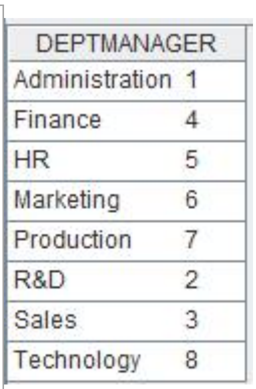


_1	_2
Administration	1
Finance	4

With the fields to be imported and *b* being omitted, retrieve the records from the 1st byte to the 20th byte.

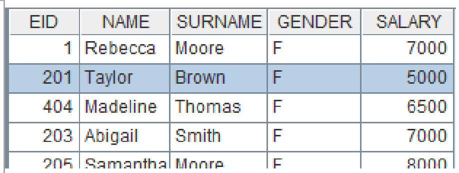
<table>
<tr><td>6</td><td>=file("D:\\Department2.txt").import(;"|")</td><td>
<table>
<tr><th>_1</th><th>_2</th></tr>
<tr><td>Administration</td><td>1</td></tr>
<tr><td>Finance</td><td>4</td></tr>
<tr><td>HR</td><td>5</td></tr>
<tr><td>Marketing</td><td>6</td></tr>
<tr><td>Production</td><td>7</td></tr>
<tr><td>R&D</td><td>2</td></tr>
<tr><td>Sales</td><td>3</td></tr>
<tr><td>Technology</td><td>8</td></tr>
</table>
With the fields to be imported, <i>b</i> and <i>e</i> being omitted, retrieve the record from the 1st byte to the last one.</td></tr>
<tr><td>7</td><td>=file("D:\\Department2.txt").import(;"|",1:3)</td><td>
<table>
<tr><th>_1</th><th>_2</th></tr>
<tr><td>Administration</td><td>1</td></tr>
</table>
Omit the fields to be imported</td></tr>
<tr><td>8</td><td>=file("D:\\ EMPLOYEE.txt").import@b(GENDER;,1:128)</td><td>
<table>
<tr><th>GENDER</th></tr>
<tr><td>F</td></tr>
<tr><td>F</td></tr>
<tr><td>F</td></tr>
<tr><td>F</td></tr>
</table>
Retrieve GENDER field and the data from the 1st byte to the 128th byte from the binary file <b>EMPLOYEE.txt</b>. Since the 128th byte falls in the first data block, all data of this block will be retrieved.</td></tr>
<tr><td>9</td><td>=file("D:\\Department5.txt").import@t(DEPT, MANAGER:int;"/",1:200)</td><td>The contents of <b>Department5.txt</b> are separated with slash and retrieved by the specified fields <b>DEPT</b> and <b>MANAGER</b>

<table>
<tr><th>DEPT</th><th>MANAGER</th></tr>
<tr><td>Administration</td><td>1</td></tr>
<tr><td>Finance</td><td>4</td></tr>
<tr><td>HR</td><td>5</td></tr>
<tr><td>Marketing</td><td>6</td></tr>
</table></td></tr>
<tr><td>10</td><td>=file("D:\\EMPLOYEE1.txt").import@bz(;,1:3)</td><td>
<table>
<tr><th>_1</th><th>_2</th><th>_3</th><th>_4</th><th>_5</th></tr>
<tr><td>1</td><td>Rebecca</td><td>Moore</td><td>F</td><td>7000</td></tr>
<tr><td>2</td><td>Ashley</td><td>Wilson</td><td>F</td><td>11000</td></tr>
<tr><td>3</td><td>Rachel</td><td>Johnson</td><td>F</td><td>9000</td></tr>
<tr><td>4</td><td>Emily</td><td>Smith</td><td>F</td><td>7000</td></tr>
<tr><td>5</td><td>Ashley</td><td>Smith</td><td>F</td><td>16000</td></tr>
</table>
Retrieve file <b>EMPLOYEE1.txt</b>, divide its contents into 3 shares, and get the first share.</td></tr>
</table>

11 `=file("D:\\Department.txt").import@ts()`



DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6
Production	7
R&D	2
Sales	3
Technology	8

12 `=file("D:\\EMPLOYEE.txt").import@bz(;,1:2)`



EID	NAME	SURNAME	GENDER	SALARY
1	Rebecca	Moore	F	7000
201	Taylor	Brown	F	5000
404	Madeline	Thomas	F	6500
203	Abigail	Smith	F	7000
205	Samantha	Moore	F	8000

Retrieve the binary file **EMPLOYEE.txt,** which is exported in the example of *f*.export@*g(A,x:F,…;x…)*, divide its contents into 2 shares, and get the first share. In this case, the same group of exported records will not be split apart.

13 `=file("D:\\orders.txt").import@mt(;",")`

Increase the speed of retrieving data from the big file. The record order in the result is not the same as that in the file.

Note:

Text file format: Separate records with carriage return, and fields with the optional separator. The default separator is the tab.

Related concepts:

f.export()

importxls()

f.importxls()

Description:

Retrieve contents as record from an Excel file object, and return the result in the form of TSeq

Syntax:

f.**importxls**()

f.**importxls**(F_i,…;*s*,*b*: *e*) From *f*, retrieve the data of entire rows from contents of *b* to *e*, and return in the form of TSeq. The "*e*<0" represents the reciprocal number, and "F_i" represents the retrieved fields. By default, all fields will be retrieved. *s* is the name or sequence number of a sheet.

Remark:

From *f*, retrieve the data of each row as record and return in the form of TSeq.

Parameters:

f	EXCEL file object
F_i	Retrieved fields. By default, all fields will be retrieved.
s	Name or sequence number of a sheet. If *s* is omitted, then ";" must be omitted as well.
b	The starting row. If *b* is omitted, the data from the first row to the row *e* will be retrieved. In this case, ":" can be omitted.
e	The ending row. If *e* is omitted, the data will be retrieved from the row *b* to the last row. In this case, ":" cannot be omitted. If *e* is greater than the actual number of rows, then the actual number of rows shall prevail. If both *b* and *e* are omitted, the data will be retrieved from the first row to the last row.

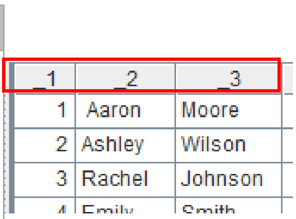
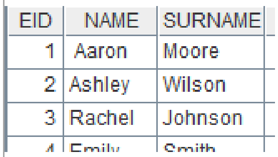
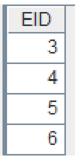
Options:

@t	Take the first row in *f* as the field names. If not using this option, then use _1, and _2,… as the field names.
@x	Use the xlsx format. Determine the format according to the file extension by default. If fail to determine it by this way, then use xls.

Return value:

A TSeq that takes the contents of the file object *f* as its records.

Example:

	A	
1	=file("D:\\EMPLOYEE1.xls").importxls()	 _1 | _2 | _3 1 | Aaron | Moore 2 | Ashley | Wilson 3 | Rachel | Johnson 4 | Emily | Smith
2	=file("D:\\EMPLOYEE2.xls").importxls@t()	 EID | NAME | SURNAME 1 | Aaron | Moore 2 | Ashley | Wilson 3 | Rachel | Johnson 4 | Emily | Smith
3	=file("D:\\EMPLOYEE3.xlsx").importxls@x()	With **@x** option, use xlsx format
4	=file("D:\\EMPLOYEE4.xls").importxls@t(EID; "employee",3:6)	 EID 3 4 5 6 Retrieve the specified field from the sheet of "**employee**", and specify the starting and ending row.
5	=file("D:\\EMPLOYEE4.xls").importxls@t(EID; "employee",:6)	If omitting *b*, then retrieve from the first row, ":" cannot be omitted
6	=file("D:\\EMPLOYEE4.xls").importxls@t(EID; "employee",490:)	If omitting *e*, then retrieve till the last row. The ":" can be omitted.
7	=file("D:\\EMPLOYEE4.xls").importxls@t(EID; "employee",490:-5)	Retrieve from the 490th row to the 5th row from the last

8	=file("D:\\EMPLOYEE4.xls").importxls@t(EID;"employee")	Omit *b* and *e*, and retrieve all rows
9	=file("D:\\EMPLOYEE4.xl").importxls()	Retrieve the "**EMPLOYEE4.xl**" sheet in xls format

Related concepts:

f.import()

in()

A.in()

Description:

Judge if a sequence contains another sequence

Syntax:

A.**in**(*B*)

Remark:

Judge if sequence *B* contains sequence *A*. Return true if the former contains the latter, otherwise, return false.

Parameters:

A Sequence object

B Sequence object

Return value:

true/false

Example:

	A	
1	=[2,3,4,5].in([1,2,3,4,5,6])	**true**

in()

Description:

Based on the passed-in parameter, judge if Parameter 1 is between Parameter 2 and Parameter 3.

Syntax:

in(*x*,*a*:*b*) To judge if *x* is between *a* and *b*. The default include *a* and *b*.

in(*x*,*a*) Equivalent to in(x,a:a)

in(*x*,*a*:) Equivalent to in(x,a:infinity)

in(*x*,:*b*) Equivalent to in(x, infinitesimal:b)

Remark:

To judge if *x* is between *a* and *b*.

Parameters:

x An expression, the result of which must be a numeric string or a number.

a An expression, the result of which must be a numeric string or a number.

b An expression, the result of which must be a numeric string or a number.

Options:

@l	Exclude *a*
@r	Exlude *b*
@b	If *x*<*a* return -1;if *b*<*x* return 1.Otherwise return 0

Return value:

Boolean

Example:

–	in(4,5:6)	false
–	in(4,3:6)	true
–	in(4,4:6)	true
–	in@r(5,5:6)	false
–	in@l(6,5:6)	false
–	in@b(5,6:9)	-1

Related concepts:

if()

case()

index()

T.index()

Description:

Create a hash index for the primary key of a TSeq

Syntax:

T.**index**(*n*)

Remark:

Create a hash table index mapping the primary key of a TSeq. The hash table will be as long as *n*. If *n* is omitted, then the length will be assigned automatically. Performance will be improved by using the index if data query is frequently done by primary key. When the primary key is reset for the TSeq, the index will be cleared. (Even if the primary key is unchanged).

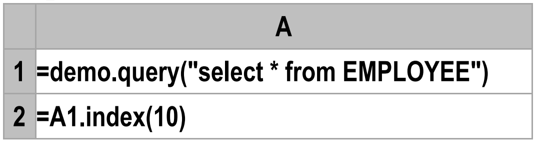
Parameters:

T	TSeq
n	Integer

Return value:

A TSeq whose primary key has the hash table index

Example:



	A
1	=demo.query("select * from EMPLOYEE")
2	=A1.index(10)

f.index()

Description:

Generate an index file for a file

Syntax:

f.index(*fi*,k_i,…)

Remark:

For the file *f*, generate the index file *fi* based on the field k_i. The *f* only supports the binary file. The index file will be created according to the row numbers when k_i is omitted.

Parameters:

f	Binary file object
fi	Index file
k_i	The name of field based on which the index file is generated.

Option:

@a For any updates to the file *f*, append them to the index. By default, a new index will be rebuilt. The index field is unchanged.

Return value:

Boolean value

Example:

	A	
1	=file("D:\\EMPLOYEE.txt").index(file("D:\\e"),EID)	For the binary file **EMPLOYEE.txt,** create the index file **e**, and the index field is **EID.** If the index file is created successfully, return **true**
2	=file("D:\\EMPLOYEE1.txt").index(file("D:\\es"),EID,SALARY)	Create a multi-field index file
3	=file("D:\\EMPLOYEE.txt").index@a(file("D:\\e"),EID)	Append updates to the existing index file

Related concepts:

f. icursor()

insert()

A.insert()

Description:

Insert members into a sequence.

Syntax:

A.insert(*k*,*X*)	Insert the members of *X* before position *k* in *A*, and return *A*.
A.insert(*k*,*x*)	Insert member *x* before position *k* in *A*, and return *A*.

Remark:

Insert member *x* or members of *X* into sequence *A* according to position *k*.

Parameters:

k The position before which one or more members are inserted, when *k*==0, the member(s) will be appended in the end.

A A sequence

X A sequence composed of the members to be inserted

x A member

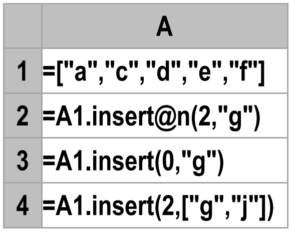
Option:

@n *A* will not be changed, and only a new sequence is returned.

Return value:

The sequence into which you have inserted new members

Example:



	A	
1	=["a","c","d","e","f"]	
2	=A1.insert@n(2,"g")	**[a,g,c,d,e,f]**, option **@n** does not change **A1**.
3	=A1.insert(0,"g")	**[a,c,d,e,f,g]**, the member appended in the end changes **A1**
4	=A1.insert(2,["g","j"])	**[a,g,j,c,d,e,f,g]**, insert the new members before the second member.

Related concepts:

A.delete()

A.modify()

T.insert()

T.insert()

Description:

Insert one or more records into a table sequence.

Syntax:

T.**insert**(*k*) Insert a blank record before the position *k* in the *T*. If *k* is **0**, then append it in the end and return *T*

T.**insert**(*k*,x_i:F_i, …) Insert a record into *T* before the position *k* where values of F_i are x_i and return *T*.

T.**insert**(*k*:*A*,x_i:F_i, …) Insert multiple records into *T* before the position *k* where values of F_i are x_i. The number of the records to be inserted is determined by the length of sequence *A*.

Remark:

Insert one or more records into the table sequence *T*.

Parameters:

k The position before which a member or a record is inserted. When *k*==**0**, the member(s) will be appended in the end.

x_i The field values of F_i before which the new record is to be inserted

F_i The name of field before which a new record is to be inserted; without F_i, it will be the corresponding i^{th} field.

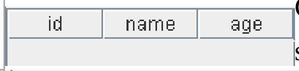
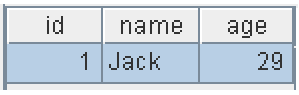
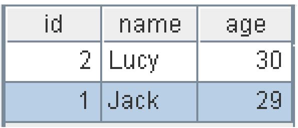
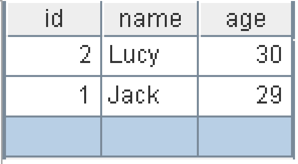
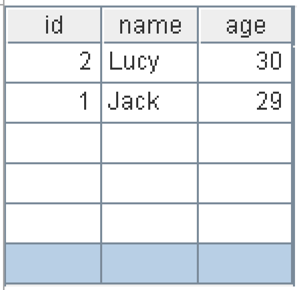
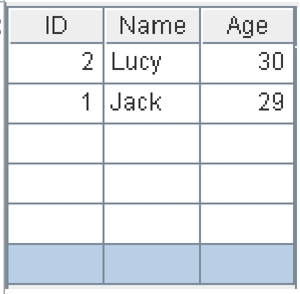
T Table sequence

A A sequence or an integer; if *A* is an integer, then it is equal to **to**(*A*)

Return value:

The table sequence *T* into which you have inserted new records

Example:

	A		
1	=create(id,name,age)	 id name age	Construct an empty table sequence
2	=A1.insert(0,1,"Jack",29)	 id name age 1 Jack 29	Append a record whose field values are **1,Jack**, and **29**
3	=A2.insert(1,2,"Lucy",30)	 id name age 2 Lucy 30 1 Jack 29	Add a record before the first record whose values are **2, Lucy, and 30**.
4	=A3.insert(0)	 id name age 2 Lucy 30 1 Jack 29	Append an empty record in the end.
5	=A4.insert(0:3)	 id name age 2 Lucy 30 1 Jack 29	Append three empty records in the end.
6	=create(ID,Name,Age)		
7	=A6.insert(0:A1,id:ID,name:Name,age:Age)	 ID Name Age 2 Lucy 30 1 Jack 29	Add the records of **A1** into **A6** one by one.

Related concepts:

T.modify()

T.delete()

A.insert()

int()

Description:

This is a data type conversion function. It is used to obtain the integer part of a given numeric value expression or a numeric string, and convert its data type to 32-bit integer .

Syntax:

int(*valueExp***)**

Remark:

The returned result of *valueExp* must be a numeric string or a number.

Parameter:

valueExp An expression, the result of which must be a numeric string or a number.

Return value:

32-bit integer

Example:

–	int("33")	33
–	int("33.999d")	33
–	int(1.5*1.5)	2
–	int(25.67)	25

Related concepts:

float()

decimal()

long()

number()

string()

interval()

Description:

Compute the interval between two date/time values

Syntax:

interval (*datetimeExp1*,*datetimeExp2*)

datetimeExp2- datetimeExp1 **interval** (*datetimeExp1*,*datetimeExp2*)

Remark:

Compute the interval between two date/time data *datetimeExp1* and *datetimeExp2*

Parameters:

datetimeExp1 The date expression whose value is a date/time or a string of standard date/time format

datetimeExp2 The date expression whose value is a date/time or a string of standard date/time format

Options:

@y	Compute the years between two date/time values
@q	Compute the quarters between two date/time values
@m	Compute the months between two date/time values
@s	Compute the seconds between two date/time values
@ms	Compute the milliseconds between two date/time values
@r	Compute the timespan between two date/time values and return a real number value
	By default it computes the days between two date/time values

Return value:

Integer

Example:

- interval(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45")) 1096
- interval@y(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45")) 3
- interval@q(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45")) 12
- interval@m(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45")) 36
- interval@s(datetime("19800227","yyyyMMdd"),datetime("1980-02-27 00:00:45")) 45
- interval@s ("1972-11-08 10:20:30","1972-11-08 10:30:50") 620
- interval@ms(datetime("19800227","yyyyMMdd"),datetime("1980-02-27 00:00:45")) 45,000
- interval@ms("1972-11-08 10:20:30","1972-11-08 10:30:50") 620,000
- interval@r(datetime("19800227","yyyyMMdd"),datetime("1980-02-27 00:00:45")) 5.208333333333333E-4
- interval@r("1972-11-08 10:20:30","1973-11-08 10:30:50") 365.00717592592594
- datetime("19850227","yyyyMMdd")-datetime("1983-02-27 00:00:45") 731

inv()

A.inv(*p*)

Description:

Adjust the order of members of a sequence

Syntax:

A.inv(*p*)

Remark:

Members of *p* are rankings of members of *A*. Adjust the order of members of *A* according to *p*, and return a new *A* after the adjustment

Parameters:

p An integer sequence, members of which are rankings of members of *A*. The number of its members is the same as that of members of *A* , and it is a unique *n* sequence (*n*=*A*.len())

A A sequence or a record sequence

Return value:

Sequence *A* after the adjustment

Example:

	A	
1	[b,c,a,d]	
2	=A1.inv([2,3,1,4])	[a,b,c,d]

Note:

p must be a unique *n* integer sequence,i.e. *n* must be equal to *A*.len()

If *p* has duplicate members or the number of its members is not equal to that of members of *A*, return null.

If the member value of *p* exceeds the maximum sequence number of *A*, return null

The case of tied rankings is not processed

Related concepts:

p.inv(k)

p.inv(*k*)

Description:

To compute the sequence numbers of members of an ISeq in another ISeq.

Syntax:

p.**inv**(*k*)

Remark:

Return the sequence numbers of the numbers from **1** to *k* in integer sequence *p*. Return **0** for the numbers do not exist in *p*.

Parameters:

p	An integer sequence
k	An integer, *k* is *p*.**len()** by default

Return value:

The integer sequence composed of the sequence numbers of the numbers from **1** to *k* in integer sequence *p*

Example:

	A
1	=[1,3,5,7]
2	=A1.inv(4)

Among the four numbers **1, 2, 3** and **4**, the sequence numbers of **1** and **3** in sequence **A1** are **1** and **2**. **2** and **4** do not exist in sequence **A1**, so their sequence numbers are **0**. **[1,0,2,0]** is returned finally.

Related concepts:

A.inv(p)

invoke()

Description:

Invoke the static function of class in the package

Syntax:

invoke(*p.c.f*,a_i,…)

Remark:

Call the static function *f* of class *c* in the package *p*. *p* can be omitted. a_i is the parameter.

Parameters:

p	Package path
c	Class name
f	Static method name
a_i	Parameter

Example:

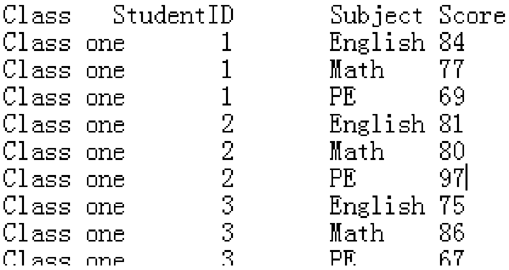
(I) Create the customized class **LineReaders** to implement the **ILineInput** interface. By doing so, users can use the file cursor to retrieve data files of various types. The file path can be used as parameter to be passed to the customized class. By retrieving file contents through stream, implement the record retrieval in single row or skipping of *n* records.

The code is shown below:

```
package api;
import java.io.*;
import com.raqsoft.dm.ILineInput;
import com.raqsoft.dm.Sequence;
public class LineReaders implements ILineInput {
    private String pathName;
    private BufferedReader br;
    private String []title;
 // With title, the title will be returned when calling the readLine at the first time
    public LineReaders(String pathName) {
        this.pathName = pathName;
    }
    // Static method
    public static LineReaders newInstance(String pathName) {
        return new LineReaders(pathName);
    }
    private BufferedReader getInputStream() throws IOException {
        if (br == null) {
            br=new            BufferedReader(new             InputStreamReader(new
FileInputStream(pathName), "UTF-8"));
            // Read into the title if there is one
        }
        return br;
    }
    public Object[] readLine() throws IOException {
        BufferedReader brs=getInputStream();
        String s = brs.readLine();
        // If any title, then return the title when calling the readLine at the first
time
        if (title != null) {
            String []tmp = s.split("\t");
            return tmp;
        }
        Sequence a=new Sequence();
            if(s!=null){
            String[] fields = s.split("\t");
            return fields;
        }
            return null;
    }
    public boolean skipLine() throws IOException {
        BufferedReader brs=getInputStream();
        try {
        String s = brs.readLine();
        if(s!=null){
            return true;
        }
        } catch (EOFException e) {
            return false;
        }
            return false;
    }
    public void close() throws IOException {
        if (br != null) {
            br.close();
            br = null;
        }
    }
}
```

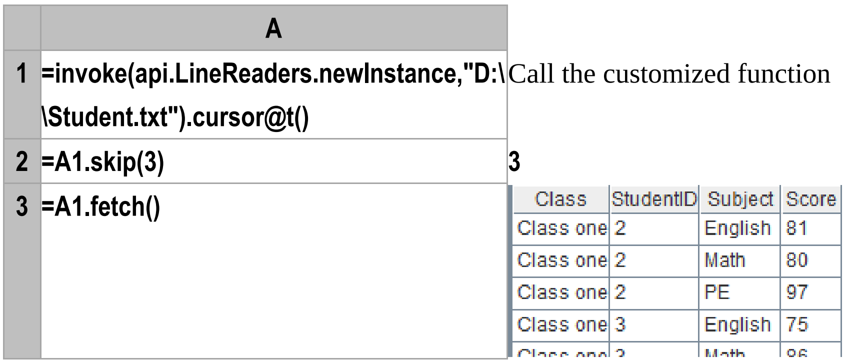
(II) Place such files under the [esProc installation root directory]/classes

(III) The file contents retrieved from cursor are as follows:



```
Class    StudentID      Subject Score
Class one      1        English 84
Class one      1        Math    77
Class one      1        PE      69
Class one      2        English 81
Class one      2        Math    80
Class one      2        PE      97
Class one      3        English 75
Class one      3        Math    86
Class one      3        PE      67
```

(IV) In the esProc cellset file, the customized functions are invoked with **invoke()** function:



	A	
1	=invoke(api.LineReaders.newInstance,"D:\\Student.txt").cursor@t()	Call the customized function
2	=A1.skip(3)	3
3	=A1.fetch()	

Class	StudentID	Subject	Score
Class one	2	English	81
Class one	2	Math	80
Class one	2	PE	97
Class one	3	English	75
Class one	3	Math	86

isalpha()

Description:

Judge if the first character of a string is a letter

Syntax:

isalpha(*s*)

Remark:

Judge if the first character of string *s* is a letter. If *s* is an integer, look it up in the ASCII table to see if its corresponding character is a letter.

Parameter:

s	String/ numeric expression

Return value:

Boolean

Example:

– **isalpha("abc")**	**true**
– **isalpha(97)**	**true**
– **isalpha("@#$")**	**false**
– **isalpha("1@23")**	**false**
– **isalpha("a@23")**	**false**

Related concepts:

isdigit()

isdigit()

Description:

Judge if the first character of a string is a number.

Syntax:

isdigit (*string*)

Remark:

Judge if the first character of the string *string* is a number. If *string* is an integer, look it up in the ASCII table to see if its corresponding character is a number.

Parameter:

string	String/ numeric expression

Return value:

Boolean

Example:

– **isdigit("123")**	**true**
– **isdigit(123)**	**false**
– **isdigit("abc")**	**false**
– **isdigit("123ss")**	**false**

Related concepts:

isalpha()

isect()

A.isect()

Description:

Compute the intersection of all the member sequences of a sequence

Syntax:

A.isect()

Remark:

Members of sequence *A* are also sequences. The function creates a new sequence composed of all common members of the sub-sequences

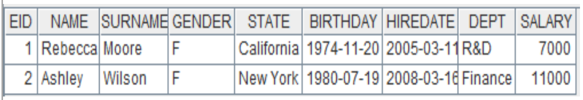
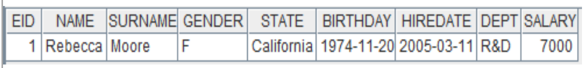
Parameter:

A A sequence whose members are also sequences

Return value:

A sequence

Example:

	A	
1	=[[1,2,3,4,5],[3,7,8]].isect()	[3]
2	=[[1,2,3],[3,2]].isect()	[2,3]
3	=[[1,2,2,3],2].isect()	[2]
4	=demo.query("select top 2 * from EMPLOYEE")	
5	=demo.query("select top 1 * from EMPLOYEE")	
6	=[A4,A5].isect()	[] Since A4 and A5 come from different TSeqs and have different store addresses, so same records are regarded as different members

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000

Related concepts:

A.union()

A.diff()

A.conj()

A.isect(*x*)

Description:

Compute *x* with each member of the sequence whose members are sequences, and then compute intersection of members of the new sequence.

Syntax:

A.**isect(*x*)**

Remark:

Members of sequence *A* are also sequences. Compute *x* on members of sequence *A* by loop and create a new sequence composed of all common members of the sub-sequences.

Parameters:

A A sequence whose members are also sequences

x An expression that returns a sequence

Return value:

A sequence

Example:

	A	
1	=demo.query("select * from EMPLOYEE where GENDER = 'M' order by NAME")	
2	=demo.query("select * from EMPLOYEE where GENDER = 'F' order by NAME")	
3	=[A1,A2].isect(~.(NAME))	Intersection operation between A1 and A2

Related concepts:

A.isect()

islower()

Description:

Judge if the first letter of a string is in lower case.

Syntax:

islower (*string*)

Remark:

Judge if the first character of the string *string* is in lower case. If *string* is an integer, look it up in the ASCII table to see if its corresponding character is a letter in lower case.

Parameter:

string String expression/numeric expression

Return value:

Boolean

Example:

– islower("dgfdsgf")	true
– islower(97)	true
– islower("dsfaAFD")	false
– islower("97ffdsf")	false

Related concepts:

isupper()

isupper()

Description:

Judge if the first letter of a string is in upper case.

Syntax:

isupper (*string* **)**

Remark:

Judge if the first letter of the string *string* is in upper case. If *string* is an integer, look it up in the ASCII table to see if its corresponding character is a letter in upper case.

Parameter:

string	String expression/numeric expression

Return value:

Boolean

Example:

– **isupper("ADSFDGKJ")**	**true**
– **isupper(85)**	**true**
– **isupper("SDsdsSDAS")**	**false**
– **isupper("8ASDS7")**	**false**

Related concepts:

islower()

join()

join()

Description:

Join multiple sequences together.

Syntax:

join(A_i:F_i,x_j,..;…)

Remark:

Join multiple sequences of A_{i} according to the relational field/expression x_j whose value is equal to x_1, and generate a table sequence whose fields are F_i,…. which reference the records of the original record sequences *Ai*. If x_j is omitted, then use the primary key of A_{i} or A_{i} itself.

No matter how many record sequences are mutually related, the equivalence determination is conducted according to the x_1 in A_1. Therefore this is a one-to-many relationship beteen the tables.

Parameters:

F_i	Field name of the result table sequence
A_i	Sequences or record sequences to be joined together
x_j	Relational field/ expression

Options:

@f	Full join. If no matching records are found, then use nulls to correspond

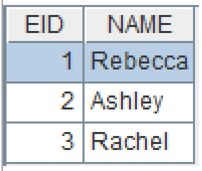
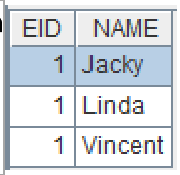
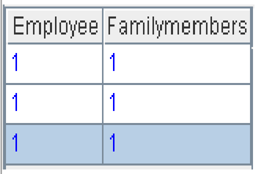
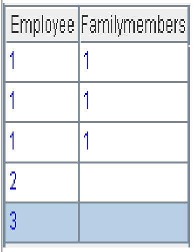
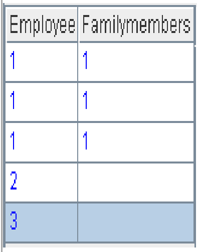
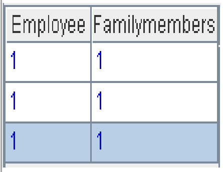
@1 Left join. Please note that this is the number **"1"** instead of the letter **"l"**

@m If all A_i are ordered against $x_{j,}$ then use merge operation to compute

Return value:

The new table sequence whose fields are all referencing ones

Example:

	A	
1	=demo.query("select top 3 EID,NAME from EMPLOYEE")	
2	=demo.query("select top 3 EID,NAME from FAMILY")	
3	=join(A1:Employee,EID;A2:Familymembers,EID)	 Normal join. The non-matching items will be discarded. Every field is a *ref* field pointing to the corresponding record in the original table sequence
4	=join@f(A1:Employee,EID;A2:Familymembers,EID)	 Full join. If no matches, then use the nulls
5	=join@1(A1:Employee,EID;A2:Familymembers,EID)	 Left join. Take the first table sequence as the basis, and use nulls if no matching items are found
6	=join@m(A1:Employee,EID;A2:Familymembers,EID)	

EID	NAME
1	Rebecca
2	Ashley
3	Rachel

EID	NAME
1	Jacky
1	Linda
1	Vincent

Employee	Familymembers
1	1
1	1
1	1

Employee	Familymembers
1	1
1	1
1	1
2	
3	

Employee	Familymembers
1	1
1	1
1	1
2	
3	

Employee	Familymembers
1	1
1	1
1	1

		If all the relational fields are in the same order, then merge operation can be used to compute; If they are not in the same order, then error will occur.

Related concepts:

pjoin()

xjoin()

cs.join()

Description:

Join a cursor with an RSeq or another cursor through foreign keys

Syntax:

cs.**join**(*x:…,A:y:…,z:F,…;x:…,A:y:…,z:F,…;*)

Remark:

Join the field *x,…* of cursor *cs* with the *y,…* field of RSeq/cursor *A* through the foreign key. In ***cs***, compute the expression *z* of *A* to generate a cursor based on all fields of ***cs*** and *F* field. If omitting *y*, then use the primary key of *A*. Read data into memory all at once if *A* is a cursor, which means that the memory capacity should be sufficient for holding *A*'s data.

Parameters:

cs	Cursor
x	Foreign key of cursor *cs*
A	RSeq of cursor
y	Primary key of *A*
z	Field expression of *A*
F	Field name of expression *z*

Option:

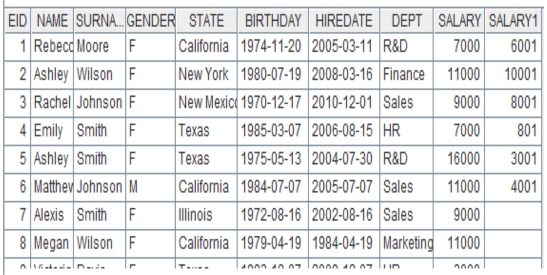
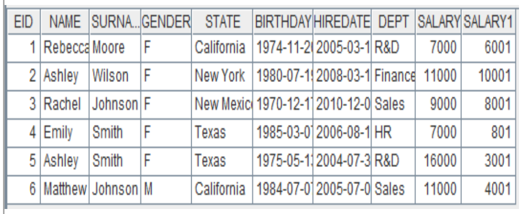
@i Remove those records of foreign key that are not matchable. Wihout the option, their default matching values will be nulls.

Return value:

Cursor

Example:

	A	
1	=demo.cursor("select * from EMPLOYEE order by EID")	
2	=demo.cursor("select * from PERFORMANCE order by EMPLOYEE ID")	
3	=A1.join(ID,A2: EMPLOYEE ID, BONUS+1:SALARY1)	Normal join. By default use nulls to correspond to the non-matching records of foreign key,

4	=A3.fetch()	
5	=demo.cursor("select * from EMPLOYEE order by EID")	
6	=demo.cursor("select * from PERFORMANCE order by EMPLOYEE ID")	
7	=A5.join@i(EID,A6: EMPLOYEE ID, BONUS+1:SALARY1)	Delete the non-matching records of foreign key
8	=A7.fetch()	

EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY	SALARY1
1	Rebecc	Moore	F	California	1974-11-20	2005-03-11	R&D	7000	6001
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000	10001
3	Rachel	Johnson	F	New Mexic	1970-12-17	2010-12-01	Sales	9000	8001
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000	801
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000	3001
6	Matthev	Johnson	M	California	1984-07-07	2005-07-07	Sales	11000	4001
7	Alexis	Smith	F	Illinois	1972-08-16	2002-08-16	Sales	9000	
8	Megan	Wilson	F	California	1979-04-19	1984-04-19	Marketing	11000	

EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY	SALARY1
1	Rebecca	Moore	F	California	1974-11-2	2005-03-1	R&D	7000	6001
2	Ashley	Wilson	F	New York	1980-07-1	2008-03-1	Finance	11000	10001
3	Rachel	Johnson	F	New Mexic	1970-12-1	2010-12-0	Sales	9000	8001
4	Emily	Smith	F	Texas	1985-03-0	2006-08-1	HR	7000	801
5	Ashley	Smith	F	Texas	1975-05-1	2004-07-3	R&D	16000	3001
6	Matthew	Johnson	M	California	1984-07-0	2005-07-0	Sales	11000	4001

Related concepts:

join()

cs.joinx()

join@x()

Description:

Join table sequences corresponding to cursors

Syntax:

join@x(*cs*$_i$:*F*$_i$,*x*$_j$,..;…)

Remark:

Suppose cs_i is ordered by x_j. Perform join operation on multiple cursors cs_i according to the relational field/expression x_j whose value is equal to x_1 and generate a cursor whose fields are F_i,…. F_i are the referencing fields that reference the records in the original cursor sequence cs_i. Note: x_j… only supports the ascending order.

Regardless of the number of cursors being joined, the equivalence determination is conducted according to the x_1 in cs_1. Therefore, it is a one-to-many relationship.

Option:

@f	Full join. If no matching records are found, thenuse nulls to correspond
@1	Left join. Please note it is the number “1”, instead of letter “l”

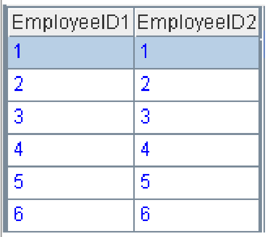
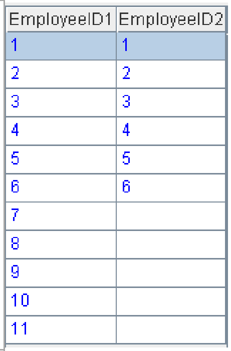
Parameters:

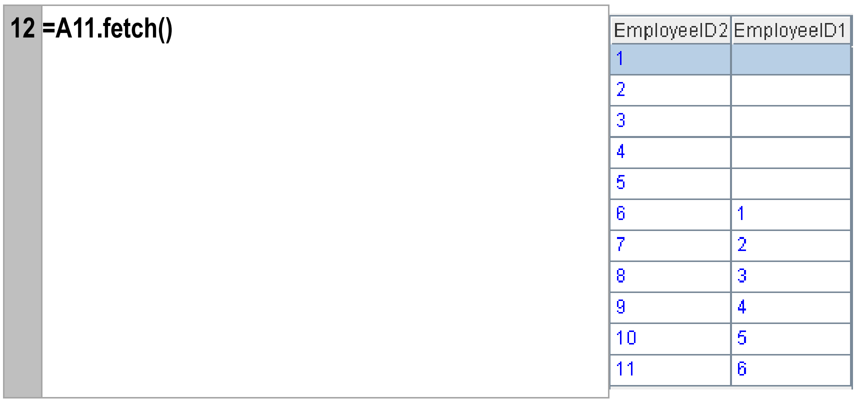
cs_i	Cursor for join
F_i	Field name of the result TSeq
x_j	Relational field/expression

Return value:

Cursor

Example:

<table>
<tr><th></th><th>A</th><th></th></tr>
<tr><td>1</td><td>=demo.cursor("select * from EMPLOYEE order by EID")</td><td></td></tr>
<tr><td>2</td><td>=demo.cursor("select * from PERFORMANCE order by EMPLOYEEID")</td><td></td></tr>
<tr><td>3</td><td>=join@x(A1:EmployeeID1,EID;A2:EmployeeID2,EMPLOYEEID)</td><td>Normal join. Discard the non-matching items, and each field value points to a record in the original cursor</td></tr>
<tr><td>4</td><td>=A3.fetch()</td><td>
<table>
<tr><th>EmployeeID1</th><th>EmployeeID2</th></tr>
<tr><td>1</td><td>1</td></tr>
<tr><td>2</td><td>2</td></tr>
<tr><td>3</td><td>3</td></tr>
<tr><td>4</td><td>4</td></tr>
<tr><td>5</td><td>5</td></tr>
<tr><td>6</td><td>6</td></tr>
</table></td></tr>
<tr><td>5</td><td>=demo.cursor("select * from EMPLOYEE order by EID")</td><td></td></tr>
<tr><td>6</td><td>=demo.cursor("select * from PERFORMANCE order by EMPLOYEEID")</td><td></td></tr>
<tr><td>7</td><td>=join@xf(A5:EmployeeID1,EID;A6: EmployeeID2,EMPLOYEEID)</td><td>Full join. If no matching records, then use nulls to correspond.</td></tr>
<tr><td>8</td><td>=A7.fetch()</td><td>
<table>
<tr><th>EmployeeID1</th><th>EmployeeID2</th></tr>
<tr><td>1</td><td>1</td></tr>
<tr><td>2</td><td>2</td></tr>
<tr><td>3</td><td>3</td></tr>
<tr><td>4</td><td>4</td></tr>
<tr><td>5</td><td>5</td></tr>
<tr><td>6</td><td>6</td></tr>
<tr><td>7</td><td></td></tr>
<tr><td>8</td><td></td></tr>
<tr><td>9</td><td></td></tr>
<tr><td>10</td><td></td></tr>
<tr><td>11</td><td></td></tr>
</table></td></tr>
<tr><td>9</td><td>=demo.cursor("select * from EMPLOYEE order by EID")</td><td></td></tr>
<tr><td>10</td><td>=demo.cursor("select * from PERFORMANCE order by EMPLOYEEID")</td><td></td></tr>
<tr><td>11</td><td>=join@x1(A9:EmployeeID2,EID-5;A10:EmployeeID1,EMPLOYEEID)</td><td>Left join. The first cursor is regarded as the basis. If there are no matching records, then use null to correspond.</td></tr>
</table>



```
12 =A11.fetch()
```

EmployeeID2	EmployeeID1
1	
2	
3	
4	
5	
6	1
7	2
8	3
9	4
10	5
11	6

Related concepts:

join()

joinx()

cs.joinx()

Description:

Join cursor with one or more dimension tables and then return a cursor

Syntax:

cs.**joinx**(*ds*;*x*:…,*V*,*y*:*F*,…;*x*:…;*V*,*y*:*F*,…;…)

Remarks:

Join the cursor *cs* with the piecewise dimension table whose global variable is *V*, and return a new cursor. The dimension space of the piecewise dimension table is *ds*, which is joined on the condition that the results of computational expression *x*:… in the *cs* are the same to the values of the primary key of the piecewise dimension table. If there are multiple primary keys, separate the computational expressions in *cs* with colons, and write them as x_1:x_2:…. Then append the field *F*,… to the records of *cs*. The field value is the result of computing expression *y* based on the piecewise dimension table. With this function, users can also join multiple dimension tables. The joining parameters for multiple dimension tables are separated with semicolons. One thing to note is that the piecewise information about these dimension tables must be included in the dimension space *ds* when using multiple dimension tables.

Parameters:

cs	Cursor
ds	Dimension table space
x	Foreign key of cursor *cs*
V	Piecewise dimension table
y	Computational expression in the dimension table
F	Field name of expression *x*

Options:

@i Remove those records of foreign key that are not matchable. Wihout the option, their default matching values will be nulls.

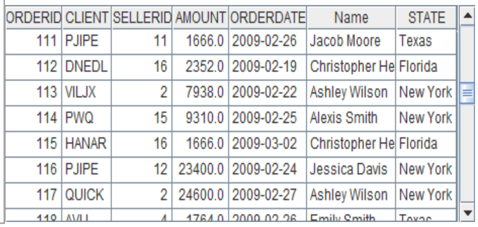
Return value:

Cursor

Example:

DimEmployee.dfx file contents are shown below:

	A
1	=connect("demo")
2	=@DimEmployee=A1.query("SELECT * FROM EMPLOYEE where EID in (?)",ID1)
3	result A2

	A	B	
1	=demo.cursor("select * from SALES")		
2	=["192.168.0.232:8281","192.168.0.147:8281"]		
3	=to(200)	=to(201,300)	
4	=callx@a("DimEmployee.dfx",[A3:B3];A2)		Set global variable on node A2
5	=dims(A2;@DimEmployee,if(?<=100,1,2))		On node A2, generate the piecewise dimension table **@DimEmployee**
6	=A1.joinx(A5;SELLERID,@DimEmployee,NAME+" "+SURNAME:Name,STATE)		Join cursor A1 to the primary key of piecewise dimension table **@DimEmployee** according to the expression **SELLERID**
7	=A6.fetch(200)		Fetch 100 joined records 

ORDERID	CLIENT	SELLERID	AMOUNT	ORDERDATE	Name	STATE
111	PJIPE	11	1666.0	2009-02-26	Jacob Moore	Texas
112	DNEDL	16	2352.0	2009-02-19	Christopher He	Florida
113	VILJX	2	7938.0	2009-02-22	Ashley Wilson	New York
114	PWQ	15	9310.0	2009-02-25	Alexis Smith	New York
115	HANAR	16	1666.0	2009-03-02	Christopher He	Florida
116	PJIPE	12	23400.0	2009-02-24	Jessica Davis	New York
117	QUICK	2	24600.0	2009-02-27	Ashley Wilson	New York
118	AVLI	4	1764.0	2009-02-26	Emily Smith	Texas

Related concepts:

join()

cs.join()

left()

Description:

Get the substring to the left of a string

Syntax:

left(*string*, *n*)

Remark:

Get the substring to the left of string *string*, the length of which is *n*.

Parameters:

string	Get the source string of the substring
n	Get the length of the substring

Return value:

Character

Example:

- left("abcdefg",3)   "abc"

Related concepts:

mid()

right()

len()

len()

Description:

Compute the length of string

Syntax:

len(*s*)

Remark:

Compute the length of string *s*

Parameter:

s   String for which you want to compute the length

Return value:

Integer

Example:

- len("adfg")   4
- len(" abd ")   5

A.len()

Description:

Get the length of a sequence

Syntax:

A.len ()

Remark:

Get the length of sequence *A*.

Parameter:

A   Sequence object

Return value:

Integer

Example:

	A	
1	=[1,2,4]	
2	=A1.len()	3
3	=["a","b"].len()	2
4	=[]	
5	=A4.len()	0

lg()

Description:

Compute the logarithm with 10 as the base

Syntax:

lg(*numberExp*)

Remark:

Compute *numberExp*'s logarithm with 10 as the base

Parameters:

numberExp	Data whose logarithm with 10 as the base is to be computed

Return value:

Numeric

Example:

– **lg(54)**	**1.7323937598229684**

Related concept:

ln()

like()

Description:

Judge if a string matches a format string.

Syntax:

like(*stringExp*, *formatExp* **)**

Remark:

Judge if the string *stringExp* matches the format string *formatExp* (**"*"** is to match 0 or multiple characters; **"?"** is to match single character). The escape character can be used to match **"*"**, for example, the result of **like ("abc*123", "abc*")** is true.

Parameters:

stringExp	A string expression
formatExp	A format string expression

Option:

@c	The matching is case-insensitive. It is case-sensitive by default

Return value:

Boolean

Examples:

– **like("abc123", "abc*")**	**true**
– **like("abc123", "abc1?3")**	**true**
– **like("abc123", "abc*34")**	**false**
– **like("abc123", "ABC*")**	**false**
– **like@c("abc123", "ABC*")**	**true**
– **like ("abc*123", "abc*")**	**true**

ln()

Description:

Compute the natural logarithm of the parameter

Syntax:

ln(*numberExp***)**

Remark:

Compute the natural logarithm of parameter *numberExp*

Parameter:

numberExp	Data for which you want to compute the natural logarithm

Return value:

Numeric

Example:

– **ln(54)**	**3.9889840465642745**

Related concept:

lg()

long()

Description:

Convert a string or a number to a 64-bit long integer.

Syntax:

long(*stringExp***)**

long(*numberExp***)**

Remark:

The value of *stringExp* must be a string that consists of a long integer which is less than or equal to 64 bit. For a value more than 64 bits, the result of **long(***stringExp***)** will be approximate and its decimal part, if any, will be truncated.

The value of *numberExp* must be a long integer which is less than or equal to 64 bit. For a value more than 64 bits, the result of **long(***numberExp***)** will be approximate and its decimal part, if any, will be truncated.

Parameters:

stringExp	The string expression you want to return as a long integer.
numberExp	The number you want to return as a long integer. If the number contains decimal part, the decimal part will be truncated.

Return value:

64-bit long integer

Examples:

– **long("1234567")**	**1234567**
– **long(1234567.789)**	**1234567**

Related concepts:

float()
int()
decimal()
number()
string()

lookup()

A.lookup()

Description:

Locate the position(s) of one or more members in a sequence, and get the members in the position(s) from another sequence.

Syntax:

A.**lookup(**A_i:x_i,…**)**

Remark:

Locate the position(s) of member x_i in A_i, acquire the intersection of these positions if needed and return the member(s) of *A* in the position or these positions.

This function can be used after the alignment of the main table and the sub table. For example, the main table is an **Employee** table and the sub table is a **Department** table, after the alignment of the two tables according to the **Dept**, a certain department can be found in the **Department** table first, then use its position to find the employees of this department in the **Employee** table.

Option:

@a Return all members found in the above-mentioned position; without the function return only the first member.

Parameters:

A_i	A sequence
x_i	tMembers of A_i
A	The target sequence

Return value:

A sequence

Example:

- The search after the alignment of main table and sub table

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=demo.query("select * from EMPLOYEE")	
3	=A1.align(A2:DEPT,DEPT)	Align **EMPLOYEE** to **A2** by **DEPT**
4	=A3.lookup(A2.(DEPT):"R&D")	Return the **EMPLOYEE** record whose **DEPT** is **"R&D"**

- The search of multiple tables

	A

1	=demo.query("select * from PERFORMANCE")	
2	=demo.query("select * from ATTENDANCE")	
3	=demo.query("select * from EMPLOYEE")	
4	=join(A3:EMPLOYEE,EID;A2:ATTENDANCE,EMPLOYEEID;A1:PERFORMANCE,EMPLOYEEID)	
5	=A3.lookup@a(A4.(ATTENDANCE).(ABSENCE):1, A4.(PERFORMANCE).(EVALUATION):0.75)	Return the EMPLOYEE records whose ABSENCE equals 1 and EVALUATION equals 0.75

loop()

A.loop(*x*;*a*;*c*)

Description:

Iterative loop of an RSeq

Syntax:

A.loop(*x*;*a*;*c*)

Remark:

Loop record sequence *A*, **~~** is the result of last running of *x*. The default initial value is *a*. On each run of **~~**, the result of *x* will be reassigned to *a*, and *a* is null by default. If the result of expression *c* is true, then break off the loop.

Parameters:

a	Initial value
x	Expression
A	Sequence or RSeq
c	An expression that returns true/false

Return value:

Sequence or value of a certain member

Example:

	A	
1	[2,222,22,122,2222]	
2	=A1.loop(~*2)	[4,444,44,244,4444]
3	=A1.loop (~~*2;3)	[6,12,24,48,96]
4	=A1.loop(~~*2;5;~>200)	[10]
5	=demo.query("select * from SALES")	
6	=A4.derive(A5.(AMOUNT).loop(~~+~;)(#):Cumulation)	Cumulative sales

Related concept:

A.loops()

loops()

A.loops(*x*;*a* ;*c*)

Description:

Perform cyclic iteration over an RSeq and return the result of the last running of *x*

Syntax:

A. **loops**(*x*;*a*;*c*)

Remarks:

Loop RSeq *A*, ~~ is the result of last running of *x*. Default initial value is *a*. On each run of **~~**, the result of *x* will be reassigned to *a*, and *a* is null by default. Lastly, return the last computational result of *x*. If the result of expression *c* is true, then break off the loop.

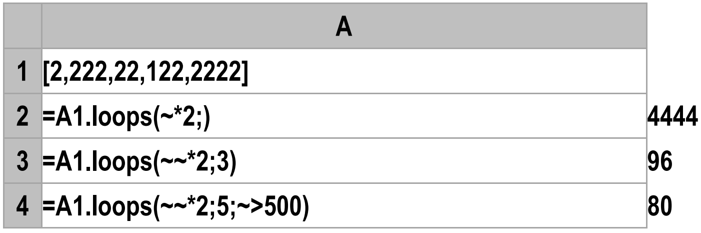
Parameters:

a	Initial value
x	Expression
A	Sequence/RSeq
c	An expression that returns true/false

Return value:

Sequence or value of a certain member

Example:



	A	
1	[2,222,22,122,2222]	
2	=A1.loops(~*2;)	4444
3	=A1.loops(~~*2;3)	96
4	=A1.loops(~~*2;5;~>500)	80

Related concepts:

A.loop()

lower()

Description:

Convert all characters of a string to lower case

Syntax:

lower(*s*)

Remark:

Convert all characters of a string to lower case

Parameter:

s	The string you want to convert to the lower case

Return value:

Character

Example:

- **lower("ABCdef")** **"abcdef"**
- **lower("defABC")** **"defabc"**

Related concepts:

upper()

m()

A.m()

Description:

Get members at specified positions.

Syntax:

A.**m(***i***)** -*n*<=*i*<=*n* and *i* is not equal to *0*; For 1<=*i*<=*n*, it indicates getting the i^{th} member; For -*n*<=*i*<=-*1*, it indicates getting the i^{th} member by counting backwards.

A.**m(***P***)** *P* is the *n*-integer sequence whose length is *m*, the member values of which should be larger than -*n* and less than *n*, but not equal to 0**.**

Remark:

A is an *n* sequence. Get members at specified positions. The function is generally used to get the sequence members reversely.

Parameters:

A	A sequence expression
i	An integer
P	the *n* integer sequence whose length is *m* (its member values are larger than or equal to –*n*, or less than or equal to *n*, but not equal to 0)

Options:

@r	Turn back the position exceeding the boundary of *A*, that is, to set *i***=if(***i***%***n***==0,***n***,***i***%***n***)**, where *n* is the length of *A*.
@0	The position exceeding the boundary of *A* will be ignored.

Return value:

Members at the specified positions in sequence *A*

Example:

	A	
1	[a,b,c,d,e,f,g,h,i,j]	
2	=A1.m(2)	b
3	=A1.m(-2)	i
4	=A1.m([2,3])	[b,c]
5	=A1.m([-2,-3])	[i,h]
6	=A1.m@0([5,12])	[e]
7	=A1.m@r([5,12])	[e,b]

Related concepts:

A.p()

max()

A.max()

Description:

Compute the maximum value of all the non-null members in a sequence

Syntax:

A.**max()** Equivalent to **max**(x_1,…,x_n)

Remark:

Compute the maximum value of all the non-null members in sequence *A*. Please note that this function doesn’t apply to a sequence whose members are not of the same data type

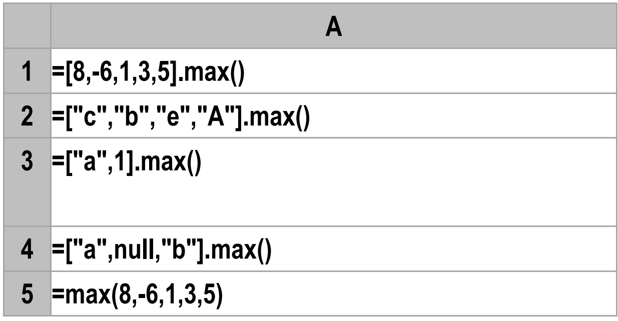
Parameter:

A	A sequence

Return value:

The maximum value of all members in sequence *A*

Example:



	A	
1	=[8,-6,1,3,5].max()	8
2	=["c","b","e","A"].max()	"e"
3	=["a",1].max()	Error message is displayed because the members are of different data types
4	=["a",null,"b"].max()	"b"
5	=max(8,-6,1,3,5)	8

Related concepts:

A.sum()

A.avg()

A.min()

A.count()

A.max(x)

A.variance()

A.max(*x*)

Description:

Compute *x* with each member of the sequence and then compute the maximum value of the members of the new sequence

Syntax:

A.**max**(*x*) Equivalent to *A*.(*x*).**max**()

Remark:

Compute *x* against sequence *A* by loops and return the maximum value of members of the results

Parameters:

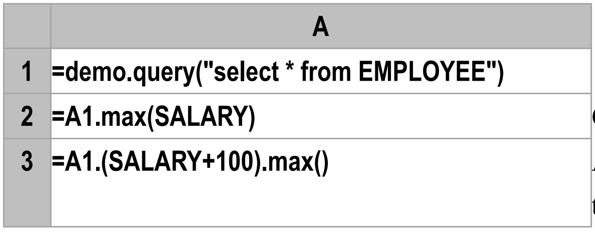
A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names

Return value:

The maximum value of members of the new sequence got by performing computations on members of sequence *A*

Example:



	A
1	=demo.query("select * from EMPLOYEE")
2	=A1.max(SALARY)
3	=A1.(SALARY+100).max()

Compute the highest salary

Add 100 to the salary of each employee and then compute the highest salary

Related concepts:

A.max()

maxif()

A.maxif()

Description:

Locate all the positions of a member in a sequence, (if needed, get the intersection sets of positions of different members of different sequences) and get the maximum of the members at these positions in another sequence.

Syntax:

A.**maxif(**A_i:x_i,…**)**

Remark:

Locate all positions of member x_i or those of all members of subsequence x_i in sequence A_i and return the maximum of members at these positions in sequence *A*; if there are more than one pair of A_i:x_i, find the positions of each x_i in its A_i and get intersection of these sets of positions, then return the maximum of members at these intersection positions in sequence *A*.

Parameters:

A_i A sequence

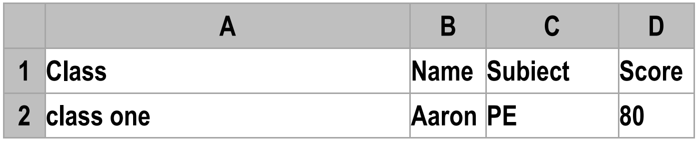
x_i Members in A_i or a sequence composed of members of A_i

A The target sequence

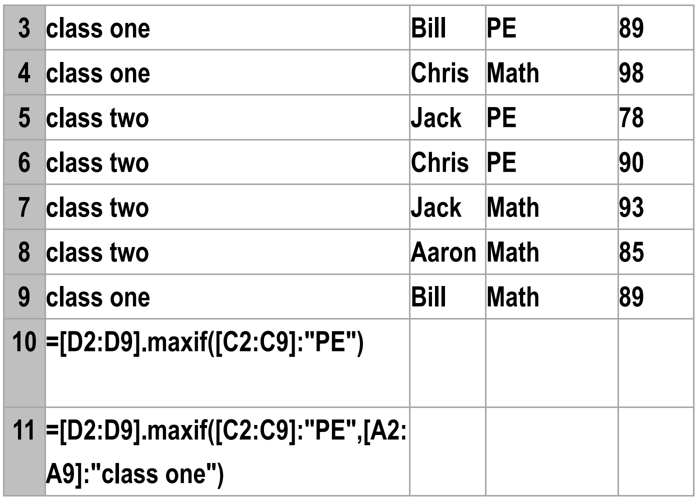
Return value:

The maximum value of the members in those result positions in *A*

Example:



	A	B	C	D
1	Class	Name	Subiect	Score
2	class one	Aaron	PE	80



3	class one	Bill	PE	89	
4	class one	Chris	Math	98	
5	class two	Jack	PE	78	
6	class two	Chris	PE	90	
7	class two	Jack	Math	93	
8	class two	Aaron	Math	85	
9	class one	Bill	Math	89	
10	=[D2:D9].maxif([C2:C9]:"PE")				**90**, search with a single condition
11	=[D2:D9].maxif([C2:C9]:"PE",[A2:A9]:"class one")				**89**, search with composite condition

Related concepts:

A.countif(Ai:xi,…)

A.avgif(Ai:xi,…)

A.minif(Ai:xi,…)

A.sumif(Ai:xi,…)

maxp()

A.maxp()

Description:

Pick out the maximum member of a sequence.

Syntax:

A.**maxp(** *x*)

Remark:

Compute expression *x* against each member of sequence *A* and return the member which makes the value of expression *x* maximum

Options:

@1	Return the first member that fulfills the conditions.
@a	Return all the members that fulfill the conditions. By default, it is **@1.**
@z	Search the members from back to front

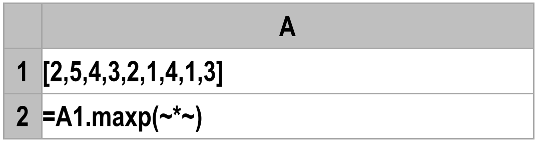
Parameters:

A	A sequence
x	The expression to be computed

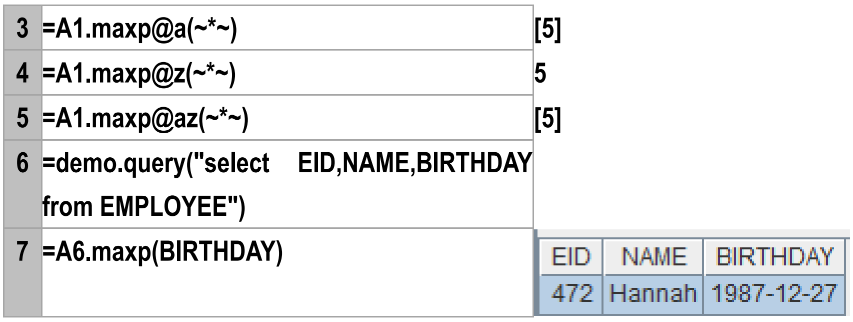
Return value:

The member which makes the value of the expression *x* maximum

Example:



	A	
1	[2,5,4,3,2,1,4,1,3]	
2	=A1.maxp(~*~)	**5, @1** is the default option



3	=A1.maxp@a(~*~)	[5]
4	=A1.maxp@z(~*~)	5
5	=A1.maxp@az(~*~)	[5]
6	=demo.query("select EID,NAME,BIRTHDAY from EMPLOYEE")	
7	=A6.maxp(BIRTHDAY)	EID NAME BIRTHDAY 472 Hannah 1987-12-27

Related concepts:

A.pmax()

A.minp()

merge()

A.merge()

Description:

Merge all sorted *A(i)*s and keep the new sequence in order; If x_i is omitted, then use the sequence itself.

Syntax:

A.**merge** (x_i,…)

Remark:

Merge all *A(i)*s , as shown with *A(i)*|…., *A(i)* is sorted by [x_i,…]. The omission of x_i indicates the sequence itself will be used. If *A* is a sequence composed of RSeqs, then x_i must be set to be merged by the specified field(s); If x_i is a null, then perform merge by the primary key. Every *A(i)* must be of the same structure.

Parameters:

A	Multiple sequences of the same structure
x_i	Members of *A*(i). If perform merge by multiple fields, use the comma to separate them, for example, x_1, x_2. . .

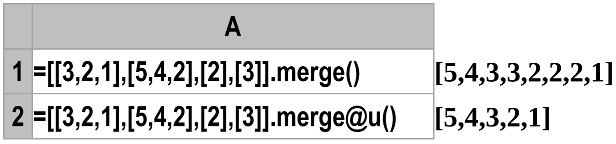
Options:

@u	The members of sequence *A(i)* will be merged as a new sequence in proper order and the duplicate members will be removed.
@i	Return the sequence which is composed of same members of sequence *A(i)s*
@d	A new sequence generated by removing members of *A(2)*&…*A(n)* from sequence *A(1)*.

Return value:

A sequence in the same order as *A(i)*

Example:



	A	
1	=[[3,2,1],[5,4,2],[2],[3]].merge()	[5,4,3,3,2,2,2,1]
2	=[[3,2,1],[5,4,2],[2],[3]].merge@u()	[5,4,3,2,1]

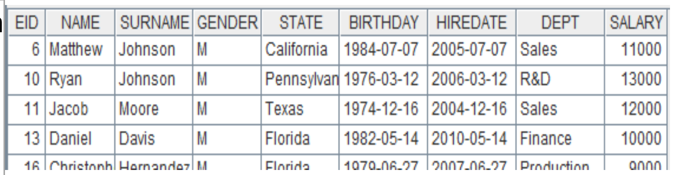
3 =[[3,2,1],[5,3,2]].merge@i()

[3,2]

4 =[[3,2,1],[5,4,2],[2],[3]].merge@d()

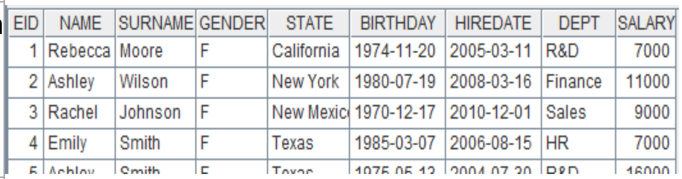
[1]

5 =demo.query("select * from EMPLOYEE where GENDER = 'M'")



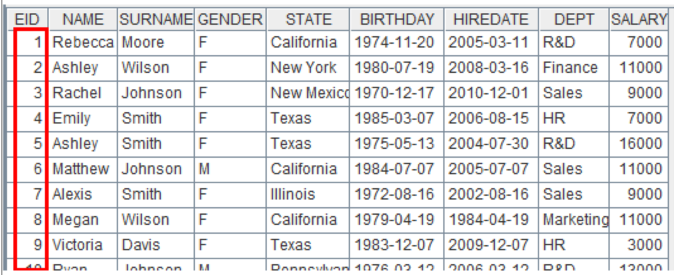
EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
6	Matthew	Johnson	M	California	1984-07-07	2005-07-07	Sales	11000
10	Ryan	Johnson	M	Pennsylvan	1976-03-12	2006-03-12	R&D	13000
11	Jacob	Moore	M	Texas	1974-12-16	2004-12-16	Sales	12000
13	Daniel	Davis	M	Florida	1982-05-14	2010-05-14	Finance	10000
16	Christoph	Hernandez	M	Florida	1979-06-27	2007-06-27	Production	9000

6 =demo.query("select * from EMPLOYEE where GENDER = 'F'")



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexic	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000

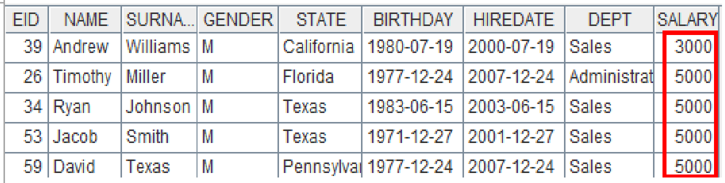
7 =[A5,A6].merge()



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000
6	Matthew	Johnson	M	California	1984-07-07	2005-07-07	Sales	11000
7	Alexis	Smith	F	Illinois	1972-08-16	2002-08-16	Sales	9000
8	Megan	Wilson	F	California	1979-04-19	1984-04-19	Marketing	11000
9	Victoria	Davis	F	Texas	1983-12-07	2009-12-07	HR	3000
10	Ryan	Johnson	M	Pennsylvan	1976-03-12	2006-03-12	R&D	13000

If x_i is omitted, this table sequence will be ordered by the primary key

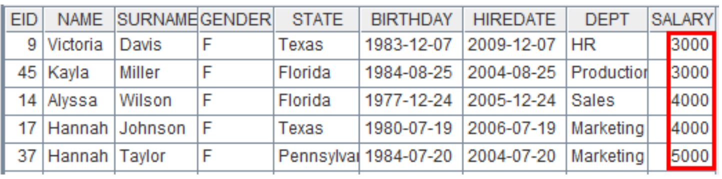
8 =A5.sort(SALARY:1)



EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
39	Andrew	Williams	M	California	1980-07-19	2000-07-19	Sales	3000
26	Timothy	Miller	M	Florida	1977-12-24	2007-12-24	Administrat	5000
34	Ryan	Johnson	M	Texas	1983-06-15	2003-06-15	Sales	5000
53	Jacob	Smith	M	Texas	1971-12-27	2001-12-27	Sales	5000
59	David	Texas	M	Pennsylvai	1977-12-24	2007-12-24	Sales	5000

Sort by **SALARY** field

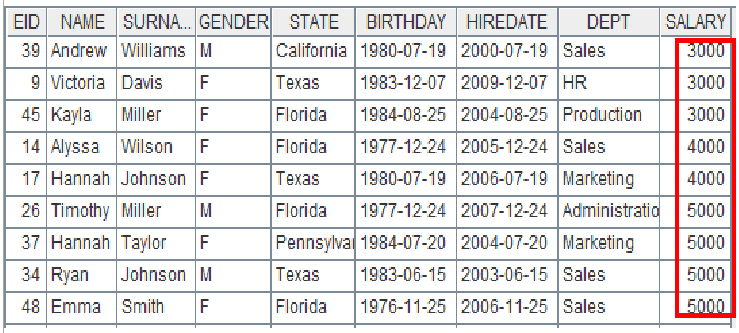
9 =A6.sort(SALARY:1)



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
9	Victoria	Davis	F	Texas	1983-12-07	2009-12-07	HR	3000
45	Kayla	Miller	F	Florida	1984-08-25	2004-08-25	Productior	3000
14	Alyssa	Wilson	F	Florida	1977-12-24	2005-12-24	Sales	4000
17	Hannah	Johnson	F	Texas	1980-07-19	2006-07-19	Marketing	4000
37	Hannah	Taylor	F	Pennsylvai	1984-07-20	2004-07-20	Marketing	5000

Sort by **SALARY** field

10 =[A8,A9].merge(SALARY,SALARY)



EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
39	Andrew	Williams	M	California	1980-07-19	2000-07-19	Sales	3000
9	Victoria	Davis	F	Texas	1983-12-07	2009-12-07	HR	3000
45	Kayla	Miller	F	Florida	1984-08-25	2004-08-25	Production	3000
14	Alyssa	Wilson	F	Florida	1977-12-24	2005-12-24	Sales	4000
17	Hannah	Johnson	F	Texas	1980-07-19	2006-07-19	Marketing	4000
26	Timothy	Miller	M	Florida	1977-12-24	2007-12-24	Administratio	5000
37	Hannah	Taylor	F	Pennsylvai	1984-07-20	2004-07-20	Marketing	5000
34	Ryan	Johnson	M	Texas	1983-06-15	2003-06-15	Sales	5000
48	Emma	Smith	F	Florida	1976-11-25	2006-11-25	Sales	5000

This table sequence is ordered by the **SALARY** field

11 =demo.query("select * from EMPLOYEE where GENDER = 'M' and EID<15")

12 =demo.query("select * from

	EMPLOYEE where GENDER = 'M' and EID>=15")	
13	=[A11,A12].merge(EID,GENDER)	This table sequence is ordered by the **EID** field and **GENDER** field.

CS.merge@x()

Description:

CS is a sequence of cursors. Perform merge operation on the sequences output from its members.

Syntax:

CS.**merge@x**(x_i,…)

Remark:

CS is a sequence of cursors. From each cursor a sequence of records can be output. Perform merge operation by x_i based on these output sequences. Members of each sequence output from each cursor must be of the same structure.

Parameters:

CS A sequence consisting of cursors

x_i A sequence. If perfoming merge by multiple fields, use comma to separate them, for example, x_1, x_2. . .

Options:

@u Combine members of *CS*, the sequence of cursors, in a certain order to create a new cursor. All duplicate members are included by default.

@i Return a cursor composed of common members of members of *CS*, the sequence of cursors.

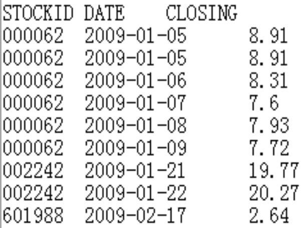
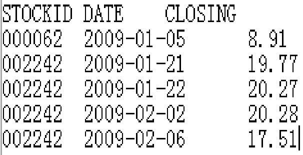
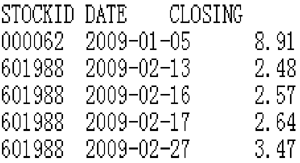
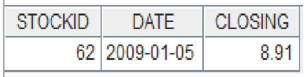
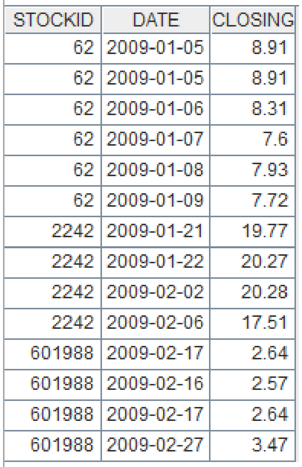
@d The new cursor created by removing members of *CS*2&…*CSn* from *CS*1.

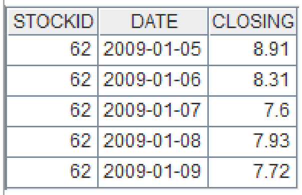
Return value:

Cursor

Example:

	A			
1	=demo.cursor("select * from STOCKRECORDS")			Retrieve data and return a cursor
2	for			
3		=A1.fetch(500)		Fetch data from the cursor
4		if B3==null	break	
5		else		
6			=B3.sort(STOCKID)	Sort by STOCKID
7			=file("D:\\"+"a"+string(A2)+".txt").export@t(C6)	The data retrieved each time will be stored in the file.
8			=B1=B1|file("D:\\"+"a"+string(A2)+".txt")	File object sequence
9	for B1			
10		=A9.cursor@t()		
11		=C1=C1|B10		File cursor sequence

<table>
<tr><td>12</td><td colspan="3">=C1.merge@x(STOCKID)</td><td>Merge the cursor sequence members by <b>STOCKID</b> of each member.</td></tr>
<tr><td>13</td><td colspan="3">=A12.fetch()</td><td>Get the post-mergence records</td></tr>
<tr><td>14</td><td colspan="3">=directory@p("D://*.txt")</td><td>txt file is as follows:

STOCKID DATE CLOSING
000062 2009-01-05 8.91
000062 2009-01-05 8.91
000062 2009-01-06 8.31
000062 2009-01-07 7.6
000062 2009-01-08 7.93
000062 2009-01-09 7.72
002242 2009-01-21 19.77
002242 2009-01-22 20.27
601988 2009-02-17 2.64

STOCKID DATE CLOSING
000062 2009-01-05 8.91
002242 2009-01-21 19.77
002242 2009-01-22 20.27
002242 2009-02-02 20.28
002242 2009-02-06 17.51

STOCKID DATE CLOSING
000062 2009-01-05 8.91
601988 2009-02-13 2.48
601988 2009-02-16 2.57
601988 2009-02-17 2.64
601988 2009-02-27 3.47</td></tr>
<tr><td>15</td><td>for A14</td><td></td><td></td><td></td></tr>
<tr><td>16</td><td></td><td>=file(A15).c▯rsor@t()</td><td>=B14=B14|B16</td><td></td></tr>
<tr><td>17</td><td>=B14.merge@xi(STOCKID)</td><td>=A17.fetch()</td><td></td><td>
STOCKID DATE CLOSING
62 2009-01-05 8.91
Return a cursor composed of common members.</td></tr>
<tr><td>18</td><td colspan="2">=directory@p("D://*.txt")</td><td></td><td>The txt file is
the same as the above</td></tr>
<tr><td>19</td><td>for A18</td><td></td><td></td><td></td></tr>
<tr><td>20</td><td></td><td>=file(A19).cursor@t()</td><td>=B18=B18|B20</td><td></td></tr>
<tr><td>21</td><td>=B18.merge@xu(STOCKID)</td><td>=A21.fetch()</td><td></td><td>
STOCKID DATE CLOSING
62 2009-01-05 8.91
62 2009-01-05 8.91
62 2009-01-06 8.31
62 2009-01-07 7.6
62 2009-01-08 7.93
62 2009-01-09 7.72
2242 2009-01-21 19.77
2242 2009-01-22 20.27
2242 2009-02-02 20.28
2242 2009-02-06 17.51
601988 2009-02-17 2.64
601988 2009-02-16 2.57
601988 2009-02-17 2.64
601988 2009-02-27 3.47
Return a new cursor in which duplicate members have been</td></tr>
</table>

				removed.
22	=directory@p("D://*▯txt")			The txt file is the same as above
23	for A22			
24		=file(A23).cursor@t()	=B22=B22|▯24	
25	=B22.merge@xd(STOCKID)	=A25.fetch()		 A new cursor created by removing from the first cursor the members of the other cursors

STOCKID	DATE	CLOSING
62	2009-01-05	8.91
62	2009-01-06	8.31
62	2009-01-07	7.6
62	2009-01-08	7.93
62	2009-01-09	7.72

Related concept:

A.merge()

mid()

Description:

Return the substring of a string

Syntax:

mid(*s*,*start*{,*len*}**)**

Remark:

Return the substring of *s*, from the specified position *start*, the length of which is *len*.

Parameters:

s	Source string from which to get the substring
start	The starting position of the substring
len	The length of substring. By default, the length will be counted from the starting character to the end of the source string

Return value:

String

Example:

–	mid("abcde",1)	abcde
–	mid("abcde",1,2)	ab
–	mid("abcde",3)	cde

Related concepts:

left()

right()

millisecond()

Description:

Get the millisecond from a specified datetime

Syntax:

millisecond(*datetimeExp***)**

Remark:

Get the millisecond from the datetime *datetimeExp*.

Parameter:

datetimeExp	Expression whose result is a date or a datetime of standard format

Return value:

An integer

Example:

– **millisecond(datetime("1980-02-27 12:00:02:123 ","yyyy-MM-dd hh:mm:ss:SSS"))**	**123**
– **millisecond(now())**	Milliseconds of the current time

Related concepts:

year()

month()

day()

hour()

minute()

second()

min()

A.min()

Description:

Compute the minimum value of all the non-null members in a sequence

Syntax:

A.**min()** Equivalent to **min**($x_1,\ldots,x_n$)

Remark:

Compute the minimum value of all the non-null members in sequence *A*. Please note that this function doesn’t apply to a sequence whose members are of different data types

Parameter:

A	A sequence

Return value:

The minimum value of all members in sequence *A*

Example:

	A	
1	=[8,-6,1,3,5].min()	-6
2	=["c","b","e","A"].min()	"A"
3	=["a",1].min()	An error message will be displayed because members are of different data types
4	=["a",null,"b"].min()	"a"
5	=min("c","b","e","A")	"A"

Related concepts:

A.sum()

A.avg()

A.count()

A.min(x)

A.max()

A.variance()

A.min(*x*)

Description:

Compute *x* with each member of the sequence and then compute the minimum value of the members of the new sequence

Syntax:

A.min(*x*) Equivalent to *A*.(*x*).**min**()

Remark:

Compute *x* with each member of sequence *A* by loop and return the minimum value of members of the resulting sequence

Parameters:

A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names

Return value:

The minimum value of all members after computation has been performed on sequence *A*

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.min(SALARY)	Compute the lowest non-null value of SALARY
3	=A1.(SALARY+100).min()	Add 100 to the salary of each employee and then compute the lowest salary

Related concept:

A.min()

minif()

A.minif()

Description:

Locate all the positions of a member in a sequence, (if needed, get the intersection sets of positions of different members of different sequences) and get the minimum of the members at these positions in another sequence.

Syntax:

A.**minif(**A_i:x_i,…)

Remark:

Locate all positions of member x_i or those of all members of subsequence x_i in sequence A_i and return the minimum of members at these positions in sequence *A*; if there are more than one pair of A_i:x_i, find the positions of each x_i in its A_i and get intersection of these sets of positions, then return the minimum of members at these intersection positions in sequence *A*.

Parameters:

A_i	A sequence
x_i	Members in A_i or a sequence composed of members of A_i
A	The target sequence

Return value:

The minimum value of the members in those result positions of *A*

Example:

	A	B	C	D	
1	Class	Name	Subiect	Score	
2	class one	Aaron	PE	80	
3	class one	Bill	PE	89	
4	class one	Chris	Math	98	
5	class two	Jack	PE	78	
6	class two	Chris	PE	90	
7	class two	Jack	Math	93	
8	class two	Aaron	Math	85	
9	class one	Bill	Math	89	
10	=[D2:D9].minif([C2:C9]:"PE")				78, search with a single condition
11	=[D2:D9].minif([C2:C9]:"PE",[A2:A9]:"class one")				80, search with multiple conditions

Related concepts:

A.countif(A_i:x_i,…)

A.avgif(A_i:x_i,…)

A.sumif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

minp()

A.minp()

Description:

Get the sequence member that makes the minimum value of the expression

Syntax:

A.minp(*x*)

Remark:

Compute the expression *x* against each member of the sequence *A* and return the member which makes the value of the expression *x* minimum

Options:

@1	Return the first member that fulfills the condition.
@a	Return all the members that fulfill the condition. Use **@1** when it is omitted
@z	Search the members from back to front

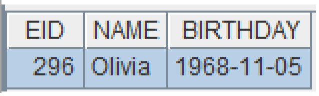
Parameters:

A	A sequence
x	The expression to be computed

Return value:

The member which makes the value of the expression *x* minimum

Example:

	A	
1	[2,5,4,3,2,1,4,1,3]	
2	=A1.minp(~*~)	1, **@1** is the default
3	=A1.minp@a(~*~)	[1,1]
4	=A1.minp@z(~*~)	1
5	=A1.minp@az(~*~)	[1,1]
6	=demo.query("select EID,NAME,BIRTHDAY from EMPLOYEE")	
7	=A6.minp(BIRTHDAY)	

EID	NAME	BIRTHDAY
296	Olivia	1968-11-05

Related concepts:

A.pmin()

A.maxp()

minute()

Description:

Get the minute from a datetime

Syntax:

minute(*datetimeExp***)**

Remark:

Get the minute from the specified datetime *datetimeExp*.

Parameters:

datetimeExp	Expression whose result is a date or standard date format

Return value:

Integer

Example:

–	minute(datetime("19800227","yyyyMMdd"))	0
–	minute("1972-11-08 10:20:30")	20
–	minute(datetime("2006-01-15 13:20:30"))	20

Related concepts:

year()

month()

day()

hour()

second()

millisecond()

modify()

r. modify(x_i:F_i,…)

Description:

Modify the field values of a record.

Syntax:

r.**modify**(x_i:F_i,…)

Remark:

Modify the field values in record *r*, the status of *r* will be changed to **1**(modify) after being modified.

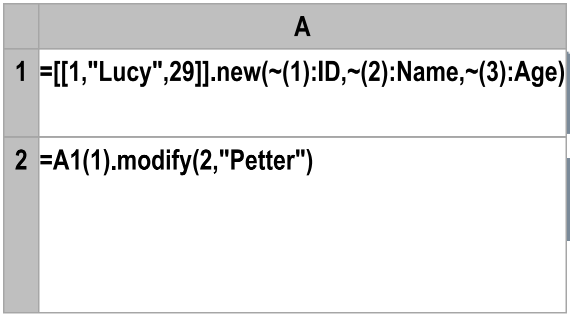
Parameters:

r	The record to be modified
x_i	Modification expression
F_i	Name of the field to be modified, the i^{th} field in *r* will be modified if F_i is omitted.

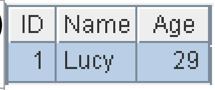
Return value:

The modified record *r*

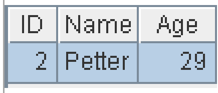
Example:



	A
1	=[[1,"Lucy",29]].new(~(1):ID,~(2):Name,~(3):Age)
2	=A1(1).modify(2,"Petter")

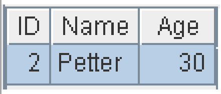
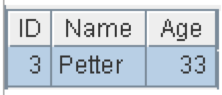


ID	Name	Age
1	Lucy	29



ID	Name	Age
2	Petter	29

modify the first and the second field

3	=A2.modify(30:Age)	 modify field **Age**
4	=A3.modify(3,33:Age)	 modify the first field and **Age** field

ID	Name	Age
2	Petter	30

ID	Name	Age
3	Petter	33

Related concepts:

T.modify()

T.insert()

T.delete()

T. modify()

Description:

Modify field values in a table sequence.

Syntax:

T.**modify**(k,x_i:F_i,…) Modify the k^{th} record, which is equal to *T*(*k*).**modify**(x_i:F_i,…).

T.**modify**(k:A,x_i:F_i,…) Modify the records form the k^{th} record to the number k+|A|-1 record

Remark:

Modify one or more records at the specified position(s). The status of the records which have been modified will be changed to **1**.

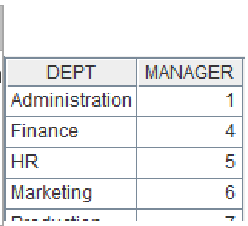
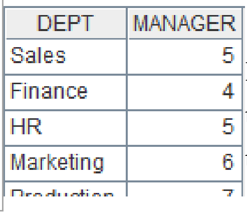
Parameters:

k	The position at which the record will be modified. If k exceeds the limit, then append a new record in the end.
x_i	The new value of F_i.
F_i	Name of field of the record which will be modified. If F_i is omitted, then modify the i^{th} field of *T*.
T	A table sequence
A	A sequence or an integer; If *A* is an integer, then it is equal to to(*A*)

Return value:

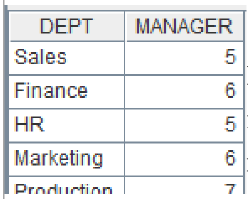
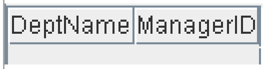
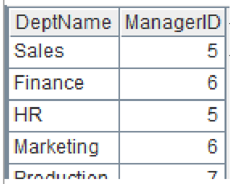
The modified table sequence *T*

Example:

	A	
1	=demo.query("select * from DEPARTMENT")	
2	=A1.modify(1,"Sales",5)	 Modify the first and the second field of the first record.

DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6
Production	7

DEPT	MANAGER
Sales	5
Finance	4
HR	5
Marketing	6
Production	7

3	=A1.modify(2,6:MANAGER)	 DEPT MANAGER Sales 5 Finance 6 HR 5 Marketing 6 Production 7	Modify the **MANAGER** field of the second record.
4	=create(DeptName,ManagerID)	 DeptName ManagerID	
5	=A4.modify(1:A1,DEPT:DeptName,MANAGER:ManagerID)	 DeptName ManagerID Sales 5 Finance 6 HR 5 Marketing 6 Production 7	Append a new record in the end because **1** exceeds the limit

Related concepts:

r.modify()

T.insert()

T.delete()

A.modify()

A.modify()

Description:

Assign values to one or more members of a sequence according to the specified position(s).

Syntax:

A.**modify**(*k*,*x*)	Assign *x* to the k^{th} member
A.**modify**(*k*,*X*)	Assign members of *X* to members of *A* from the k^{th} position to the number *k*+|*X*|-1 position in proper order

Remark:

Assign *x* to the k^{th} member of *A* or assign the members of *X* to the members of *A* from the k^{th} position to the number *k*+|*X*|-1 position in proper order.

Parameters:

A	A sequence
k	A member position; If *k* exceeds the limit, then append the member in the end
x	A member value
X	A sequence composed of member values

Option:

@n	*A* will not be changed, and only return a new sequence

Return value:

A sequence

Example:

	A
1	=["a","c","d","e","f"]

2	=A1.modify@n(2,"g")	[a,g,d,e,f], option **@n** does not change **A1**.
3	=A1.modify@n(2,[2,4,5])	[a,2,4,5,f]
4	=A1.modify(6,[2,4,5])	[a,c,d,e,f,2,4,5], The specified position exceeds the limit, then append the members in the end

Related concepts:

A.delete()

A.insert()

T.modify()

mongodb()

Description:

Establish a connection to MongoDB.

Syntax:

mongodb(*con*)

Remark:

Connect to MongoDB. The format of the connecting string is mongo://*ip*:*port*/*db*?*arg*=*v*&… So far the function supports two parameters: user and password.

Parameters:

con Database connecting string

Return value:

The connection to the database

Example:

	A	
1	=mongodb("mongo://127.0.0.1:27017/myTest?user=root&password=sa")	Connect to MongoDB **myTest**

Related concepts:

mdb.close()

mdb.find()

mdb.count()

mdb.distinct()

mdb.aggregate()

month()

Description:

Get the month from a a specified date/datetime

Syntax:

month(*dateExp*)

Remark:

Get the month from the date *dateExp*.

Parameters:

dateExp Expression whose result is a date or standard date/datetime format

Return value:

An integer

Example:

–	**month(datetime("19800227","yyyyMMdd"))**	**2**
–	**month("1972-11-08 10:20:30")**	**11**
–	**month(datetime("2006-01-15 13:20:30"))**	**1**

Related concepts:

year()

day()

hour()

minute()

second()

millisecond()

movefile()

movefile(*fn*,*path*)

Description:

Move, delete, or rename a file

Syntax:

movefile(*fn*,*path***)**

Remark:

Move the file *fn* to a file specified by *path*. If omitting *path*, then this file will be deleted; If only file name is given in the *path*, then the file will be renamed.

Parameters:

fn File object

path The path to which the file is moved (with file name) or the file name

Options:

@y Force the operation if the target file exists. Without it the operation will fail.

@c Copy the file. If the name of the target fileis the same as that of the specified file to which it will be moved, the copying will fail.

Return value:

Boolean value

Example:

	A

1	**=movefile(file("E://test.property"),"D://testfile.property")**	Move the file to **testfile.property** file under root directory on driver D
2	**=movefile(file("D://testfile.property"),"file.property")**	**D://testfile.property** is renamed **file.property**
3	**=movefile(file("D://file.property"))**	Delete **file.property** file
4	**=movefile@y(file("E://test1.property"),"D://testfile1.property")**	The file **testfile1.property** already exists. Force the move and override the original file
5	**=movefile@c(file("D://testfile1.property"),"file.property")**	Copy the file, instead of renaming it
6	**=movefile@cy(file("E://test2.property"),"D:// file.property")**	**file.property** already exists. Force the copy and override the original file

Related concepts:

f. exists()

f. date()

f. size()

movefile(*fn*,*path*,*z*)

Description:

Copy, move, delete or rename a file across zones

Syntax:

movefile(*fn*,*path*,*z***)**

Remark:

In the node machine, move or copy a file to the specified path *path* on zone *z* across zones. The omission of *path* and *z* indicates that the file will be removed. If only the file name exists in the *path*, then that file will be renamed.

Parameter:

fn File object on a zone

path The path to which the file is moved (with file name) or the file name, versus the path of zone configuration.

z Zone name

Options:

@y Force the operation if the target file already exists in the specified zone. Without it the operation will fail

@c Copy the file. If the name of the target file is the same as that of the specified file to which it will be moved, the copying will fail.

Return value:

Boolean value

Example:

On the node machine “**192.168.0.99:9282**”, cellset file *move.dfx* is shown below. Set the cellset parameter *arg1*:

	A	

1	=movefile(file("test.txt","3"),"test.txt","1")	Move the file across zones. If there is already a file with the same name in the target zone **1**, then the operation will fail
2	=movefile(file("test1.txt","3"),"testfile.txt","1")	Move the file and rename it
3	=movefile@c(file("test2.txt","3"),"testfile2.txt","3")	Copy th file
4	=movefile(file("\\txt\\test.txt","3"),"\\txt\\test.txt","1")	In the target zone **1**, if there is no file folders, then a file folder will be created automatically
5	=movefile@y(file("testfile2.txt","3"),"test.txt","1")	In the target zone **1**, if there is already a file with the same name, then force the move and override the original file.
6	=movefile(file("\\txt\\test1.txt","3"),"\\txt\\test2.txt","3")	For **\\txt\\test1.txt**, the file is renamed **\\txt\\test2.txt**
7	=movefile(file("\\txt\\test2.txt","3"))	Remove the file **\\txt\\test2.txt** on the target zone

	A	
1	=callx("move.dfx",1;"192.168.0.99:9282")	Call cellset file

Related concepts:

movefile(*fn*,*path*)

n.f(*x*)

Description:

Compute a loop function using an integer as the loop variable.

Remark:

Compute the loop function *f* using *n* as the loop variable.

n.(*x*) equals to **to**(*n*).(*x*)

n.*f*(*x*) equals to **to**(*n*),*f*(*x*)

Parameters:

n	An integer
x	An expression
f	The function name

Example:

- *n*.(*x*)

	A	
1	=3.(~*2)	[2,4,6]

- *n*.*f*(*x*)

	A	
1	=3.sum(~*2)	12

ncursor()

f.ncursor()

Description:

Use the index file to retrieve file content specified by an interval of field values

Syntax:

f.**ncursor**(*fi*,*a*:*b*;F_i,…)

Remark:

Use the index file *fi* to retrieve records which have the index field values in the range of [*a*,*b*] from *f* and return them as a cursor. For the multi-field index, *a* and *b* are sequences and the number of their respective members can be less than that of the index fields. Their members are matched to the records from the front to the back. In this case *a:b* cannot be omitted.

Parameters:

f	Binary file object
fi	Index file
a	Value of the index field. For the multi-field index, *a* is a sequence
b	Value of the index field. For the multi-field index, *b* is a sequence
F_i	Fields that are picked out

Options:

@l	Exclude *a*
@r	Exclude *b*

Return value:

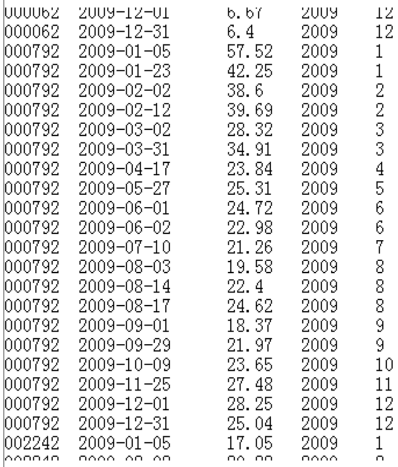
Cursor

Example:

	A	
1	=file("D:\\EMPLOYEE1.txt").icursor(file("D:\\e"),5:500;EID,NAME,DEPT)	Use the index file **e** to retrieve the records in which values of **EID** in the file **EMPLOYEE1.txt** are between 5 and 500. Then, select out fields **EID**, **NAME**, and **DEPT**.
2	=file("D:\\EMPLOYEE1.txt").icursor@l(file("D:\\e"),5:500;EID,NAME,DEPT)	Exclude records whose **EID** is 5
3	=file("D:\\EMPLOYEE1.txt").icursor@r(file("D:\\e"),5:500;EID,NAME,DEPT)	Exclude records whose **EID** is 500
4	=file("D:\\EMPLOYEE.txt").icursor(file("D:\\es"),1:496;EID,NAME,DEPT,SALARY)	Multi-field index. Search for the records whose **EID** is in the range of ［**1,496**］. In the index file **es,** the indexes are created in the order of **EID** field and then **SALARY** field.
5	=file("D:\\testfile.txt").icursor(file("testfile"),[2009,1]:[2	Binary file **testfile.txt** is shown

```
009,2];STOCKID,DATE,CLOSING,YEAR,MONTH).fetch(
)
```

below:



```
000062  2009-12-01     6.67   2009   12
000062  2009-12-31     6.4    2009   12
000792  2009-01-05     57.52  2009   1
000792  2009-01-23     42.25  2009   1
000792  2009-02-02     38.6   2009   2
000792  2009-02-12     39.69  2009   2
000792  2009-03-02     28.32  2009   3
000792  2009-03-31     34.91  2009   3
000792  2009-04-17     23.84  2009   4
000792  2009-05-27     25.31  2009   5
000792  2009-06-01     24.72  2009   6
000792  2009-06-02     22.98  2009   6
000792  2009-07-10     21.26  2009   7
000792  2009-08-03     19.58  2009   8
000792  2009-08-14     22.4   2009   8
000792  2009-08-17     24.62  2009   8
000792  2009-09-01     18.37  2009   9
000792  2009-09-29     21.97  2009   9
000792  2009-10-09     23.65  2009   10
000792  2009-11-25     27.48  2009   11
000792  2009-12-01     28.25  2009   12
000792  2009-12-31     25.04  2009   12
002242  2009-01-05     17.05  2009   1
```

The index file **testfile** comprises the index fields of **YEAR, MONTH,** and **STOCKID.** In this order of fields, the index is created.

Retrieve records whose [**YEAR,MONTH]** is not less than **[2009,1]** and not greater than **[2009,2]**

Related concept:

f. **index()**

new()

A.new(*x*$_i$:*F*$_i$,…)

Description:

Generate a new table sequence, field values of which is computed against a sequence.

Syntax:

A.**new(**x_i:F_i,…)

Remark:

Generate a new table sequence with the same length as sequence *A*, the field names and values of which are F_i and x_i.

Parameters:

F_i	Field name. If omitted, then use the identifier parsed in the x_i
x_i	Expression whose result is the field value. If omitted, the field values will be null. If omitting x_i, then you are not allowed to omit F_i
A	Sequence, according to which the expression x_i will be computed

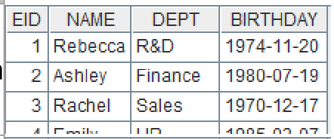
Return value:

The new table sequence

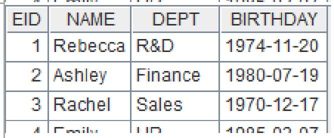
Example:

- Generate from an individual table sequence

	A
1	=demo.query("select EID,NAME,DEPT,BIRTHDAY from EMPLOYEE")
2	=A1.new(EID:EmployeeID,NAME,DEPT)
3	=A1.new(NAME,age(BIRTHDAY):AGE)

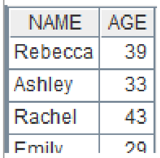


EID	NAME	DEPT	BIRTHDAY
1	Rebecca	R&D	1974-11-20
2	Ashley	Finance	1980-07-19
3	Rachel	Sales	1970-12-17
4	Emily	HR	1985-03-07



EID	NAME	DEPT	BIRTHDAY
1	Rebecca	R&D	1974-11-20
2	Ashley	Finance	1980-07-19
3	Rachel	Sales	1970-12-17
4	Emily	HR	1985-03-07

Generate a new table sequence directly. The field names which are the same as that of **A1** may be omitted.

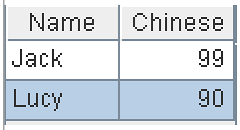


NAME	AGE
Rebecca	39
Ashley	33
Rachel	43
Emily	29

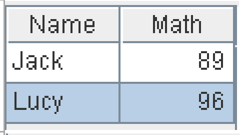
Compute field values when generating the new table sequence

- Generate from multiple table sequences of the same order

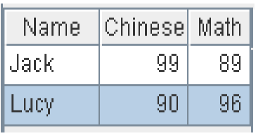
	A
1	=create(Name,Chinese).record(["Jack",99,"Lucy",90])
2	=create(Name,Math).record(["Jack",89,"Lucy",96])
3	=A1.new(Name:Name,Chinese:Chinese,A2(#).Math:Math)



Name	Chinese
Jack	99
Lucy	90



Name	Math
Jack	89
Lucy	96



Name	Chinese	Math
Jack	99	89
Lucy	90	96

Use **A2(#)** to get the record from **A2** in the same position as A1

Related concepts:

cs.new()

cs.new()

Description:

Generate a new cursor, field values of which is computed against a cursor.

Syntax:

cs.**new**(x_i:F_i,…)

Remarks:

Generate a new cursor with the same length as sequence *cs*, the field names and values of which are F_i and x_i.

Parameters:

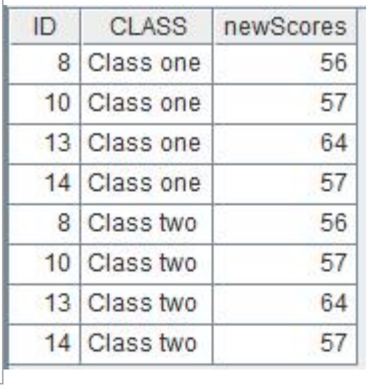
cs	Cursor
x_i	New field value of F_i
F_i	Field name of *cs*

Return value:

Cursor

Example:

	A	
1	=connect("demo").cursor("select * from SCORES where SCORES<60")	
2	=A1.new(STUDENTID:ID, CLASS, SCORE+5:newScores)	For cursor A1, run new(x_i:F_i,…) Generate a new cursor composed of ID, CLASS, and newScores For existing Score, perform the formula computation.
3	=A2.fetch()	



ID	CLASS	newScores
8	Class one	56
10	Class one	57
13	Class one	64
14	Class one	57
8	Class two	56
10	Class two	57
13	Class two	64
14	Class two	57

Related concepts:

A.new(x_i:F_i,…)

next{}

Description:

To skip the current loop and continue the next loop.

Syntax:

next {*a*}

Remark:

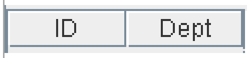
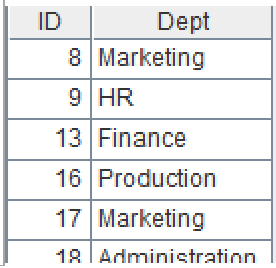
next is used to skip the remaining code block in the current loop and execute the next loop as long as the cycling does not finish.

Parameters:

a	The main cell of the loop code block. If the main cell *a* is omitted, then it indicates the current loop.

Example:

	A	B	C	D

1	=[]				[1,1,3,3]
2	for 5				In this example, the next A2 and the break A2 are to control loop of the A2 level. The computation of A1 sequence is [1, 1, 3, 3]
3		for 2			
4			if A2==2	next A2	
5			if A2==4	break A2	
6			>A1=A1|[A2]		
7	=demo.query("select * from EMPLOYEE")				
8	=create(ID,Dept)				
9	for A7				Extract records whose DEPT values are not "Sales" from the EMPLOYEE table
10		if A9.DEPT=="Sales"			
11			Next		
12		>A8.insert(0,A9.EID:ID,A9.DEPT:Dept)			A8 returns 

ID	Dept

ID	Dept
8	Marketing
9	HR
13	Finance
16	Production
17	Marketing
18	Administration

Note:

1. By comparison, the **break** is to jump out of the cycling directly, while **next** is to ignore the remaining code block in the current loop, and proceed to the next loop as long as the cycling does not finish.
2. If *a* is omitted, then the current loop will be under the control. To control the loop at higher level, *a* is required.

now()

Description:

Get the current system date time

Syntax:

now()

Remark:

Get the current system datetime measured down to the millisecond

Options:

@d	Return the date part only, date type
@t	Return the time part only, time type
@m	Measure to minute
@s	Measure to second

Return value:

Date time

Example:

- **now()** The current system date time, for example: **2010-07-15 16:10:40**
- **now@d()** Current system date, for example:**2010-07-15**
- **now@t()** The current system time, for example:**16: 10: 40**
- **now@m()** The current system time, for example:**2013-12-09 17:05:00:0**
- **now@s()** Current system date, for example:**2013-12-09 17:05:33:0**

null

Description:

Null value

Syntax:

null The value of a null cell is also a **null**.

Remark:

It can be used directly in the constant cell or expression.

Example:

	A	B	
1	=null		Assign a null value to **A1**
2	=A1==null		Judge if **A1** is null
3	if A2==true	>a=4	If null, assign **4** to **a**
4	else	>a=3	Otherwise, assign **3** to **a**

Related concepts:

true

false

number()

Description:

Convert a string to a real number.

Syntax:

number(*stringExp***)**

Remark:

The result of *stringExp* must be a numeric string.

Parameters:

stringExp A string expression, the result of which is a numeric string.

Return value:

32-bit integer, 64-bit integer, or 64-bit floating-point number.

Example:

- **number("123")** **123**

- **number("123f")** **123.0**
- **number("123.45")** **123.45**
- **number("123.456d")** **123.456**

Related concepts:

float()

int()

long()

decimal()

string()

output()

Description:

Print out data to console

Syntax:

output(*x***,...)**

Remark:

Capable to export one or several variable data separated with comma, and the printout data from console is separated with Tab.

Parameter:

x,…	Export parameters separated with comma

Options:

@t	Print out the print data along with current time and place the current time before the print data

Example:

	A	
1	=to(4)	
2	=output(A1)	Output "**[1,2,3,4]**"
3	=output@t(A1)	Output " **2013-05-02 11:51:59 [1,2,3,4]**"

p()

A.p()

Description:

Get sequence numbers of the members at the specified positions.

Syntax:

*A***.p(***i***)**	-*n***<=***i***<=***n* and *i* is not equal to **0**; For **1<=***i***<=***n*, it indicates to get the sequence number of the ith member; For -*n*<=*i*<=-1, it indicates to get the sequence number of the ith member from the last.
*A***.p(***P***)**	*P* is the *n* integer sequence whose length is *m*, the member values of which should be

larger than -*n* and less than *n*, but not equal to **0.**

Remark:

A is an *n* sequence. Get sequence numbers of the members at the specified positions *i* or *P*. This is generally used to get the sequence numbers of the members reversely.

Parameters:

A	a sequence object whose length is *n*
i	an integer
P	the *n* integer sequence whose length is *m*

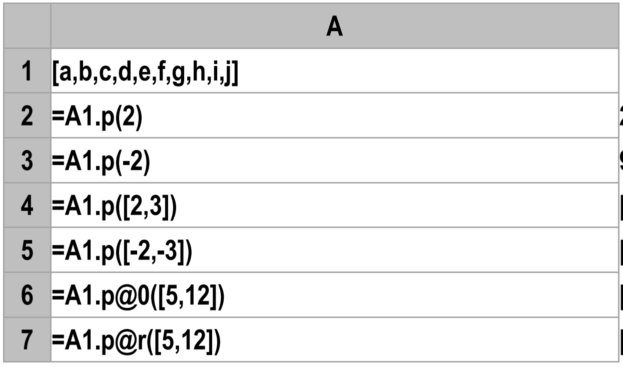
Options:

@r	Turn back the position exceeding the boundary of *A*, that is, to set *i*=**if**(*i*%*n*==0,*n*,*i*%*n*), where *n* is the length of *A*.
@0	The position exceeding the boundary of *A* will be ignored.

Return value:

An integer or an integer sequence of the sequence number of members at the specified positions in the sequence *A*

Example:



	A	
1	[a,b,c,d,e,f,g,h,i,j]	
2	=A1.p(2)	2
3	=A1.p(-2)	9
4	=A1.p([2,3])	[2,3]
5	=A1.p([-2,-3])	[9,8]
6	=A1.p@0([5,12])	[5]
7	=A1.p@r([5,12])	[5,2]

Related concepts:

A.m()

pad()

Description:

Pad another character string ahead of the character string until reaching the specified length.

Syntax:

pad(*s*,*c*,*l*)

Remark:

Pad the character string c ahead of the character string s until the total length of the first character string is *l*.

Parameters:

s	Character string expression
c	Character string expression
l	Character string whose result is the numeric value

Options:

@r	Pad another character string on the right of the character string

Return value:

Character string

Example:

	A	
1	=pad("Soth","Miss",10)	The return value is" **MissMiSoth "**
2	=pad@r("Soth","er",8)	The return value is" **Sotherer "**

paste()

P. paste()

Description:

Paste the field values of a specified RSeq into RSeq P field-by-field and record-by-record in proper order.

Syntax:

P.**paste**(*A*)

Remark:

Paste the field values of RSeq *A* into RSeq *p* field by field and record by record in proper order.

The number of the records to be pasted is determined by **min(***A***.len(),** *P***.len())**, and the modification will be stopped once the record number exceeds the boundary.

Parameters:

P	the record sequence to be modified
A	the record sequence or sequence which will be pasted into the fields of *p*.

Options:

@n	This option indicates that the modification will be executed by field names.

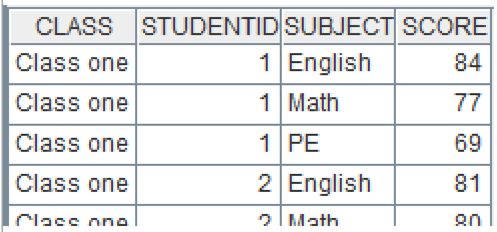
Return value:

The record sequence *p* after being pasted

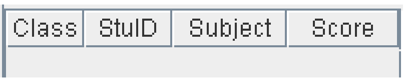
Example:

- Being modified in order of the fields

	A
1	=demo.query("select * from SCORES")
2	=create(Class,StuID,Subject,Score)

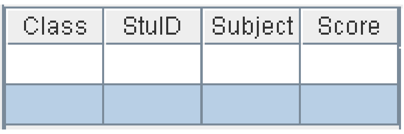


CLASS	STUDENTID	SUBJECT	SCORE
Class one	1	English	84
Class one	1	Math	77
Class one	1	PE	69
Class one	2	English	81
Class one	2	Math	80



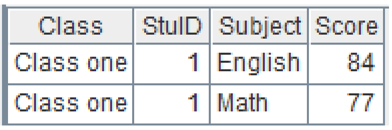
Class	StuID	Subject	Score

3	>A2.insert(0:2)



Class	StuID	Subject	Score

4	=A2.paste(A1)

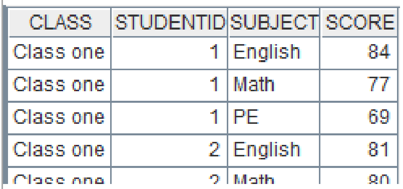


Class	StuID	Subject	Score
Class one	1	English	84
Class one	1	Math	77

Modify two records only, as there are only two records in **A2**.

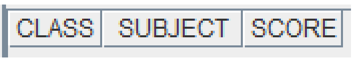
- Being modified in order of the field names

	A
1	=demo.query("select * from SCORES")



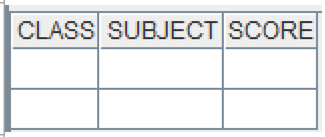
CLASS	STUDENTID	SUBJECT	SCORE
Class one	1	English	84
Class one	1	Math	77
Class one	1	PE	69
Class one	2	English	81
Class one	2	Math	80

2	=create(CLASS,SUBJECT,SCORE)



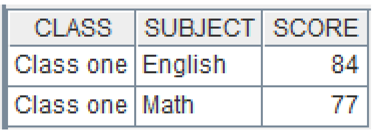
CLASS	SUBJECT	SCORE

3	>A2.insert(0:2)



CLASS	SUBJECT	SCORE

4	=A2.paste@n(A1)



CLASS	SUBJECT	SCORE
Class one	English	84
Class one	Math	77

Modify the three fields **CLASS**, **SUBJECT** and **SCORE.**

Related concepts:

T.record()

r.paste()

Description:

Modify the contents of specified records

Syntax:

r.paste(*r'*)

Remark:

To modify the record *r* with the record *r'* by fields, or

Parameters:

r the record to be modified

r' the record whose field values are used as the new values of *r*

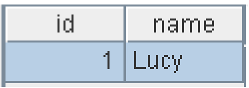
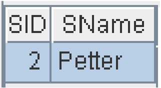
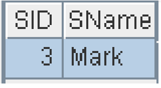
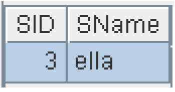
Options:

@n this option indicates that in the syntax of *r*.**paste**(*r'*), *r* will be modified according to the field names of *r'*.

Return value:

The record *r* after being pasted.

Example:

	A	
1	=new(1:id,"Lucy":name)	 id name 1 Lucy
2	=new(2:SID,"Petter":SName)	 SID SName 2 Petter
3	=A2(1).paste([3,"Mark"])	 SID SName 3 Mark Paste the members of **[3,"Mark"]** into **A2(1)** in proper order
4	=A4.paste@n(new("ella":SName)(1))	 SID SName 3 ella Modify **A3** by field names, so only the field **SName** is modified.

pcursor()

Description:

Generate cursor programally

Syntax:

pcursor(*dfx*,…)

Remark:

Pass in the parameter … , collect the results of result when executing the *dfx*, and return them as a cursor, dfx can be the file object. When there are multiple results in *dfx*, firstly merge multiple results of *result*, then collect the result of merging, and return as a cursor. These result of *result* must be of the same structure, or error will be reported.

Parameter:

dfx	cellset file
…	*dfx* parameter

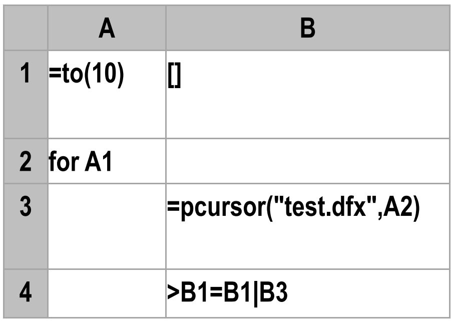
Return value:

Cursor

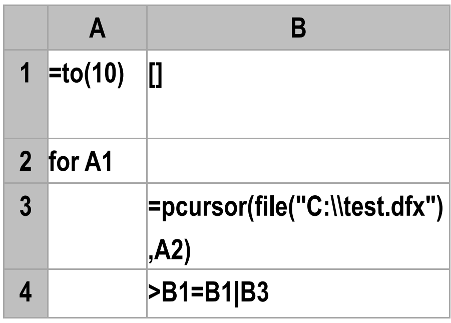
Example:

C: \\test.dfx cellset file has the below contents, in which arg1 is the cellset parameter:

	A	B
1	=connect("demo").query("select * from GYMNASTICSWOMEN where ID=?",arg1)	
2	=A1.derive(avg(VAULT,UNEVENBARS,BALANCEBEAM,FLOOR):Average)	
3	result A2	



	A	B	
1	=to(10)	[]	In **B1**, store the cursor result of accumulating the result for each time
2	for A1		
3		=pcursor("test.dfx",A2)	Cycle and call the cellset file, pass-in the parameter, and return the result as a cursor.
4		>B1=B1|B3	



	A	B	
1	=to(10)	[]	In **B1**, store the cursor result of accumulating the result for each time
2	for A1		
3		=pcursor(file("C:\\test.dfx"),A2)	Call the cellset file by loop. dfx is a file object, input parameters and return the result of code result as the
4		>B1=B1|B3	

pdate()

Description:

Get the first and the last days of the week/month/quarter to which a date belongs

Syntax:

pdate (*dateExp*)

Remark:

Get the first and the last days of the week/month/quarter to which the date *dateExp* belongs

Parameters:

dateExp	Expression whose result is a date or date time

Options:

@w	Get the Sunday of the week to which the specified date belongs
@we	Get the Saturday of the week to which the specified date belongs
@m	Get the beginning day of the month to which the specified date belongs
@me	Get the last day of the month to which the specified date belongs
@q	Get the beginning day of the quarter to which the specified date belongs
@qe	Get the last day of the quarter to which the specified date belongs
	The default is to get the Sunday of the week to which the specified date belongs

Return value:

Date time type

Example:

– **pdate@w(datetime("19800227","yyyyMMdd"))**	**1980-02-24**
– **pdate@we (datetime("19800227","yyyyMMdd"))**	**1980-03-01**

– **pdate@m(datetime("19800227","yyyyMMdd"))**	**1980-02-01**
– **pdate@me(datetime("19800227","yyyyMMdd"))**	**1980-02-29**
– **pdate@q(datetime("19800227","yyyyMMdd"))**	**1980-01-01**
– **pdate@qe(datetime ("19800227","yyyyMMdd"))**	**1980-03-31**

penum()

E.penum()

Description:

Judge to which enum group an object belongs.

Syntax:

E.**penum**(*y*)

Remark:

judge to which group of *E* record sequence the *y* belongs, and return the sequence number of the group satisfying the grouping conditions.

Parameters:

E	a sequence/record sequence
y	*y* is allowed to be omitted

Options:

@r	Return the sequence numbers of all the group satisfying the group conditions, and the default is to return the sequence number of the first group
@n	If no sequence member is found, return the length of *E* plus 1. This option is mutual exclusive to @r.

Function Keyword:

?	represents the value of *y*

Return value:

A sequence number or an integer sequence composed of sequence numbers

Example:

- *E* is the sequence

	A	
1	**=["?<=70&&?<=60","?>=70 && ?<=90","?>=90"]**	Use *y* to replace the **?** in the group conditions when computing
2	**=A1.penum(90)**	**2**; the sequence number of the first group satisfying the conditions will be returned
3	**=A1.penum@r(90)**	**[2,3]**; return the sequence numbers of all the groups
4	**=A1.penum@n(43)**	**4**

Related concepts:

P.enum()

periods()

Description:

Generate a date/time sequence by specified interval.

Syntax:

periods(*s*,*e*,*i*)

Remark:

Generate a new sequence composed of values of date and time in a period from *s* to *e* (start and end points included) at *i* interval.

Parameters:

s	a date time variable
e	a date time variable
i	an integer indicating the interval; its unit is day and its value is 1 by default

Options:

@y	*i* is in years
@q	*i* is in quarters
@m	*i* is in months
@t	*i* is in ten-days
@s	*i* is in seconds
@x	exclusive of end point
@o	Not be adjusted. By default, the result will be adjusted to the original start point of the time unit and adjustment must be done in case of @t.

Return value:

The new sequence composed of date times

Example:

	A	
1	2000-08-10 12:00:00	
2	=periods@y(A1,now(),1)	Set year as the interval unit
3	=periods@yo(A1,now(),1)	Not adjusted, it is adjusted to the original point of the time unit by default, and must be adjusted when **@t**.
4	=periods@q(A1,now(),1)	Set quarter as the interval unit
5	=periods@m(A1,now(),1)	Set month as the interval unit
6	=periods@s(A1,now(),7)	Set second as the interval unit
7	=now()	
8	=pdate@m(A7)	The start date of the current month
9	=pdate@me(A7)	The end date of the current month
10	=after(A8,6-day@w(A8))	Get the first Friday
11	=periods@x(A10,A9,7)	Get the Friday sequence
12	=A11(2)	Get the second Friday
13	=A11.m(-1)	Get the last Friday

14	=A11.len()	How many Fridays

Related concepts:

pfind()

A.pfind()

Description:

To find the sequence number of a record by its primary key.

Syntax:

A.**pfind**(*v*)

Remark:

If the sequence *A* is composed of non-record members, *A*.**pfind()** is used to get the sequence number of the first member equal to *v* from *A*.

If *A* is a record sequence, *A*.**pfind()** is used to get the sequence number of the member from *A* whose primary key values are equal to *v*.

Return 0 if not found.

Parameters:

A	a sequence
v	a member or a primary key value; a sequence shall be used if there are multiple primary key values.

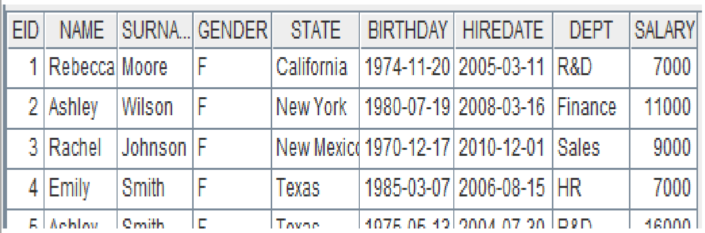
Options:

@b	Enable the dichotomizing search and *A* must be ordered by the primary key; otherwise, the result will be wrong.
@s	Return the opposite number of insertable position number when the sequence cannot be found

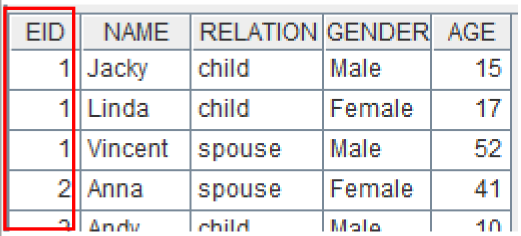
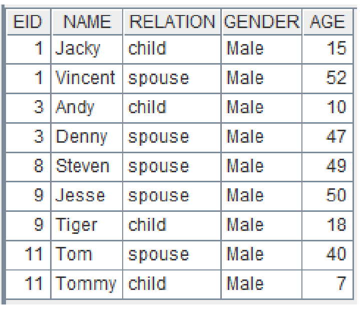
Return value:

The sequence number of the member found

Example:

	A	
1	=demo.query("select * from EMPLOYEE ")	
2	>A1.primary(NAME,DEPT)	
3	=A1.pfind(["Alexis","Sales"])	7; *v* is a sequence because there are multiple primary fields

EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000

<table>
<tr><td>4</td><td>=demo.query("select * from FAMILY")</td><td>
<table>
<tr><th>EID</th><th>NAME</th><th>RELATION</th><th>GENDER</th><th>AGE</th></tr>
<tr><td>1</td><td>Jacky</td><td>child</td><td>Male</td><td>15</td></tr>
<tr><td>1</td><td>Linda</td><td>child</td><td>Female</td><td>17</td></tr>
<tr><td>1</td><td>Vincent</td><td>spouse</td><td>Male</td><td>52</td></tr>
<tr><td>2</td><td>Anna</td><td>spouse</td><td>Female</td><td>41</td></tr>
<tr><td>3</td><td>Andy</td><td>child</td><td>Male</td><td>10</td></tr>
</table>
This sequence table is ordered by the <b>EID</b> field</td></tr>
<tr><td>5</td><td>&gt;A4.primary(EID)</td><td></td></tr>
<tr><td>6</td><td>=A4.pfind@b(3)</td><td><b>5</b>; <b>@b</b> is used to enable the dichotomizing search in order to fasten the computation</td></tr>
<tr><td>7</td><td>=demo.query("select * from FAMILY where GENDER='Male'")</td><td>
<table>
<tr><th>EID</th><th>NAME</th><th>RELATION</th><th>GENDER</th><th>AGE</th></tr>
<tr><td>1</td><td>Jacky</td><td>child</td><td>Male</td><td>15</td></tr>
<tr><td>1</td><td>Vincent</td><td>spouse</td><td>Male</td><td>52</td></tr>
<tr><td>3</td><td>Andy</td><td>child</td><td>Male</td><td>10</td></tr>
<tr><td>3</td><td>Denny</td><td>spouse</td><td>Male</td><td>47</td></tr>
<tr><td>8</td><td>Steven</td><td>spouse</td><td>Male</td><td>49</td></tr>
<tr><td>9</td><td>Jesse</td><td>spouse</td><td>Male</td><td>50</td></tr>
<tr><td>9</td><td>Tiger</td><td>child</td><td>Male</td><td>18</td></tr>
<tr><td>11</td><td>Tom</td><td>spouse</td><td>Male</td><td>40</td></tr>
<tr><td>11</td><td>Tommy</td><td>child</td><td>Male</td><td>7</td></tr>
</table></td></tr>
<tr><td>8</td><td>&gt;A7.primary(EID)</td><td></td></tr>
<tr><td>9</td><td>=A7.pfind@s(5)</td><td><b>-5</b>,Return the opposite number of insertable position number when the record of E<b>ID</b>=<b>5</b> cannot be found</td></tr>
<tr><td>10</td><td>=[1,3,5,7]</td><td></td></tr>
<tr><td>11</td><td>=A10.pfind(3)</td><td><b>2</b>; because <b>A10</b> is neither a record sequence nor a table sequence, the search operation is conducted according to members' values</td></tr>
</table>

Related concepts:

A.find()

pi()

Description:

Compute the circumference ratio and its multiple

Syntax:

pi(*numberExp*)

Remark:

Compute the circumference ratio and its multiple. The value of *numberExp* is 1 by default.

Parameter:

numberExp	Multiple. If omitting this parameter, then return the circumference ratio

Return value:

Circumference ratio and its multiple

Example:

- pi() 3.141592653589793
- pi(2) 6.283185307179586

pjoin()

pjoin()

Description:

Join several sequences together by its sequence number.

Syntax:

pjoin(x_i:F_i,…)

Remark:

Join multiple sequences of x_i by the sequence numbers of its members one after another, so as to generate a new table sequence whose fields are F_i,….The F_i is the reference fields, referencing the record of the original record sequence of x_i.

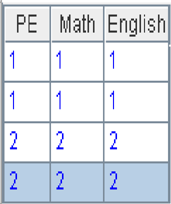
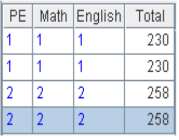
Parameters:

F_i	Field name of the resulting table sequence
x_i	Sequence or record sequence you want to join

Return value:

All fields are the new table sequence of the reference fields

Example:

	A		
1	=demo.query("select top 3 STUDENTID, SCORE from SCORES where SUBJECT='PE' order by STUDENTID ")		
2	=demo.query("select top 5 STUDENTID,SCORE from SCORES where SUBJECT='Math' order by STUDENTID")		
3	=demo.query("select top 4 STUDENTID, SCORE from SCORES where SUBJECT='English' order by STUDENTID")		
4	=pjoin(A1:PE,A2:Math,A3:English)	 PE Math English 1 1 1 1 1 1 2 2 2 2 2 2	Normal join, and the unmatched items will be discarded
5	=A4.derive(A4.PE.SCORE+A4.Math.SCORE+A4.English.SCORE:Total)	 PE Math English Total 1 1 1 230 1 1 1 230 2 2 2 258 2 2 2 258	Total scores of each student

Note:

In this case, it is the equal join on the sequence number of the sequence member, instead of the **#0** field of the record.

Related concepts:

join()

xjoin()

CS.pjoin()

Description:

Join the cursor sequence transversely, and return as a cursor.

Syntax:

CS.**pjoin**()

Remark:

The *CS* is a sequence of cursors. Join the members transversely, which equals to add column to cursor, return the cursor, and take the length of the shortest cursor

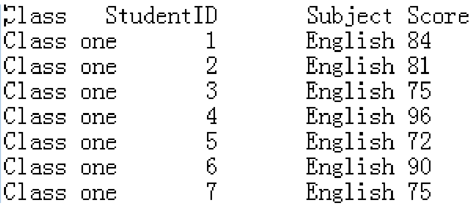
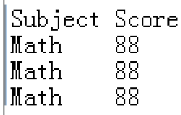
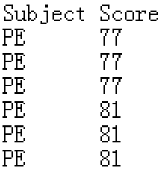
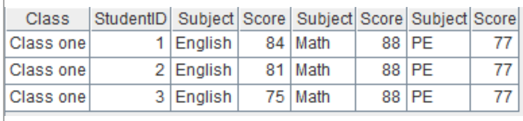
Parameters:

CS	A sequence consisting of cursors

Return value:

Cursor

Example:

	A	B	
1	=directory@p("D://txt//")	=[]	Contents of the 3 files in the txt folder is as follows:  Class StudentID Subject Score Class one 1 English 84 Class one 2 English 81 Class one 3 English 75 Class one 4 English 96 Class one 5 English 72 Class one 6 English 90 Class one 7 English 75  Subject Score Math 88 Math 88 Math 88  Subject Score PE 77 PE 77 PE 77 PE 81 PE 81 PE 81
2	for A1.len()		
3		=file(A1(A2))	
4		=B3.cursor@t()	
5		>B1=B1|B4	Save the cursor sequence to B1
6	=B1.pjoin()		Join the cursor sequence transversely
7	=A6.fetch()		 Class | StudentID | Subject | Score | Subject | Score | Subject | Score Class one | 1 | English | 84 | Math | 88 | PE | 77 Class one | 2 | English | 81 | Math | 88 | PE | 77 Class one | 3 | English | 75 | Math | 88 | PE | 77 Retrieve number from cursor

Related concepts:

A.merge()

CS.merge@x()

pkey()

r.pkey()

Description:

Obtain the field value of primary key of a record.

Syntax:

r.**pkey**()

Remark:

Get the field value of primary key of r which may be a sequence if the primary key is composed of several fields. If there is not a primary key, the value of the first column will be returned.

If there exists any referenced records in the field values, the referenced record will be replaced by its **pkey()** value.

Parameters:

r	A record

Return value:

The field value of the primary key of *r*

Example:

	A	
1	=demo.query("select * from EMPLOYEE ")	
2	=A1(1).pkey()	**1**; Because there is not a primary key in **A1**, the value of the first field is returned.
3	>A1.primary(NAME)	Set **NAME** as the primary key
4	=A1(1).pkey()	**1**; return the value of the primary key of the first record
5	>A1.primary(EID, DEPT)	
6	=A1(1).pkey()	**[1, Admin]**; Because there are two primary key fields, a sequence composed of these fields' values will be returned
7	=demo.query("select * from DEPARTMENT")	
8	>A1.switch(DEPT,A7)	The value of **DEPT** field is switched to a referenced record
9	=A1(1).pkey()	[1,**Admin**]; the value of the **DEPT** field is the primary key value of the referenced record

Related concepts:

T.primary()

v.v()

pmax()

A.pmax()

Description:

Get the position of the maximum member of a sequence.

Syntax:

A.**pmax(** *x* ,{*k*})

Remark:

Compute the expression *x* against each member of sequence *A* and return the sequence number of the member whose computation is the maximum one. The function may be used to find the position of the maximum value in a sequence.

Parameters:

A	a sequence
x	an expression, which is generally a field name or a legal expression that is composed of field names, "**~**" in which is used to reference the current record.
k	start the searching from the k^{th} member, and it is **1** by default

Options:

@a	Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an *n* integer sequence.
@z	Find the members from back to front from position *k*, and from the last position by default.

Return value:

A sequence number or a sequence composed of sequence numbers

Example:

	A	
1	[2,5,4,3,2,9,4,9,3]	
2	=A1.pmax(~*~)	6
3	=A1.pmax@a(~*~)	[6,8]
4	=A1.pmax@z(~*~)	8
5	=A1.pmax@az(~*~)	[8,6]
6	=A1.pmax(~*~,7)	8
7	=A1.pmax@z(~*~,7)	6
8	=demo.query("select * from EMPLOYEE ")	
9	=A8.pmax(BIRTHDAY)	472

Related concepts:

A.maxp()

A.pmin()

pmin()

A.pmin()

Description:

Get the position of the minimum member of a sequence.

Syntax:

A.**pmin(** *x* {,*k*} **)**

Remark:

Compute the expression *x* against each member of sequence *A* and return the sequence number of the member whose computation is the minimum one. The function may be used to find the position of the minimum value in a sequence.

Parameters:

A	a sequence
x	an expression, which is generally a field name or a legal expression that is composed of field names, "**~**" in which is used to reference the current record.
k	start the searching from the k^{th} member, and it is **1** by default

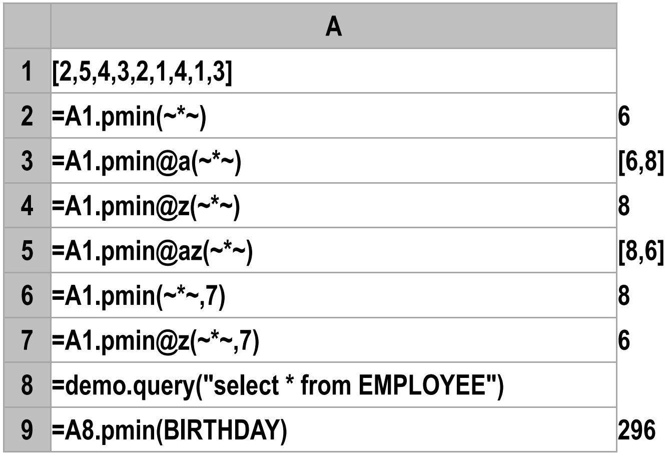
Options:

@a	Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an *n* integer sequence.
@z	Find the members from back to front from position *k*, and from the last position by default.

Return value:

A sequence number or a sequence composed of sequence numbers

Example:



	A	
1	[2,5,4,3,2,1,4,1,3]	
2	=A1.pmin(~*~)	6
3	=A1.pmin@a(~*~)	[6,8]
4	=A1.pmin@z(~*~)	8
5	=A1.pmin@az(~*~)	[8,6]
6	=A1.pmin(~*~,7)	8
7	=A1.pmin@z(~*~,7)	6
8	=demo.query("select * from EMPLOYEE")	
9	=A8.pmin(BIRTHDAY)	296

Related concepts:

A.minp()

A.pmax()

pos()

pos()

Description:

Search the position of a substring in a parent string, and return null if not found

Syntax:

pos(s_1, s_2{, *begin*}**)**

Remark:

Search the position of the substring s_2 in the parent string s_1 from the beginning position *begin*, and return null if not found

Parameters:

s_1	Parent string in which you want to search the substring
s_2	Substring to be searched
begin	The starting character to be searched, and the default is 1

Options:

@z Search forward starting from the begin, and the search will be started from back to forth by default.

Return value:

Integer

Example:

pos("abcdef","def")	4
pos("abcdefdef","def",5)	7
pos("abcdef","defa")	null
pos@z("abcdeffdef","def",7)	4

A.pos(*x*{,*k*})

Description:

Get the position of a member in a sequence.

Syntax:

A.**pos(***x*{,*k*}**)**

Remark:

Locate the position of a member *x* in sequence *A*, and *x* may appear in *A* repeatedly. The return value is determined by the options. If not found, then return null.

Parameters:

A	a sequence
x	a member
k	start the searching from the k^{th} member, and it is **1** by default

Options:

@b *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable.

@a Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an n integer sequence.

@z Find the members from back to front from position k, and from the last position by default.

@s Members of *A* are in order. With the binary search, return the position of *x* if *x* is a member of *A*; otherwise, return a number opposite to the sequence number at which position the *x* can be inserted orderly.

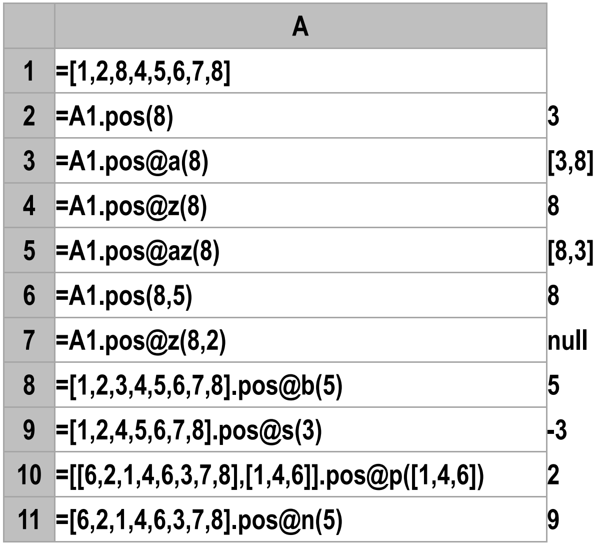
@p If *x* is a sequence, then treat it as an single value. In this case, *A* is a sequence composed of sequences

@n If no sequence member is found, return the length of A plus 1. This option is mutual exclusive to @a.

Return value:

A sequence number or a sequence composed of sequence numbers

Example:



	A	
1	=[1,2,8,4,5,6,7,8]	
2	=A1.pos(8)	3
3	=A1.pos@a(8)	[3,8]
4	=A1.pos@z(8)	8
5	=A1.pos@az(8)	[8,3]
6	=A1.pos(8,5)	8
7	=A1.pos@z(8,2)	null
8	=[1,2,3,4,5,6,7,8].pos@b(5)	5
9	=[1,2,4,5,6,7,8].pos@s(3)	-3
10	=[[6,2,1,4,6,3,7,8],[1,4,6]].pos@p([1,4,6])	2
11	=[6,2,1,4,6,3,7,8].pos@n(5)	9

Note:

If *A* is not a sorted sequence, then options **@b** and **@s** should not be used, or it may bring about the incorrect result. For example:

- [7,6,1,2,3].pos@b(3)   return **null**

Related concepts:

A.pos(x)

A.psort()

A.pos(*x*)

Description:

Get the position of a sequence member in another sequence.

Syntax:

A.pos(*x*)

Remark:

x is a sequence, return ISeq *p* to make *A*(*p*)==*x*. If not found, then return null.

Parameters:

A	a sequence
x	a sequence

Options:

@i Return single ascending ISeq *p* to make *A*(*p*)==*x*

@c Return the position in which the sequence x firstly appears in A. By doing so, seek the position of sub sequence x in the sequence A. If x is not the sub sequence of A, then return null.

@b *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable

Return value:

The unique Ascending integer sequence *p* which makes *A*(*p*)**==***x*

Example:

– **[6,2,1,4,6,3,7,8].pos@i([1,4,6])**	**[3,4,5]**, which is a unique Ascending integer sequence
– **[6,2,1,4,6,3,7,8].posi([1,4,6])**	**[3,4,1]**, which may not be a unique Ascending integer sequence
– **[1,2,3,4,6,7,8].pos@b([3,1,4,6])**	**[3,1,4,5]**
– **[2,1,4,6,3,7,8].pos@i([8,4,6])**	**null**
– **[2,1,4,6,3,7,8,4,6,1].pos@c([4,6])**	**[3]**

Related concepts:

A.pos()

A.psort()

power()

Description:

Compute the powers of a numeric value

Syntax:

power(*x*, *n***)**

Remark:

Compute the *n* powers of *x*

Parameters:

x	Base
n	Power

Return value:

Numeric

Example:

– **power(2,4)**	**16.0**

Related concepts:

exp(n)

primary()

T.primary()

Description:

Set primary key for a table sequence.

Syntax:

T.**primary**(F_1 {,F_i,…})

Remark:

Set F_1{,F_i,…} as the primary key of *T*. If no parameter is given, then remove the primary key, and meanwhile *T*.create will copy the primary key .

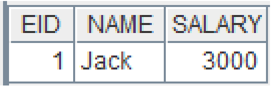
Parameters:

T	table sequence
F_i	primary key

Return value:

T with a primary key

Example:

	A	
1	=demo.query("select EID,NAME,SALARY from EMPLOYEE")	
2	>A1.primary(NAME)	Set **NAME** as the primary key of **A1**
3	=A1(1).pkey()	[**Rebecca**]
4	>A1.primary()	Delete primary key
5	=A1(1).pkey()	Because no key field is set for A1, return the first field value
6	>A1.primary(NAME)	
7	=A1.create()	If *T* has primary key, then also copy the primary key together with *T*
8	=A7.insert(0,1,"Jack",3000)	 EID NAME SALARY 1 Jack 3000
9	=A8(1).pkey()	[**Jack**]

Related concepts:

r.pkey()

v.v()

prior()

r.prior()

Description:

Among the records, query fields referred by the foreign key recursively

Syntax:

r.**prior**(*F*,*r'*,*n*)

Remark:

In record *r*, recursively query field *F* referred by the foreign key, like [*r*. *F*, *r*. *F*. *F*,…]. The query will not stop until the value of field *F* in the last item is equal to *r'*. If *r'* is omitted, its value will be null and the recursive query will be performed through all the records to the end. If the specified record *r'* does not exist, then return null. *n* is the maximum recursive number, its value is 1,000 by default.

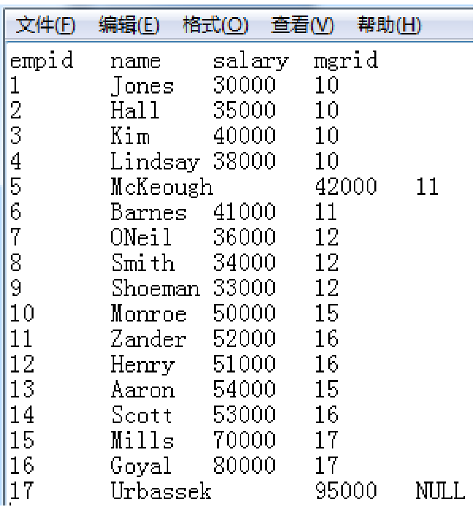
Parameters:

r	Records
F	Field name
r'	Record
n	Number

Return value:

A sequence

Example:

	A	
1	=file("D://emp.txt").import@t()	
2	>A1.switch(mgrid,A1:empid)	
3	=A1.select@1(name=="Barnes").prior(mgrid)	Based on mgrid field referred by the foreign key , select all records of people who are superior to Barnes
4	=A1.select@1(name=="Barnes").prior(mgrid,A1.select@1(name=="Urbassek"),5)	Judge if Urbassek is superior to Barnes and list all records of people superior to Barnes but inferior to Ubassek

```
empid   name     salary  mgrid
1       Jones    30000   10
2       Hall     35000   10
3       Kim      40000   10
4       Lindsay  38000   10
5       McKeough         42000   11
6       Barnes   41000   11
7       ONeil    36000   12
8       Smith    34000   12
9       Shoeman  33000   12
10      Monroe   50000   15
11      Zander   52000   16
12      Henry    51000   16
13      Aaron    54000   15
14      Scott    53000   16
15      Mills    70000   17
16      Goyal    80000   17
17      Urbassek         95000   NULL
```

proc()

db.proc()

Description:

Call a storage procedure by a database connection.

Syntax:

db.**proc**(*sql*, *param*:*type*:*mode*:*variable*, ...)

Remark:

Call the storage procedure of a database and return a table sequence composed of the executing results of the storage procedure.

The output variables will be generated into the current Context, which can be referenced in the current program cellset.

Parameters:

db	Database connection
sql	The execution statement for the storage procedure; for example, **call test(?,?)**
param	The value of an input argument.
type	The data type of an argument
mode	**"i"** or **"o"**. **"i"** indicates an input argument and **"o"** indicates an output argument.
variable	The output argument's name, which will be referenced in the current program cellset

Values for *type*:

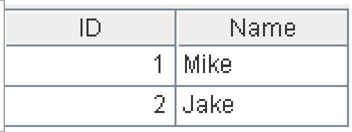
```
public final static byte DT_DEFAULT = (byte) 0; // By default, automatic recognition.
public final static byte DT_INT = (byte) 1;
public final static byte DT_LONG = (byte) 2;
public final static byte DT_SHORT = (byte) 3;
public final static byte DT_BIGINT = (byte) 4;
public final static byte DT_FLOAT = (byte) 5;
public final static byte DT_DOUBLE = (byte) 6;
public final static byte DT_DECIMAL = (byte) 7;
public final static byte DT_DATE = (byte) 8;
public final static byte DT_TIME = (byte) 9;
public final static byte DT_DATETIME = (byte) 10;
public final static byte DT_STRING = (byte) 11;
public final static byte DT_BOOLEAN = (byte) 12;

public final static byte DT_INT_ARR = (byte) 51;
public final static byte DT_LONG_ARR = (byte) 52;
public final static byte DT_SHORT_ARR = (byte) 53;
public final static byte DT_BIGINT_ARR = (byte) 54;
public final static byte DT_FLOAT_ARR = (byte) 55;
public final static byte DT_DOUBLE_ARR = (byte) 56;
public final static byte DT_DECIMAL_ARR = (byte) 57;

public final static byte DT_DATE_ARR = (byte) 58;
public final static byte DT_TIME_ARR = (byte) 59;
public final static byte DT_DATETIME_ARR = (byte) 60;
public final static byte DT_STRING_ARR = (byte) 61;
public final static byte DT_BYTE_ARR = (byte) 62;
public final static byte DT_CURSOR = (byte) 101;
public final static byte DT_AUTOINCREMENT = (byte) 102;
```

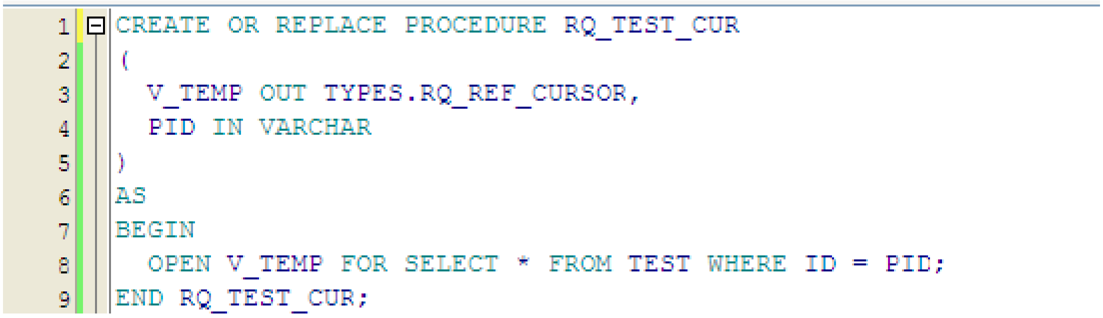
Example:

- Return a record through a storage procedure

	A	
1	=orac.proc("{call RQ_TEST_CUR(?,?)}",:101:"o":table1,1:0:"i":)	
2	=table1	**A2** references the output variables **table1** generated by **A1**

ID	Name
1	Mike
2	Jake

The contents of the storage procedure are as below:

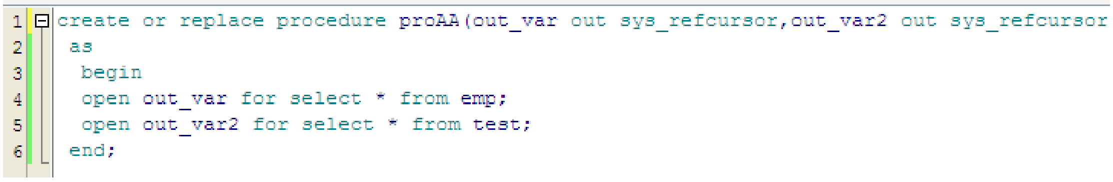


```
CREATE OR REPLACE PROCEDURE RQ_TEST_CUR
(
  V_TEMP OUT TYPES.RQ_REF_CURSOR,
  PID IN VARCHAR
)
AS
BEGIN
  OPEN V_TEMP FOR SELECT * FROM TEST WHERE ID = PID;
END RQ_TEST_CUR;
```

- Return table sequences through a storage procedure

	A	
1	=orac.proc("{call proAA(?,?)}",:101:"o":a,:101:"o":b)	**[[7369,7499,7521,…],[1,2,3]]**, return a sequence composed of multiple table sequences; each table sequence is referenced by an output variable
2	=A1(1)	Return the first table sequence in the sequence of **A1**
3	=a	**A3** and **A4** are to reference the output variables generated by **A1**
4	=b	

The stored procedure contents are as follows:



```
create or replace procedure proAA(out_var out sys_refcursor,out_var2 out sys_refcursor
 as
  begin
  open out_var for select * from emp;
  open out_var2 for select * from test;
 end;
```

Related concepts:

db.execute()

db.query()

property()

f.property()

Description:

Retrieve property value from property file

Syntax:

f.property(*p*)

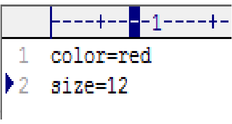
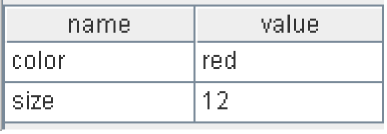
Remark:

Retrieve property *p* from property file *f* and return. If omitting *p*, then return *a* TSeq composed of all properties.

Parameters:

f	File object
p	Property name

Example:

	A	
1	=file("E://test.property").property("color")	Property file  1 color=red 2 size=12 , export"**red**"
2	=file("E://test.property").property()	 name: color, size value: red, 12

Related concept:

f.read()

f.write()

pseg()

A.pseg(*x*)

Description:

Return the position of a member in a sequence

Syntax:

A.pseg(*x*)

Remark:

Return the position of *x* in sequence *A*, which must be an ordered one. If *x* does not exist in *A*, return the position where *x* can be inserted by the order.

Parameters:

A	A sequence
x	An expression

Return value:

The ranking of member *x*

Example:

	A	
1	[2,22,122,222,2222]	
2	=A1.pseg(12)	2

3	=A1.pseg(222)	4

Related concepts:

A.ranki(y,x)

A.ranki(y)

pselect()

A.pselect()

Description:

Get the positions of the selected members from a sequence.

Syntax:

A.**pselect(** *x* {,*k*}**)**

A.**pselect(**x_1:y_1,x_2:y_2,......x_i:y_i{,*k*}**)** the simplified syntax for multiple conditions using "**&&**" to do the union query, it is equal to *A*.**pselect(**x_1**==**y_1 **&&** x_2**==**y_2 **&&......** x_i**==**y_i {,*k*}**)**.

Remark:

Return the sequence number of a member that fulfils condition *x*, and the return value is subject to the options. If not found, then return the empty sequence or null.

Options:

@a Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an *n* integer sequence.

@z Find the members from back to front from position *k*, and from the last position by default.

@b *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable. Note: x_i must be totally sorted ascendingly or descendingly. If *A* is not a sorted sequence, then option **@b** should not be used, or it may bring about the incorrect result. When option **@b** is used in *A*.**pselect(**x_1:y_1,x_2:y_2,......x_i:y_i{,*k*}**)**, that is to find out the members whose "Return value" are **0** in **cmp(***x*,*y***)**, **cmp()** is not needed in this syntax, *A*.**pselect@b(**x_1:y_1,x_2:y_2,......x_i:y_i{,*k*}**)** is enough.

@s The member in *A* is ordered for formula *x*. With the binary search, if none members in *A* can make the formula *x* generate a result of 0, then return a number opposite to the position at which the number meeting the conditions can be inserted.

@n If no sequence member is found, return the length of *A* plus 1. This option is mutual exclusive to @a

Parameters:

A A sequence

x an Boolean expression, which may be null. when using option **@b**, *x* should be an expression whose return value is a number

x_i:y_i	x_i is an expression, and y_i is a comparing value
k:	start the searching from the k^{th} member, and it is **1** by default

Return value:

The sequence number of a member that fulfils condition *x*. Use option **@a** to return a sequence composed of all the sequence numbers that fulfils condition *x*, and return a sequence composed of the sequence numbers of all the members when *x* is null.

Example:

	A	
1	[2,5,4,3,2,9,4,9,3]	
2	=A1.pselect(~>4)	2
3	=A1.pselect@a(~>4)	[2,6,8]
4	=A1.pselect@z(~>4)	8
5	=A1.pselect@az(~>4)	[8,6,2]
6	=A1.pselect(~>4,7)	8
7	=A1.pselect@z(~>4,7)	6
8	=[1,3,5,7,9].pselect@b(~-5)	**3**, The syntax of **A8** and **A9** are equal, and the return values of **A8** and **A9** are the same. When using a colon, the syntax while using **@b** needs not to be changed
9	=[1,3,5,7,9].pselect@b(~:5)	
10	=[1,3,5,7,9].pselect@s(~:4)	**-3**,Return the opposite number of insertable position number when 4 is not found
11	=[1,3,5,7,9].pselect@n(~:4)	6
12	=demo.query("select * from EMPLOYEE")	
13	=A12.pselect(GENDER:"M",DEPT:"Sales")	6
14	=demo.query("select * from EMPLOYEE where GENDER='M' order by EID asc")	**EID** and **GENDER** are in the same order
15	=A14.pselect@b(EID:10,GENDER:"M")	2
16	=A14.pselect@s(EID:3,GENDER:"M")	**-1**,Return the opposite number of insertable position number when the record whose **EID =3** and **GENDER** is M cannot be found
17	=A14.pselect@n(EID:3,GENDER:"M")	**239**

Note:

If *A* is not a sorted sequence, and x_i is totally sorted in ascending or descending order, then options **@b** and **@s** should not be used, or it may bring about the incorrect result.

Related concepts:

A.select()

psort()

A.psort()

Description:

Get the positions of the sorted members of a sequence.

Syntax:

A.**psort** **(** *x* **)**

A.**psort(***x_i*:*d_i*,......**)**

Remark:

Generate a new sequence composed of the sequence numbers of all the members in sequence *A* in the order of the values of expression *x*.

Parameters:

A	an *n* sequence
x	the sorting expression
x_i:*d_i*	If multiple expressions are used for the sorting, it may be in the format of *x_i*:*d_i*,.., which is sorted in the order of *x_i*,…., *d_i* of which indicates the sorting direction, 1 indicates ascending, -1 indicates descending , 0 indicates the original order.

Options:

@i	return the rankings of each member in the original order.

Return value:

The new sequence composed of the sequence numbers of all the members in sequence *A* in the order of the values of expression *x*

Example:

	A	
1	[a,c,e,g,f,d,b]	
2	=A1.psort(~)	[1,7,2,6,3,5,4]
3	=A1.psort@i(~)	[1,3,5,7,6,4,2]
4	=demo.query("select * from EMPLOYEE")	
5	=A4.psort(DEPT:1,BIRTHDAY:-1)	It is sorted in ascending order of **DEPT** first, and then sorted in descending order of **BIRTHDAY**.

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.swap(p,q)

A.rvs()

ptop()

A.ptop()

Description:

Get the sequence numbers of the top n smallest members in a sequence

Syntax:

A.**ptop**(*x*,…;*n*)

Remark:

x is an expression based on which compute for each member of a sequence. A sequence comprising the sequence numbers of the top n smallest members in the original ISeq will be returned.

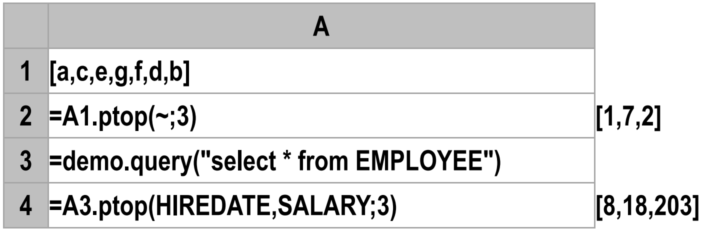
Parameter:

A	A sequence
x	Sort expression
n	Integer

Return value:

A sequence composed of sequence numbers of members

Example:



	A	
1	[a,c,e,g,f,d,b]	
2	=A1.ptop(~;3)	[1,7,2]
3	=demo.query("select * from EMPLOYEE")	
4	=A3.ptop(HIREDATE,SALARY;3)	[8,18,203]

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.psort()

A.top()

query()

db.query()

Description:

To execute a sql statement by a database connection and obtain the query results.

Syntax:

db.**query**(*sql* {,*args* …})

db.**query**(*A*,*sql*{,*args* …})

Run the *sql* and return a table sequence composed of the query results. Here, *args* may be an expression computed against *A* *db*.**query**(*queryArgs*; *switchArgs*) Run switch after query (**@1** is not allowed here)

Remark:

Query a specified *sql* in the data source *db* and return a table sequence composed of the query results of *sql* (where: *db* is a database connection).

The table following FROM in the *sql* is a single table; the primary key of the table sequence for results is automatically set.

Parameters:

db Database connection

sql A *sql* statement. For example, **select * from table**

args If there are parameters in the *sql*, they must be converted into arguments; *args* can be either constants or expressions. Note: Arguments shall be separated by commas.

A A sequence; *sql* is executed for each member of *A*, in general, *args* are computed against each member of *A* and then sent to *sql* for execution.

queryArgs Arguments for a query functions, which can be either *sql* {,*args* …} or *A*,*sql*{,*args* …}

switchArgs Arguments for a switch function, which may be F_i:…, A_i:*x*;…

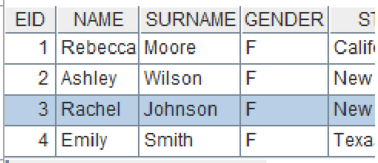
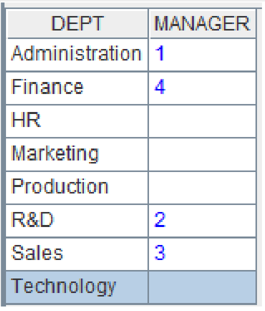
Options:

@1 Return the first record, which may be a single value of one field or a sequence composed of multiple field values.

Return value:

A table sequence composed of results of *sql*.

Example:

	A	
1	=demo.query("select * from EMPLOYEE where DEPT=?","Sales")	
2	=demo.query@1("select * from EMPLOYEE where DEPT=?","Sales")	[3,"Rachel","Johnson","F","New Mexico",1970-12-17,2010-12-01,"Sales",9000]; Use @1 option to return a sequence composed of multiple field values of the first record
3	[1,2,3,4]	
4	=demo.query(A3,"select * from EMPLOYEE where EID=?",~)	
5	=demo.query("select * from DEPARTMENT "; MANAGER,A4:EID)	

EID	NAME	SURNAME	GENDER	ST
1	Rebecca	Moore	F	Calif
2	Ashley	Wilson	F	New
3	Rachel	Johnson	F	New
4	Emily	Smith	F	Texa

DEPT	MANAGER
Administration	1
Finance	4
HR	
Marketing	
Production	
R&D	2
Sales	3
Technology	

		The "**MANAGER**" field of the result will reference the record of **A4**.

Related concepts:

db.execute()

db.proc()

r.(*x*)

Description:

Compute an expression against a record and return the result.

Syntax:

r.(*x*)

Remark:

Compute the expression *x* against record *r*, and return the result.

Parameters:

r	a record
x	an expression, which is generally a field name or a legal expression that is composed of field names. Use "**~**" to reference the current record.

Return value:

The computation of *x*

Example:

	A	
1	=[[12,23]].new(~(1):col1,~(2):col2)	
2	=A1(1).(col2+1)	24

r.*F*

The *r* is a record, and the *r*.*F* is the value of field *F* of record *r* .

Besides using *r*.*F* to indicate field value, you can also use *r*.**#***i*, of which **#***i* means the ith field.

r.*F*=*x*

Description:

Assign value to a field of a record.

Syntax:

r.*F*=*x*

Remark:

Assign *x* to field *F* of record *r*

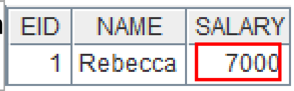
Parameters:

r	record
F	field name
x	expression, the computation of which is the field value of *F*

Return value:

The computation of *x*

Example:

	A	
1	=demo.query("select EID,NAME, SALARY from Employee")	
2	>A1(1).SALARY=A1(1).SALARY+100	The value of "**SALARY** " in **A1(1)** is changed into **7100.00**

EID	NAME	SALARY
1	Rebecca	7000

rand()

Description:

Get a random value

Syntax:

rand(*n*)

Remark:

Get the integers from 0 to n-1.

Parameters:

n	Integer, by default, the result is a random float number between 0 and 1.0

Options:

@s	Set the seed value for generating random number.

Return value:

Numeric

Example:

- **rand()** Get a random value in the range of **[0, 1.0]**
- **rand()*100** Get a random float value in the range of **[0, 100]**
- **rand(100)** Get a random integer in the range of **[0, 100]**

➢ The seed value for generating random number.

	A	
1	=rand@s(5)	
2	=10.(rand())	
3	= rand@s(5)	
4	=10.(rand())	The 10 random numbers generated in **A4** and **A2** are identical

rands()

Description:

Get a random character string

Syntax:

rands(*s*,*l***)**

Remark:

Generate a character string of length *l* using the characters from s randomly

Parameter:

s	Character string
l	Integer

Return value:

Character string

Example:

=rands("abc",5)	Get a character string of length 5 comprising the character string "**abc**"

rank()

A.rank(*x*)

Description:

Get the ranking of sequence *A*.(*x*)

Syntax:

*A***.rank(***x***)** Equivalent to *A***.** (*x*)**.rank()**

Remark:

Compute the value of expression x according to each member of sequence *A* and return the ranking of sequence *A*.(*x*)

Options:

@z Rank members in ascending order (in descending order by default). Note: Here "**z**" is in lowercase

@i Remove duplicate members from sequence *A*.(*x*) and then compute the ranking

Parameters:

x	An expression computed according to sequence *A*
A	A sequence

Return value:

The ranking of members of sequence *A*.(*x*)

Example:

	A
1	=demo.query("select * from SCORES where

	SUBJECT='English'")	
2	=A1.rank(SCORE)	[7,9,13,1,17,5,13,19,3,25,11,21,23,25,7,9,13,1,17,5,13,19,3,25,11,21,23,25] Rank members in descending order
3	=A1.rank@z(SCORE)	[21,19,13,27,11,23,13,9,25,1,17,7,5,1,21,19,13,27,11,23,13,9,25,1,17,7,5,1] Rank members in ascending order
4	=A1.rank@i(SCORE)	[4,5,7,1,8,3,7,9,2,12,6,10,11,12,4,5,7,1,8,3,7,9,2,12,6,10,11,12] It is the ranking after duplicate members are removed
5	=A1.(SCORE).rank()	Same requirement as that in **A2**

Related concepts:

A.rank()

A.rank()

Description:

Compute the ranking of each member in a sequence

Syntax:

A.**rank()**

Remark:

Compute the ranking of each member in sequence *A* and return a sequence composed of rankings of members. By default the result is sorted in descending order

Parameters:

A A sequence

Options:

@z Rank members in ascending order. Note: Here "**z**" is in lowercase

Return value:

A sequence composed of rankings of members of sequence *A*

Example:

	A	
1	=[2,1,3,4,8,5]	
2	=A1.rank()	[5,6,4,3,1,2] Rank members in descending order
3	=A1.rank@z()	[2,1,3,4,6,5] Rank members in ascending order

Related concepts:

A.rank(x)

ranki()

A.ranki(*y*,*x*)

Description:

Get the ranking of a certain sequence member after the sequence is computed

Syntax:

A.**ranki**(*y*,*x*) Equivalent to *A*.(*x*).**ranki**(*y*)

Remark:

Compute the value of expression *x* according to each member of sequence *A* and return the ranking of *y* in sequence *A*.(*x*)

Parameters:

x An expression computed according to sequence *A*

y A member of sequence *A* or it is used to compare with values of sequence *A*.(*x*)

A A sequence

Options:

@z Rank members in ascending order. Note: Here "**z**" is in lowercase

@i Remove duplicate members from sequence *A*.(*x*) and then compute the ranking of *y* in in the sequence

Return value:

The ranking of member *y*

Example:

	A	
1	=demo.query("select * from SCORES where SUBJECT='English'")	
2	=A1.ranki(90,SCORE)	5 members ranked in descending order
3	=A1.ranki@z(90,SCORE)	23 members ranked in ascending order
4	=A1.ranki@i(90,SCORE)	3 the ranking after duplicate members are removed
5	=A1.(SCORE).ranki(90)	5

Related concepts:

A.ranki(y)

A.ranki(*y*)

Description:

Compute the ranking of a member in a sequence

Syntax:

*A***.ranki(***y***)**

Remark:

Compute the ranking of member *y* in sequence *A* and return the ranking of *y*. By default the result is sorted in descending order

Parameters:

A A sequence

y A member of sequence *A*

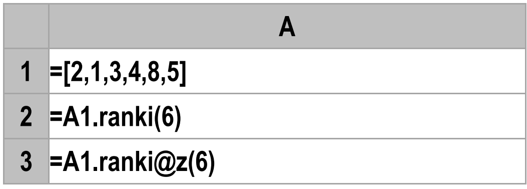
Options:

@z Rank members in ascending order. Note: Here "**z**" is in lowercase

Return value:

A sequence composed of rankings of members in sequence *A*

Example:



	A
1	=[2,1,3,4,8,5]
2	=A1.ranki(6)
3	=A1.ranki@z(6)

2 members ranked in descending order

6 members ranked in ascending order

Related concepts:

A.ranki(y,x)

read()

f.read()

Description:

Read contents from a file object.

Syntax:

f.**read**()

Remark:

Fetch the contents of the file object *f* as a string.

Parameters:

f A file object

Options:

@n Return the contents of the file object *f* as a string sequence; each row is corresponding to a member.

@v Interpret the returned string as the corresponding data type. The combined use of this option and @n is acceptable

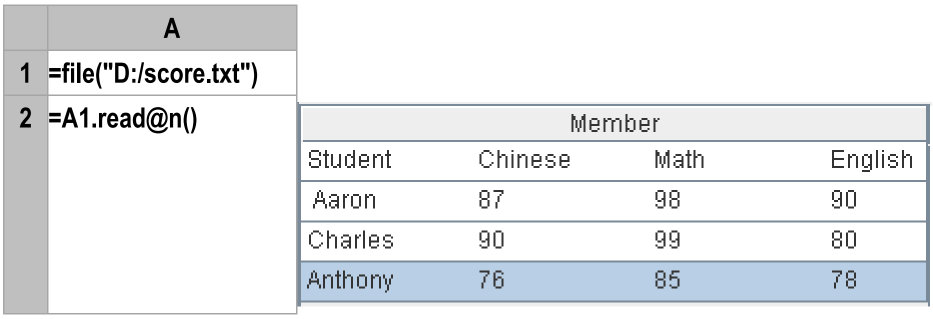
@b Read in as binary data

Return value:

A string

Example:

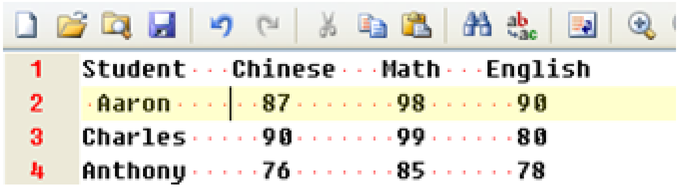
- **file("D:\\test.txt").read()** Read the contents of the file object *f* as a string.
- **file("D:\\test.png").read@b()** Read out the contents in the file object as binary data
- Use **@n** to connect the rows in the txt file into a sequence which is to be returned.



	A
1	=file("D:/score.txt")
2	=A1.read@n()

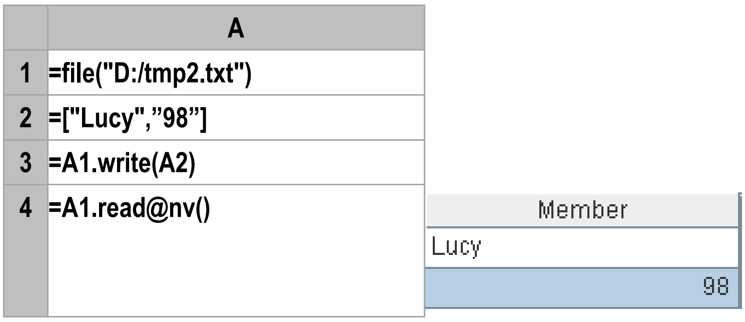
Member
Student Chinese Math English
Aaron 87 98 90
Charles 90 99 80
Anthony 76 85 78

The contents in the "score.txt" file:



```
Student   Chinese   Math   English
 Aaron     87        98     90
Charles    90        99     80
Anthony    76        85     78
```

- Use **@v** to automatically interpret the returned string as the corresponding data type.



	A
1	=file("D:/tmp2.txt")
2	=["Lucy",”98”]
3	=A1.write(A2)
4	=A1.read@nv()

Member
Lucy
98

The **Lucy** is interpreted as string, and the **98** is interpreted as integer.

Related concepts:

f.write()

record()

T.record()

Description:

Make the members of sequence *A* the new field values of records of TSeq *T*

Syntax:

T.**record**(*A*,*k*)

Remark:

From the record with a specified position *k*, reassign values to the fields of TSeq *T*’s records one after another with members of sequence *A*. The number of to-be-reassigned records are determined by the number of both *A*’s members and *T*’s fields. If the number of *A*’s members is indivisible by the number of *T*’s fields, *A*’s remaining members will go on to reassign values to the fields of the next record.

Parameters:

k	An integer; this means the records will be modified from the *k*th one By default or when *k* ==0, append records to TSeq *T* to go on with the modification
A	A sequence whose members are used to reassign values to *T*’s fields
T	A TSeq

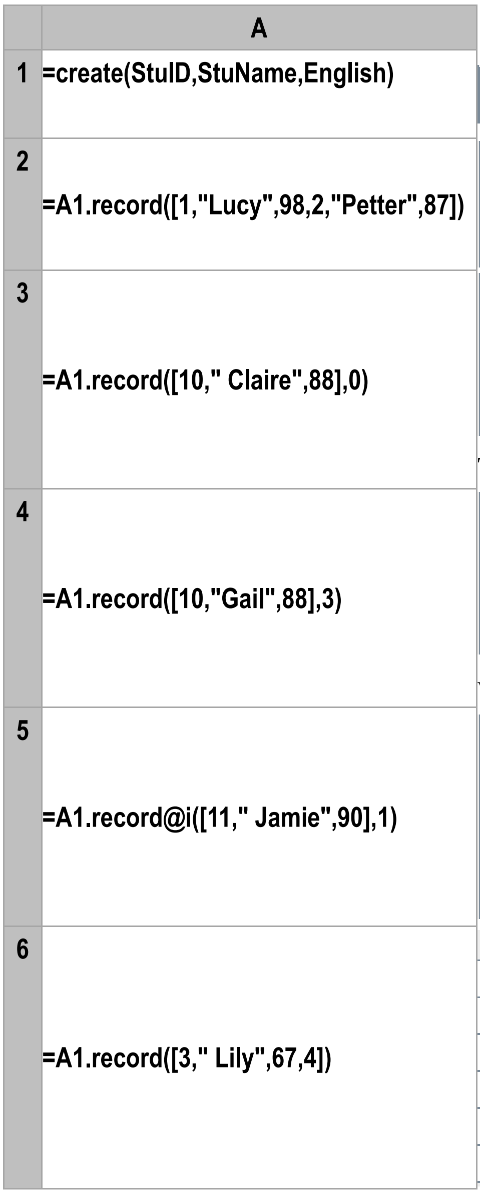
Options:

@i	Insert one or more records before the record at the specified position *k* for modifcation

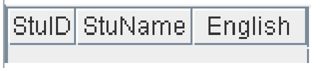
Return value:

The modified TSeq *T*

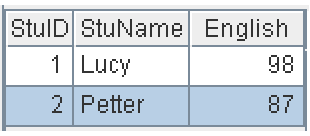
Example:



	A
1	=create(StuID,StuName,English)
2	=A1.record([1,"Lucy",98,2,"Petter",87])
3	=A1.record([10," Claire",88],0)
4	=A1.record([10,"Gail",88],3)
5	=A1.record@i([11," Jamie",90],1)
6	=A1.record([3," Lily",67,4])

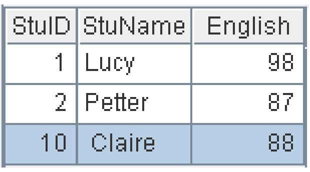


StuID	StuName	English



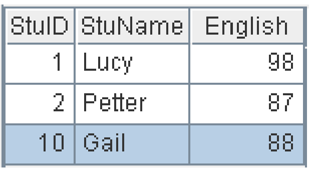
StuID	StuName	English
1	Lucy	98
2	Petter	87

Append records to the TSeq by default



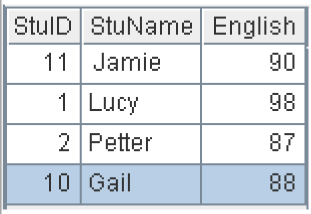
StuID	StuName	English
1	Lucy	98
2	Petter	87
10	Claire	88

Append records to the TSeq when *k*=0



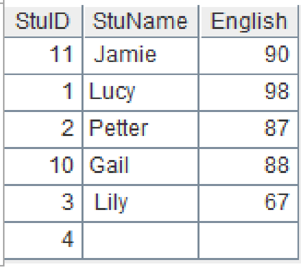
StuID	StuName	English
1	Lucy	98
2	Petter	87
10	Gail	88

Modify the 3rd record when ***k*=3**



StuID	StuName	English
11	Jamie	90
1	Lucy	98
2	Petter	87
10	Gail	88

Insert a record before the 1st record when both ***k*=1** and **@i** is used



StuID	StuName	English
11	Jamie	90
1	Lucy	98
2	Petter	87
10	Gail	88
3	Lily	67
4		

4, A's remaining member, is entered into the next record

r.record()

Description:

Make the members of sequence *A* the new field values of record *r*

Syntax:

r.record(*A*)

Remark:

Enter members of of sequence *A* into the fields of record *r*. If the number of A's members is greater than *r*'s fields, throw the surplus members away; if the former is lesser than the latter, keep the orginal values of the remaining fields.

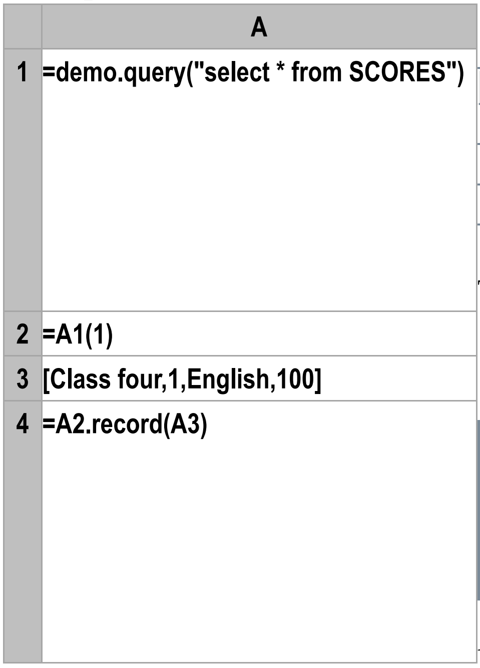
Parameters:

r	A record
A	A sequence

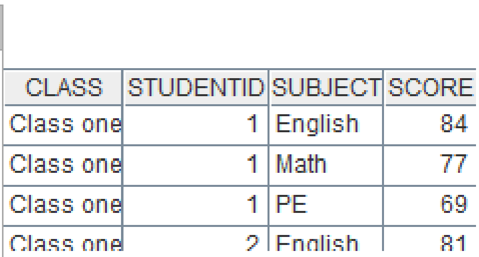
Return value:

A record

Example:

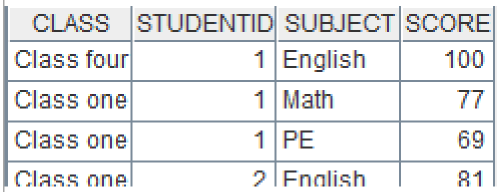


	A
1	=demo.query("select * from SCORES")
2	=A1(1)
3	[Class four,1,English,100]
4	=A2.record(A3)



CLASS	STUDENTID	SUBJECT	SCORE
Class one	1	English	84
Class one	1	Math	77
Class one	1	PE	69
Class one	2	English	81

The original TSeq in A1



CLASS	STUDENTID	SUBJECT	SCORE
Class four	1	English	100
Class one	1	Math	77
Class one	1	PE	69
Class one	2	English	81

A1's TSeq after the first record is modified

regex()

s.regex()

Description:

Match the character string with regular expression

Syntax:

s.regex(*rs*)

Remark:

With the regular expression *rs*, match the string *s*, and return an array of section matches. If no match is found, then return null

Parameter:

s	Character strings
rs	Regular expression

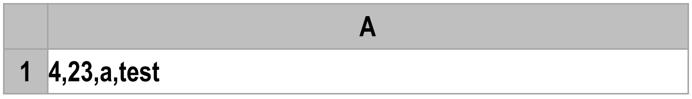
Options:

@c	Case is insensitive
@u	Use Unicode

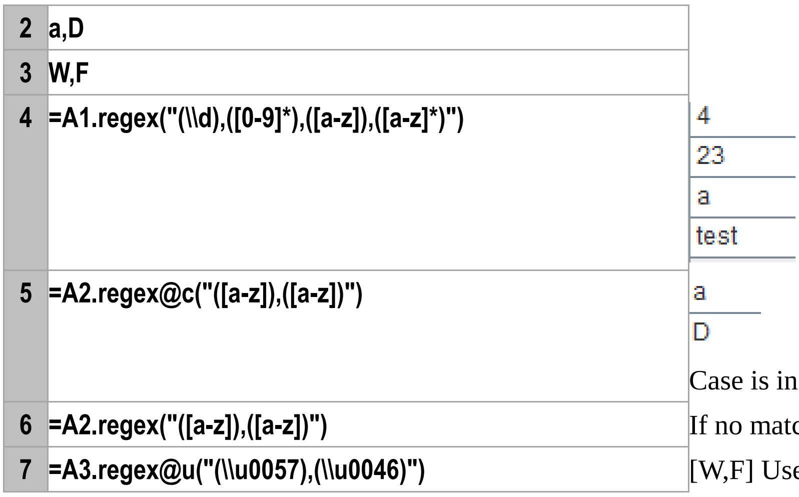
Return value:

An array of section matches

Example:



	A
1	4,23,a,test



2	a,D	
3	W,F	
4	=A1.regex("(\\d),([0-9]*),([a-z]),([a-z]*)")	4 23 a test
5	=A2.regex@c("([a-z]),([a-z])")	a D Case is insensitive
6	=A2.regex("([a-z]),([a-z])")	If no match is found, then return null
7	=A3.regex@u("(\\u0057),(\\u0046)")	[W,F] Use unicode to match

Related concepts:

A.regex()

cs.regex()

A.regex()

Description:

Match the string members with the regular expression

Syntax:

A.**regex**(*rs*,*F_i*) If no extracting item is specified in *rs* which represents regular expressions, match field F_i of the string data type with *rs*. Then return the new RSeq after RSeq *A* is filtered. Use the first field of *A* if omitting F_i

A.**regex**(*rs*;*F_i*,…) If one or more extracting items are specified in *rs*, split the string members of sequence *A* according to them, and return the results as a TSeq whose fields are F_i

Remark:

Match the members of string type in the sequence *A* with the regular expression *rs*. The results will be merged into a TSeq whose fields are F_i for returning

Parameter:

A A sequence or an RSeq whose members are strings

rs Regular expressions. The extracting items are specified sub-regular expressions which are separated from each other by separators and each is surrounded with the parentheses. They will match the corresponding fields respectively. For example, **"(.*),(a.*)"** are two extracting items separated by a comma.

F_i Resulting field names of string type

Options:

@c Case is insensitive

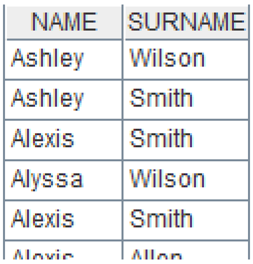
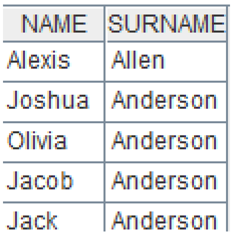
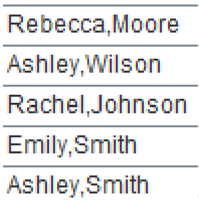
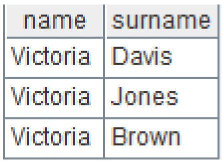
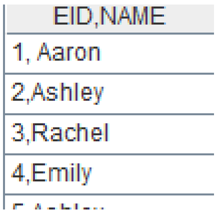
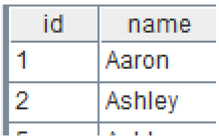
@u Use Unicode

Return value:

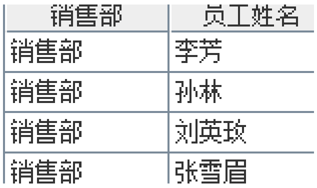
A TSeq

Example:

	A

1	=demo.query("select NAME,SURNAME from EMPLOYEE")	
2	=A1.regex("A.*")	 NAME SURNAME Ashley Wilson Ashley Smith Alexis Smith Alyssa Wilson Alexis Smith By default, match the first field of the string with *rs* if omitting F_i
3	=A1.regex("A.*",SURNAME)	 NAME SURNAME Alexis Allen Joshua Anderson Olivia Anderson Jacob Anderson Jack Anderson Match *rs* with sepecified fields
4	=A1.(~.string())	 Rebecca,Moore Ashley,Wilson Rachel,Johnson Emily,Smith Ashley,Smith Convert to string sequences
5	=A4.regex("(V.*),(.*)";id,name)	 name surname Victoria Davis Victoria Jones Victoria Brown Match members of A4 with the regular expressions and return a TSeq
6	=file("D:\\a.txt").import@ts()	 EID,NAME 1, Aaron 2,Ashley 3,Rachel 4,Emily
7	=A6.(#1).regex@c("(.*),(a.*)";id,name)	 id name 1 Aaron 2 Ashley Match names that start with a or A
8	=demo.query("select DEP, NAME from EMPLOYEE")	
9	=A8.(~.string())	

10	=A9.regex@u("(\\u9500\\u552e\\u90e8),(.*)";SalesDep., Employee Names)



销售部	员工姓名
销售部	李芳
销售部	孙林
销售部	刘英玫
销售部	张雪眉

Use Unicode to match the Sales Dep.

Related concepts:

s.regex()

cs.regex()

cs.regex()

Description:

Match the member of character string in the cursor with the regular expression

Syntax:

cs.**regex**(*rs*,F_i,…)

Remark:

Match the member of character string in the cursor *cs* with the regular expression *rs*. Return the result as a cursor whose fields are F_i

Parameter:

cs	Member is the cursor of character string
rs	Regular expression
F_i	Result field name

Options:

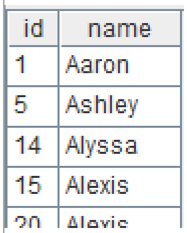
@c	Case is insensitive
@u	Use Unicode

Return value:

A cursor

Example:

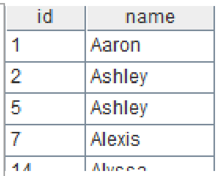
	A
1	=file("D:\\a.txt").import@ts()
2	=A1.cursor()
3	=A1.cursor()
4	=A2.regex("(.*),(A.*)";id,name).fetch()



id	name
1	Aaron
5	Ashley
14	Alyssa
15	Alexis
20	Alexis

Match employees whose names start with A

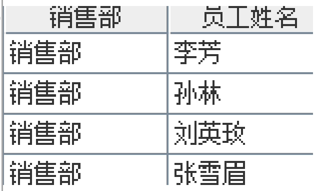
5	=A3.regex@c("(.*),(A.*)";id,name).fetch()



id	name
1	Aaron
2	Ashley
5	Ashley
7	Alexis
14	Alyssa

Match employees whose names start with a or A

6	=file("D:\\c.txt":"UTF-8").import@ts()
7	=A6.cursor()
8	=A7.regex@u("(\\u9500\\u552e\\u90e8),(.*)";销售部,员工姓名).fetch()



销售部	员工姓名
销售部	李芳
销售部	孙林
销售部	刘英玫
销售部	张雪眉

Use Unicode to match the Sales Dep.

Related concepts:

s.regex()

A.regex()

rename()

T.rename()

Description:

Rename the fields in a table sequence

Syntax:

T.rename (*F*, *F'*)

Remark:

Rename *F* field as *F'*.

Parameters:

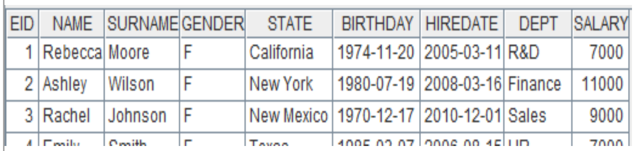
F	Field names.
F'	The modified field names
T	Table sequence

Return value:

Table sequences.

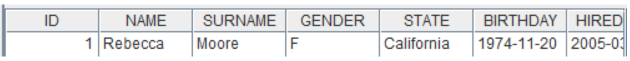
Example:

	A
1	=demo.query("select * from EMPLOYEE")



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	[illegible]	[illegible]	HR	7000

	A
2	=A1.rename(EID:ID)



ID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIRED
1	Rebecca	Moore	F	California	1974-11-20	2005-03

replace()

Description:

Change the substring of a source string

Syntax:

replace (*src*,*a*,*b*)

Remark:

Change the substring of *src* from *a* to *b*

Parameters:

src	Source string
a	The source substring
b	The target substring

Options:

@q	Quoted characters need not to be replaced

Return value:

String after replacing

Example:

replace("abc'abc'def","a","China")	**"Chinabc'Chinabc'def"**
replace("abc'abc'def","a","China")	**"Chinabc'Chinabc'def"**
replace@q("abc'abc'def","a","China")	**"Chinabc'abc'def"**

reset()

T.reset()

Description:

Clear the members of a table sequence.

Syntax:

T.**reset()**

Remark:

Clear the members in table sequence *T*, but reserve the data structure.

Parameters:

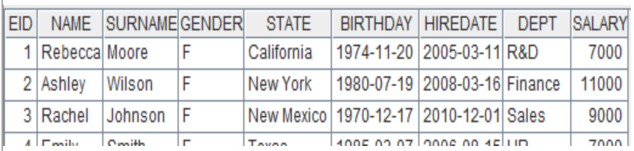
T	A TSeq

Return value:

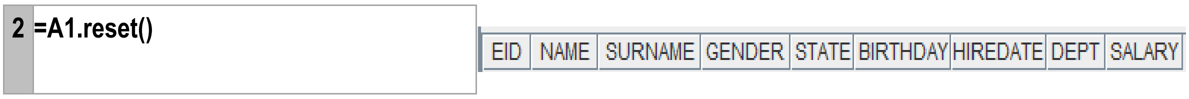
The empty table sequence *T*.

Example:

	A
1	**=demo.query("select * from EMPLOYEE")**



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-02-07	2006-08-15	HR	7000



2	=A1.reset()

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY

result x_i

Description:

Return the result x_i,…, then free up the resource after being terminated by the main program.

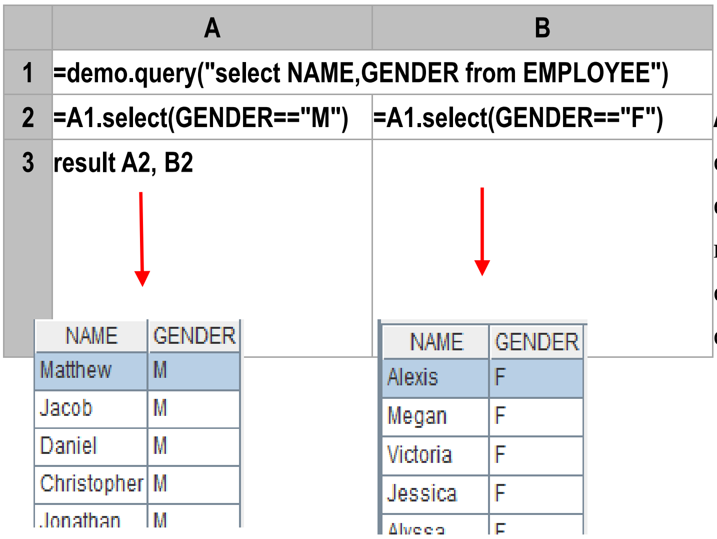
Syntax:

result x_i

Remark:

With **result**, program will return the result *xi*. Generally speaking, the return result is the TSeq, pure RSeq, or cursor. TSeq and pure RSeq are used for the esProc dataset, and cursor can be used for the parallel program.

Example:

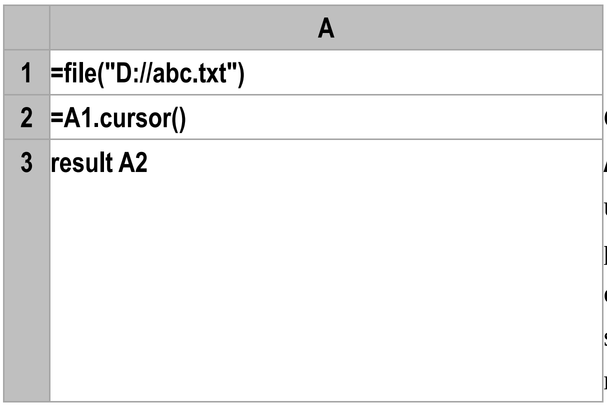


	A	B
1	=demo.query("select NAME,GENDER from EMPLOYEE")	
2	=A1.select(GENDER=="M")	=A1.select(GENDER=="F")
3	result A2, B2	

NAME	GENDER
Matthew	M
Jacob	M
Daniel	M
Christopher	M
Jonathan	M

NAME	GENDER
Alexis	F
Megan	F
Victoria	F
Jessica	F
Alyssa	F

A3 return the two result set of A2 and B2. When this esProc file is used for a report tool, the two datasets can be used as two datasets of the report



	A
1	=file("D://abc.txt")
2	=A1.cursor()
3	result A2

Get fetch cursor

A3 returns the fetch cursor **A2**. When using this esProc file for parallel program, only the cursor which are created on the basis of binary file object sequence will be returned for the use of main program.

rgb()

Description:

Convert the red, green, blue, and transparency value to the corresponding color value

Syntax:

rgb(*redIntExp*, *greenIntExp*, *blueIntExp*{, *alphaIntExp*} **)**

Remark:

The value of *redIntExp*, *greenIntExp*, *blueIntExp*, *alphaIntExp* should be between 0-255.

Parameters:

redIntExp	The integer expression to indicate the red, of which the value is between 0-255
greenIntExp	The integer expression to indicate the green, of which the value is between 0-255
blueIntExp	The integer expression to indicate the blue, of which the value is between 0-255
alphaIntExp	The integer expression to indicate the transparency, of which the value is between 0-255 The 0 represents the radical transparent, and 255 indicates totally opaque. The other values respectively represent the transparency of various levels and the default is 255

Return value:

The 64 bit long integer

Example:

– **rgb(123,123,123)**	**-8684677**
– **rgb(123,123,123,123)**	**2071690107**
– **rgb(123,123,123,255)**	**-8684677**
– **rgb(123,123,123,0)**	**8092539**

right()

Description:

Get the substring from the right of a string

Syntax:

right(*s*, *n***)**

Remark:

Get the substring from the right of string *s*, the length of which is *n*.

Parameters:

s	Source string from which to get the substring
n	Get the length of substring

Return value:

String

Example:

– **right("abced",2)**	**"ed"**

Related concepts:

left()

mid()

rmv()

Description:

To remove variables

Syntax:

rmv(v_1, {v_2,......v_i}**)**

Parameters:

v_i variables to be removed

Return value:

none

Example:

	A	
1	**>@name="tiger"**	To begin with **@** while defining it
2	**=@name**	**"tiger"**, to begin with **@** while using it
3	**>rmv(@name)**	To remove global variable **@name**
4	**=@name**	Error, unrecognizable expression: **@name**

rollback()

db.rollback()

Description:

To roll back the database transaction.

Syntax:

db.**rollback** ()

Remark:

Roll back the transaction, which is the same as **rollback()** of the Connection class in Java.

Parameters:

db Database connection

Example:

	A	B	
1	**=file("D://files//student.txt")**		
2	**=A1.import@t()**		
3	**=connect@e("demo")**		Create a connection and automatically control the commit and rollback operations
4	**>A3.execute@k(A2,"update STUDENTS2 set NAME=?,GENDER=?,AGE=? where ID=?",NAME,GENDER,AGE,ID)**		The transaction is not committed
5	**=A3.error()**		Read the error code generated due to the execution of the previous *sql*

6	if A5==0	>A3.commit()	Commit if there is not an error
7	else	>A3.rollback()	Roll back if there is an error
8	>A3.close()		Close the connection

Related concepts:

db.close()

db.error()

db.commit()

connect()

round()

Description:

Truncate the data at the specified position, and round off the remaining part

Syntax:

round(*numberExp*, {*nExp*})

Remark:

Truncate the data *numberExp* at the specified position *nExp*, and round off the remaining part

Parameters:

numberExp	Data to be intercepted
nExp	Integer to specify the position at which to intercept **>0**: Move the decimal point to the right for *nExp* places **<0**: Move the decimal point to the left for *nExp* places **=0**: Indicate the current decimal places.

Return value:

Numeric

Example:

round(3451251.274,0)	3451251.0
round(3451251.274,-1)	3451250.0
round(3451251.274,-2)	3451300.0
round(3451251.274,1)	3451251.3
round(3451251.274,2)	3451251.27

Related concepts:

ceil()

floor()

run()

r.run(x_i,…)

Description:

Compute an expression against a record and return the record itself.

Syntax:

r.**run**(x_i,...)

Remark:

Compute the expression *x* against record *r* and return record *r* itself. This function is usually used to change the field values of *r*, for example, *x* is **col1=col2+1**, which will change the field value of **col1.**

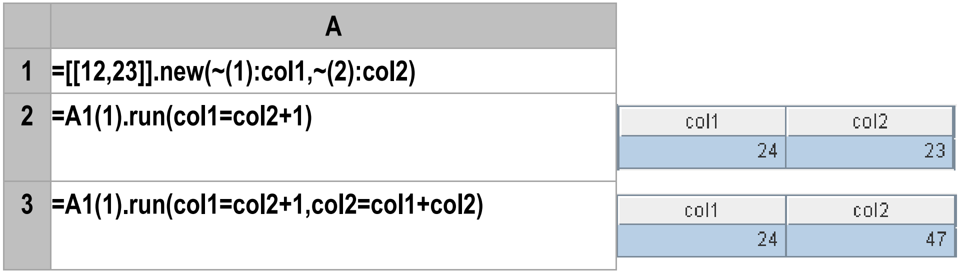
Parameters:

r	a record
x_i	an expression, which is generally a field name or a legal expression that is composed of field names. Use "**~**" to reference the current record.

Return value:

r, which may be changed during the computation of expression *x*.

Example:



	A
1	=[[12,23]].new(~(1):col1,~(2):col2)
2	=A1(1).run(col1=col2+1)
3	=A1(1).run(col1=col2+1,col2=col1+col2)

col1	col2
24	23

col1	col2
24	47

Note:

The differences between *r*.(*x*) and *r*.**run**(*x*): *r*.(*x*) is aimed to compute the value of expression *x* and return it; *r*.**run**(*x*) is aimed to make some changes on *r* through the computation of *x*, and thereby return *r* which has been modified.

Related concepts:

A.run(x_1,x_2,…x_i)

A.run(x_i:F_i,…)

A.run(x_i:F_i,…)

Description:

Compute expressions against each member of a sequence.

Syntax:

A.**run**(x_i:F_i,…)

Remark:

Compute expression x_i against each record in a record sequence/a table sequence, assigning the results to F_i and return the *A* that has been modified. In the expression x_i, use "**~**" to reference the current member of *A*. The status of the records which have been modified will not be changed.

Parameters:

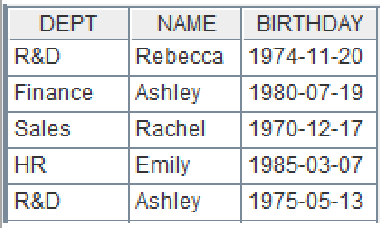
A	a record sequence/table sequence
x_i	The new value of F_i
F_i	The field name of *A*

Return value:

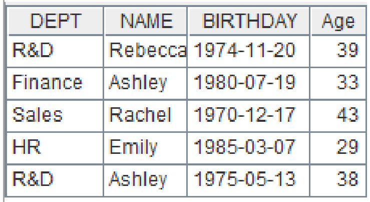
The sequence *A* that has been modified

Example:

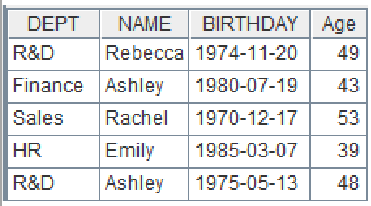
	A
1	=demo.query("select DEPT,NAME,BIRTHDAY from EMPLOYEE")
2	=A1.derive(age(BIRTHDAY):Age)
3	=A2.run(Age+10:Age)



DEPT	NAME	BIRTHDAY
R&D	Rebecca	1974-11-20
Finance	Ashley	1980-07-19
Sales	Rachel	1970-12-17
HR	Emily	1985-03-07
R&D	Ashley	1975-05-13



DEPT	NAME	BIRTHDAY	Age
R&D	Rebecca	1974-11-20	39
Finance	Ashley	1980-07-19	33
Sales	Rachel	1970-12-17	43
HR	Emily	1985-03-07	29
R&D	Ashley	1975-05-13	38



DEPT	NAME	BIRTHDAY	Age
R&D	Rebecca	1974-11-20	49
Finance	Ashley	1980-07-19	43
Sales	Rachel	1970-12-17	53
HR	Emily	1985-03-07	39
R&D	Ashley	1975-05-13	48

Related concepts:

r.run()

A.run(x_1,x_2,…x_i)

A.run(x_1,x_2,…x_i)

Description:

Compute expressions against each member in a sequence and return the sequence itself.

Syntax:

A.run(*x*)	Compute one expression only
A.run(x_1,x_2,…x_i)	Compute multiple expressions

Remark:

Compute the expressions x_i against each member in *A* and return *A* itself. This function is usually used to change the field values of *A*, for example, *x* is **col1=col2+1**, which will change the field value of **col1.** "**~**" in *x* is used to reference the current member in *A*.

Parameters:

A	a sequence/a record sequence
x	an expression, which is generally a field name or a legal expression that is composed of field names, "**~**" in which is used to reference the current record.

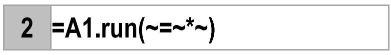
Return value:

Sequence *A* whose field values may have been modified

Example:

Use "**run**" function to modify the member values

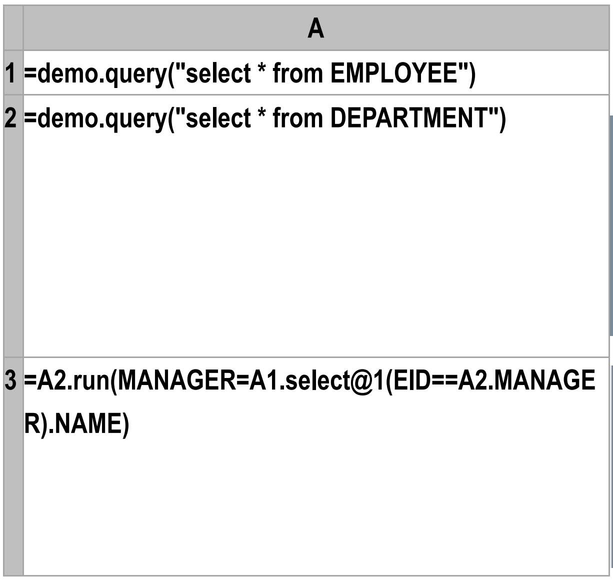
	A
1	=[1,2,3,4,5]



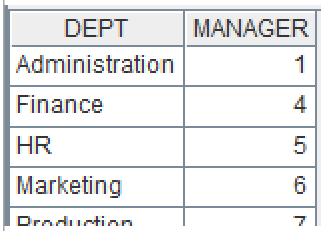
2	=A1.run(~=~*~)

[1,4,9,16,25], use "~" to reference the current member.

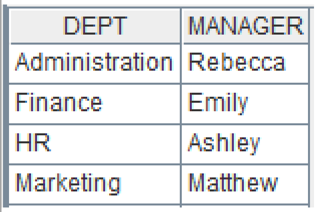
Use " **run**" function to realize the transformation between **ID** and **Name**



	A
1	=demo.query("select * from EMPLOYEE")
2	=demo.query("select * from DEPARTMENT")
3	=A2.run(MANAGER=A1.select@1(EID==A2.MANAGER).NAME)

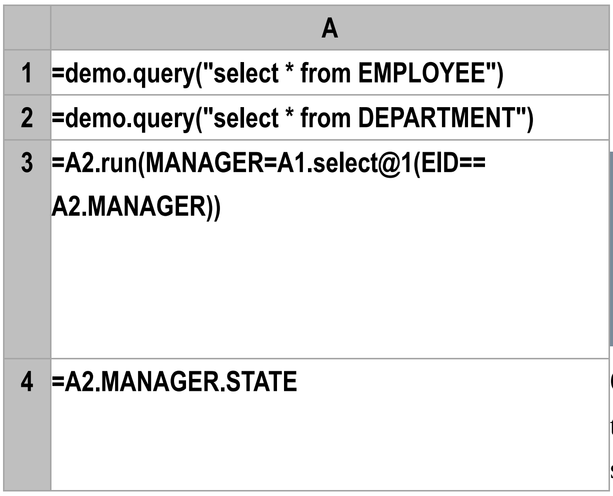


DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6
Production	7

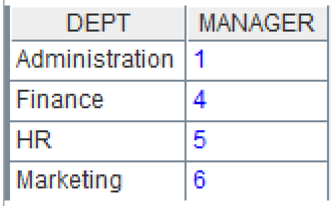


DEPT	MANAGER
Administration	Rebecca
Finance	Emily
HR	Ashley
Marketing	Matthew

Use "**run**" function to realize the association between tables



	A
1	=demo.query("select * from EMPLOYEE")
2	=demo.query("select * from DEPARTMENT")
3	=A2.run(MANAGER=A1.select@1(EID==A2.MANAGER))
4	=A2.MANAGER.STATE



DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6

California, when the field value is a record, the dot operators is used to do the reference stage by stage.

Note:

The difference between *A*.(*x*) and *A*.**run**(*x*) :

The aim of *A*.(*x*) is to compute the values of expression *x* and return a sequence that is composed of the values of this expression;

The aim of *A*.**run**(*x*) is to make some changes on *A* through the computation of *x* and thereby return *A* which has been modified

Related concepts:

r.run()

A.run(x_i:F_i,…)

cs.run()

Description:

Use a formula to compute over the record in a cursor, and return the result as a cursor

Syntax:

cs.**run** (x_i:F_i,…)

Remark:

Use the formula x_i to compute over each record in the cursor. The result will be assigned to F_i, and the modified *cs* will be returned. In the process of computation, the state of the modified record will remain unchanged.

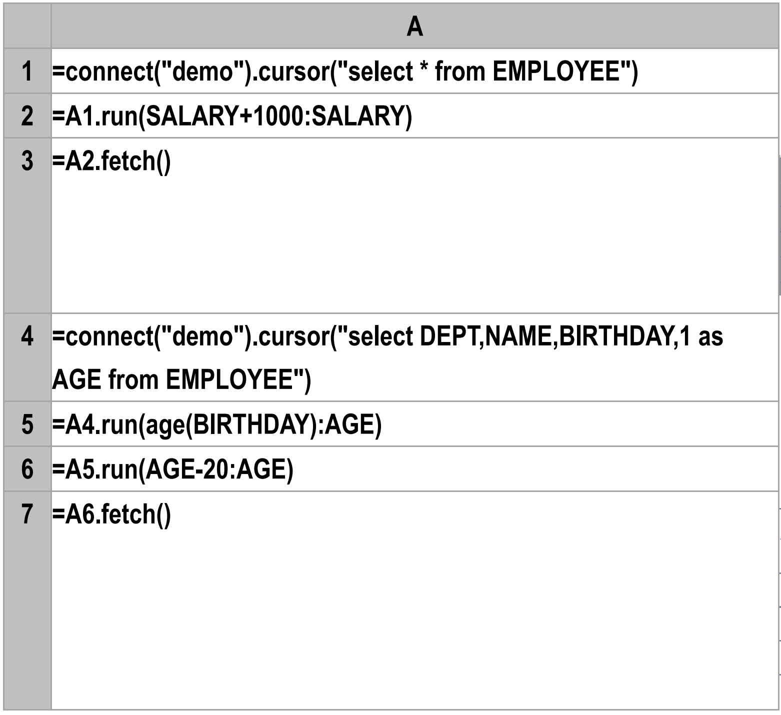
Parameters:

cs	Cursor
x_i	New field value of F_i
F_i	Field name of *cs*

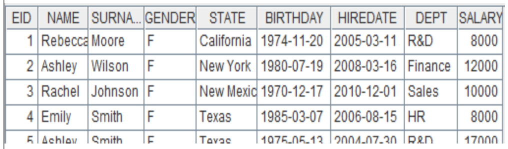
Return value:

Cursor

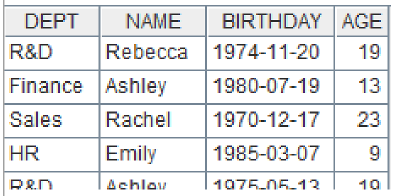
Example:



	A
1	=connect("demo").cursor("select * from EMPLOYEE")
2	=A1.run(SALARY+1000:SALARY)
3	=A2.fetch()
4	=connect("demo").cursor("select DEPT,NAME,BIRTHDAY,1 as AGE from EMPLOYEE")
5	=A4.run(age(BIRTHDAY):AGE)
6	=A5.run(AGE-20:AGE)
7	=A6.fetch()



EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	8000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	12000
3	Rachel	Johnson	F	New Mexic	1970-12-17	2010-12-01	Sales	10000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	8000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	17000



DEPT	NAME	BIRTHDAY	AGE
R&D	Rebecca	1974-11-20	19
Finance	Ashley	1980-07-19	13
Sales	Rachel	1970-12-17	23
HR	Emily	1985-03-07	9
R&D	Ashley	1975-05-13	19

Related concepts:

A.run(xi:Fi,…)

rvs()

A.rvs()

Description:

Generate a new sequence by reversing the members in a sequence.

Syntax:

A.**rvs()**

Remark:

Generate a new sequence by reversing the members in *A*.

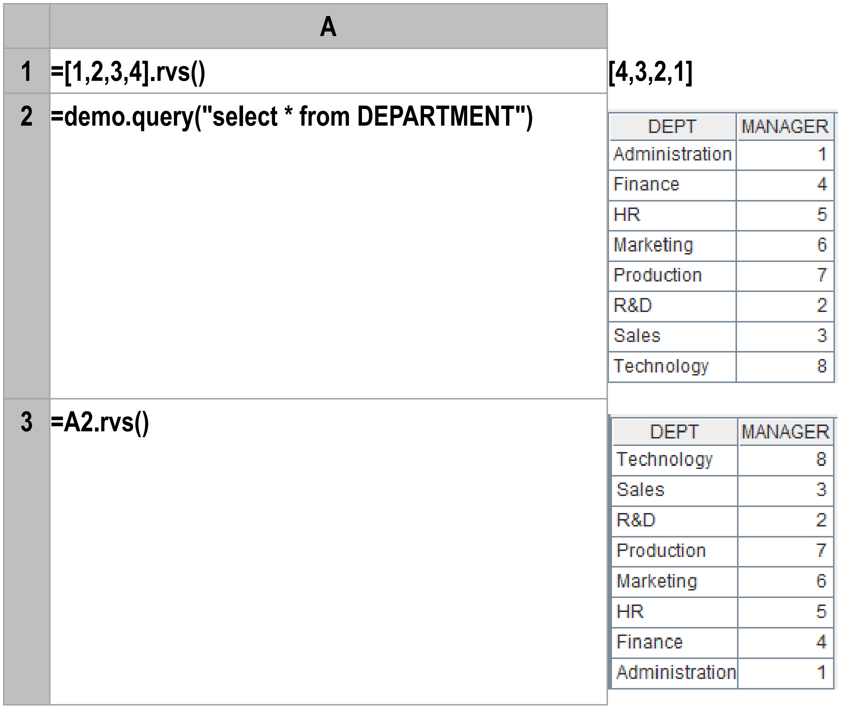
Parameters:

A　　　　A sequence

Return value:

The new sequence by reversing the members in *A*

Example:



	A	
1	=[1,2,3,4].rvs()	[4,3,2,1]
2	=demo.query("select * from DEPARTMENT")	
3	=A2.rvs()	

DEPT	MANAGER
Administration	1
Finance	4
HR	5
Marketing	6
Production	7
R&D	2
Sales	3
Technology	8

DEPT	MANAGER
Technology	8
Sales	3
R&D	2
Production	7
Marketing	6
HR	5
Finance	4
Administration	1

Related concepts:

A.psort()

A.sort()

A.swap(p,q)

second()

Description:

Get the second from a time

Syntax:

second(*datetimeExp***)**

Remark:

Get the second from the time *datetimeExp*.

Parameters:

datetimeExp　　　Expression whose result is a time or date time

Return value:

Integer

Example:

–	**second(datetime("19800227","yyyyMMdd"))**	**0**
–	**second("1972-11-08 10:20:30")**	**30**
–	**second(datetime("2006-01-15 13:20:45"))**	**45**

Related concepts:

year()

month()

day()

hour()

minute()

millisecond()

select()

A.select()

Description:

Pick out members from a sequence which satisfied a condition.

Syntax:

A.**select(** *x* **)**

A.**select(**x_1:y_1, x_2:y_2, ...x_i:y_i**)** The simplified syntax used for the combining query with multiple conditions using "&&", which is equal to *A*.**select(**x_1**==** y_1 && x_2**==** y_2 **&&......** x_i**==**y_i**).** While using option **@b** in case of multiple conditions, **cmp()** is not needed in this syntax and *A*.**select@b(**x_1:y_1, x_2:y_2, ...x_i:y_i**)** is enough

Remark:

Compute the expression *x* against each member of the sequence *A*, then generate a new sequence composed of those members which make the value of the expression x to be true.

Parameters:

A	A sequence
x	A Boolean expression, which may be null. when using option **@b**, *x* should be an expression whose return value is a number
x_i:y_i	x_i is an expression, and y_i is a comparing value

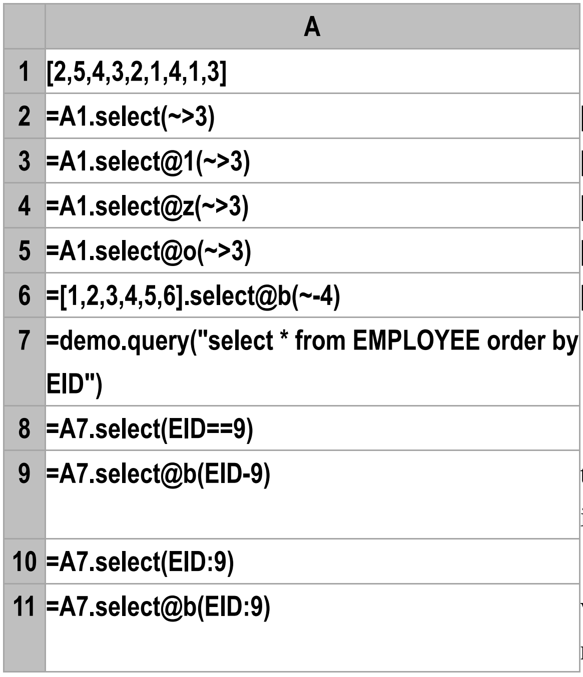
Options:

@1	return the first member that fulfills the conditions.
@z	search the members from back to front
@b	*A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable. Note: If *A* is not a sorted sequence, then option **@b** should not be used, or it may bring about the incorrect result. When option **@b** is used in *A*.**select(**x_1:y_1, x_2:y_2, ...x_i:y_i**)**, that is to find out the members whose "Return value" are **0** in **cmp(***x*,*y***)**, **cmp()** is not needed in this syntax, *A*.**select@b(**x_1:y_1, x_2:y_2, ...x_i:y_i**)** is enough.
@o	Do not create a new sequence, but alter the original sequence *A* instead.

Return value:

The new sequence composed of those members which make the value of the expression x to be true

Example:



	A	
1	[2,5,4,3,2,1,4,1,3]	
2	=A1.select(~>3)	[5,4,4], @a by default
3	=A1.select@1(~>3)	[5]
4	=A1.select@z(~>3)	[4,4,5]
5	=A1.select@o(~>3)	[5,4,4], the original sequence A1 is changed
6	=[1,2,3,4,5,6].select@b(~-4)	[4]
7	=demo.query("select * from EMPLOYEE order by EID")	
8	=A7.select(EID==9)	
9	=A7.select@b(EID-9)	the expression *x* need to be converted to an integer in case of @b
10	=A7.select(EID:9)	
11	=A7.select@b(EID:9)	while using the colon syntax, and changes are not needed in case of @b

Related concepts:

A.pselect()

cs.select()

Description:

Generate a new cursor after filtering the records in the cursor

Syntax:

cs.**select**(*x*)

Remark:

Filter the record from cursor *cs* against x, return the cursor. For x, reference A.select(x).

Parameters:

cs	A cursor
x	Filtering conditions

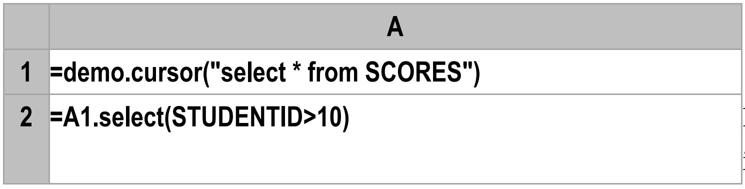
Options:

@c	Find the first continuous range of records which makes x true

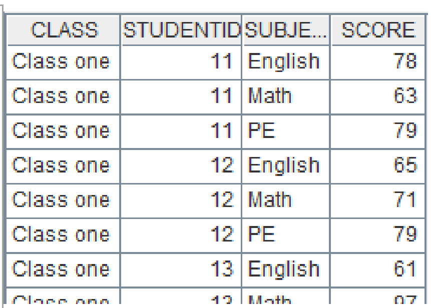
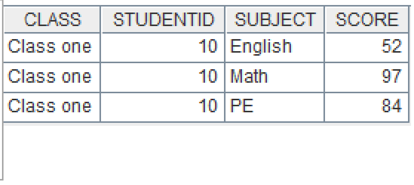
Return value:

Cursor

Example:



	A	
1	=demo.cursor("select * from SCORES")	
2	=A1.select(STUDENTID>10)	Filter the students whose STUDENTID is greater than 10 Score information

3	=A2.fetch()	
4	=demo.cursor("select * from SCORES order by CLASS asc")	CLASS sort in ascending order
5	=A4.select@c(STUDENTID==10)	**@c** option Find the first continuous range of student score data whose STUDENTID is 10
6	=A5.fetch()	

CLASS	STUDENTID	SUBJE...	SCORE
Class one	11	English	78
Class one	11	Math	63
Class one	11	PE	79
Class one	12	English	65
Class one	12	Math	71
Class one	12	PE	79
Class one	13	English	61
Class one	13	Math	97

CLASS	STUDENTID	SUBJECT	SCORE
Class one	10	English	52
Class one	10	Math	97
Class one	10	PE	84

sign()

Description:

To judge whether the parameter is positive, negative or **0**.

Syntax:

sign(*number***)**

Remark:

If *number* is positive value, return **1**; If it is negative value, return **-1**; If it is 0, return **0**

Parameters:

number	Data for which you want to judge whether it is positive or negative

Return value:

Integer

Example:

–	**sign(-10)**	**-1**
–	**sign(30)**	**1**
–	**sign(0)**	**0**

sin()

Description:

Compute the sine value.

Syntax:

sin(*number***)**

Remark:

The parameter *number* is in radians.

Parameters:

number	Radians for which you want to compute the sine

Return value:

float type

Example:

sin(pi())	**1.2246467991473532E-16**
sin(pi(2))	**-2.4492935982947064E-16**
sin(pi()/2)	**1.0**

Related concepts:

cos()

tan()

size()

f.size()

Description:

Return the length of file

Syntax:

***f*.size()**

Remark:

Return the character length of the file *f*

Parameter:

f	file object

Example:

	A
1	=file("E://student.dfx").size()

Export”**11195**”

Related concept:

f.exists()

f. date()

movefile()

skip()

cs.skip()

Description:

To skip records while fetching records from a database cursor.

Syntax:

cs.**skip**(*n*;*x*)

Remark:

Skip *n* records or till *x*… has changed. Return the number of records actually skipped. If *n* is omitted, return the remaining records and close the cursor.

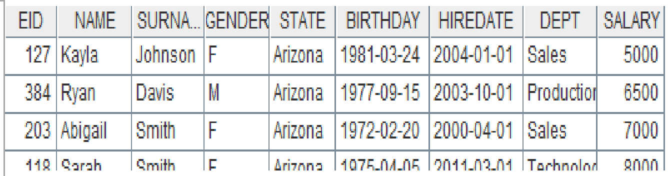
Parameters:

cs	a cursor
n	an integer
x	expression for grouping, and *cs* is sorted by *x*. With *x*, *n* can be omitted.

Return value:

Number of records skipped

Example:

	A	
1	=demo.cursor("select * from EMPLOYEE order by STATE")	Return a cursor of retrieved data, and sort by STATE
2	=A1.skip(50;STATE)	Till the STATE has changed. The number of records actually skipped is 4
3	=A1.fetch()	 Return the remaining cursor

EID	NAME	SURNA...	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
127	Kayla	Johnson	F	Arizona	1981-03-24	2004-01-01	Sales	5000
384	Ryan	Davis	M	Arizona	1977-09-15	2003-10-01	Productior	6500
203	Abigail	Smith	F	Arizona	1972-02-20	2000-04-01	Sales	7000
118	Sarah	Smith	F	Arizona	1975-04-05	2011-03-01	Technolog	8000

Related concepts:

cs.fetch()

db.cursor()

sort()

A.sort()

Description:

Generate a new sequence by sorting the members of a sequence.

Syntax:

A.**sort(** *x*;*loc*)

A.**sort(**x_i:d_i,..... ;*loc*)

Remark:

Generate a new sequence by sorting the members of sequence *A* according to the values of *x* and local language *loc*. If omitting *loc*, then compare Unicode value.

Parameters:

A	A sequence
x	an expression, according to which the members of sequence *A* will be sorted ascendingly.

x_i:d_i If multiple expressions are used for the sorting, it may be in the format of x_i:d_i,.., which is sorted in the order of x_i,…., d_i of which indicates the sorting direction, **1** indicates ascending, **-1** indicates descending , **0** indicates the original order.

loc Language name

Values for *loc*:

ja_JP	Japanese	Japan
es_PE	Spanish	Peru
en	English	
ja_JP_JP	Japanese	Japan
es_PA	Spanish	Panama
sr_BA	Serbian	Bosnia and Herzegovina
mk	Macedonian	
es_GT	Spanish	Guatemala
ar_AE	Arabic	United Arab Emirates
no_NO	Norwegian	Norway
sq_AL	Albanian	Albania
bg	Bulgarian	
ar_IQ	Arabic	Iraq
ar_YE	Arabic	Yemen
hu	Hungarian	
pt_PT	Portuguese	Portugal
el_CY	Greek	Cyprus
ar_QA	Arabic	Qatar
mk_MK	Macedonian	Macedonia
sv	Swedish	
de_CH	German	Switzerland
en_US	English	United States
fi_FI	Finnish	Finland
is	Icelandic	
cs	Czech	
en_MT	English	Malta
sl_SI	Slovenian	Slovenia
sk_SK	Slovak	Slovakia
it	Italian	
tr_TR	Turkish	Turkey
zh	Chinese	
th	Thai	
ar_SA	Arabic	Saudi Arabia
no	Norwegian	
en_GB	English	United Kingdom
sr_CS	Serbian	Serbia and Montenegro

lt	Lithuanian		
ro	Romanian		
en_NZ	English	New Zealand	
no_NO_NY	Norwegian	Norway	Nynorsk
lt_LT	Lithuanian	Lithuania	
es_NI	Spanish	Nicaragua	
nl	Dutch		
ga_IE	Irish	Ireland	
fr_BE	French	Belgium	
es_ES	Spanish	Spain	
ar_LB	Arabic	Lebanon	
ko	Korean		
fr_CA	French	Canada	
et_EE	Estonian	Estonia	
ar_KW	Arabic	Kuwait	
sr_RS	Serbian	Serbia	
es_US	Spanish	United States	
es_MX	Spanish	Mexico	
ar_SD	Arabic	Sudan	
in_ID	Indonesian	Indonesia	
ru	Russian		
lv	Latvian		
es_UY	Spanish	Uruguay	
lv_LV	Latvian	Latvia	
iw	Hebrew		
pt_BR	Portuguese	Brazil	
ar_SY	Arabic	Syria	
hr	Croatian		
et	Estonian		
es_DO	Spanish	Dominican Republic	
fr_CH	French	Switzerland	
hi_IN	Hindi	India	
es_VE	Spanish	Venezuela	
ar_BH	Arabic	Bahrain	
en_PH	English	Philippines	
ar_TN	Arabic	Tunisia	
fi	Finnish		
de_AT	German	Austria	
es	Spanish		
nl_NL	Dutch	Netherlands	
es_EC	Spanish	Ecuador	
zh_TW	Chinese	Taiwan	

ar_JO	Arabic	Jordan
be	Belarusian	
is_IS	Icelandic	Iceland
es_CO	Spanish	Colombia
es_CR	Spanish	Costa Rica
es_CL	Spanish	Chile
ar_EG	Arabic	Egypt
en_ZA	English	South Africa
th_TH	Thai	Thailand
el_GR	Greek	Greece
it_IT	Italian	Italy
ca	Catalan	
hu_HU	Hungarian	Hungary
fr	French	
en_IE	English	Ireland
uk_UA	Ukrainian	Ukraine
pl_PL	Polish	Poland
fr_LU	French	Luxembourg
nl_BE	Dutch	Belgium
en_IN	English	India
ca_ES	Catalan	Spain
ar_MA	Arabic	Morocco
es_BO	Spanish	Bolivia
en_AU	English	Australia
sr	Serbian	
zh_SG	Chinese	Singapore
pt	Portuguese	
uk	Ukrainian	
es_SV	Spanish	El Salvador
ru_RU	Russian	Russia
ko_KR	Korean	South Korea
vi	Vietnamese	
ar_DZ	Arabic	Algeria
vi_VN	Vietnamese	Vietnam
sr_ME	Serbian	Montenegro
sq	Albanian	
ar_LY	Arabic	Libya
ar	Arabic	
zh_CN	Chinese	China
be_BY	Belarusian	Belarus
zh_HK	Chinese	Hong Kong
ja	Japanese	

iw_IL	Hebrew	Israel	
bg_BG	Bulgarian	Bulgaria	
in	Indonesian		
mt_MT	Maltese	Malta	
es_PY	Spanish	Paraguay	
sl	Slovenian		
fr_FR	French	France	
cs_CZ	Czech	Czech Republic	
it_CH	Italian	Switzerland	
ro_RO	Romanian	Romania	
es_PR	Spanish	Puerto Rico	
en_CA	English	Canada	
de_DE	German	Germany	
ga	Irish		
de_LU	German	Luxembourg	
de	German		
es_AR	Spanish	Argentina	
sk	Slovak		
ms_MY	Malay	Malaysia	
hr_HR	Croatian	Croatia	
en_SG	English	Singapore	
da	Danish		
mt	Maltese		
pl	Polish		
ar_OM	Arabic	Oman	
tr	Turkish		
th_TH_TH	Thai	Thailand	TH
el	Greek		
ms	Malay		
sv_SE	Swedish	Sweden	
da_DK	Danish	Denmark	
es_HN	Spanish	Honduras	

Options:

@o Do not create a new sequence, but alter the original sequence *A* instead.

Return value:

The new sorted sequence

Example:

	A	
1	=[2,1,3,8,6,5]	
2	=A1.sort()	[1,2,3,5,6,8]
3	=A1.sort@o()	change **A1** into **[1,2,3,5,6,8]**
4	=["s","e","a","d"].sort(;"en")	[a,d,e,s]

		Sort by the local language “en” for English
5	=demo.query("select * from EMPLOYEE ")	
6	=A5.sort(DEPT:1,BIRTHDAY:-1)	Sorted by **DEPT** in ascending order first, then sorted by **BIRTHDAY** in descending order.

Related concepts:

A.psort()

A.swap(p,q)

A.rvs()

sortx()

cs.sortx()

Description:

Sort the cursor

Syntax:

cs.**sortx**(*x*…;*n*)

Remark:

Sort the cs cursor by *x*, and return as a cursor, *n* is the number of rows in buffer

Parameters:

cs	Cursor
x	a formula to sort the cs members in ascending order
n	number of rows in buffer

Return value:

Cursor

Example:

	A	
1	=demo.cursor("select NAME,BIRTHDAY,HIREDATE from Employee")	Return cursor for data retrieval
2	=A1.sortx(BIRTHDAY;300000)	Sort the BIRTHDAY field in cursor, and there are 300,000 rows set in the buffer
3	=A2.fetch()	Retrieve data from cursor A2

Related concepts:

cs.fetch()

db.cursor()

sqrt()

Description:

Compute the square root

Syntax:

sqrt(*number***)**

Remark:

Compute the square root

Parameters:

number Data for which you want to compute the square root

Return value:

Numeric

Example:

- **sqrt(100)** **10.0**
- **sqrt(99)** **9.9498743710662**

step()

A.step()

Description:

Get members from a sequence with a starting position and a step, so as to create a new sequence.

Syntax:

*A***.step(***m*,k_i, …**)**

Remark:

Find out the members whose sequence numbers are k_i, k_i**+***m*, k_i**+**2*m*,…from *A* to compose a new sequence.

Parameters:

m	a positive integer used to specify the span
k_i	the starting sequence number, **1<=**k_i**<=***m*
A	a sequence whose length is *n*

Return value:

A new sequence whose length is *m*

Example:

	A	
1	**[1,2,3,4,5,6,7,8,9,10]**	
2	**=A1.step(2,1)**	**[1,3,5,7,9]**, Get members at the odd positions.
3	**=A1.step(3,1,2)**	**[1,2,4,5,7,8,10]**, Get two every other one.

string()

A.string(*d*)

Description:

Join all the members of a sequence with a delimiter.

Syntax:

A.string(*d*)

Remark:

Join the members of *A* into a string delimited by *d*, and the subsequence member will be processed.

Parameters:

A	string sequence
d	delimiter and the default is the comma

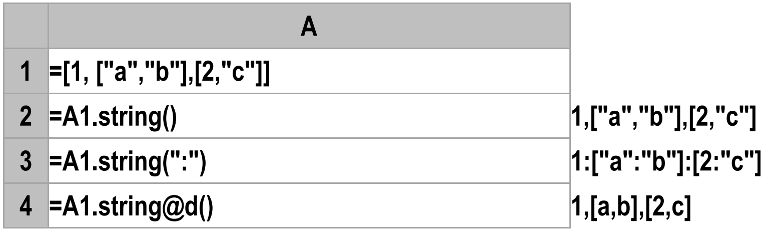
Options:

@d	Quotation marks will not be added when the string members are joined

Return value:

A string after joining

Example:



	A	
1	=[1, ["a","b"],[2,"c"]]	
2	=A1.string()	1,["a","b"],[2,"c"]
3	=A1.string(":")	1:["a":"b"]:[2:"c"]
4	=A1.string@d()	1,[a,b],[2,c]

Related concepts:

s. array()

r.string()

r.string()

Description:

Join the fields of a record with delimiter comma.

Syntax:

r.string()

r.string(F_i,…)

Remark:

Join the fields of *r* into a string delimited by comma as long as these fields could be converted to text.

If there are parameters F_i,…, then only the fields listed in F_i,… are converted.

Parameters:

r	Records
F_i	Field names

Options:

@t	Use "\t" as the delimiter
@f	Transform the field names of *r* but not the field values. And this functionality is only supported by *r*.string().
@d	Determine if a quotation mark should be used according to the data type of the field value

Return value:

The string after joining

Example:

	A

1	=[[1,"lucy",98]].new(~(1):id,~(2):name,~(3):math)	
2	=A1(1).string()	"1,lucy,98"
3	=A1(1).string@t()	"1 lucy 98"
4	=A1(1).string@f()	"id,name,math"
5	=A1(1).string@d()	"1,\"lucy\",98"
6	=A1(1).string(id,name)	"1,lucy"

Related concepts:

A.string()

s.array()

string(*expression*{, *format*})

Description:

Convert the object of other type to the string type and format it.

Syntax:

string(*expression*{, *format*}**)**

Remark:

The format string *format* must match the data type of the result of *expression*, or the result of **string(***expression*{, *format*}**)** may be incorrect.

Parameters:

expression	The constant object or expression to be converted to string.
format	A format string used to format the result of *expression*

Options:

@q Quote the string *expression*, and represent the tab, carriage return, line feed with the escape character. Add the escape character to the single quotes and double quotes. The escape character can be specified with parameter format, and the default is the "\". If the string *expression* contains *format*, then add the escape character *format* before *format*.

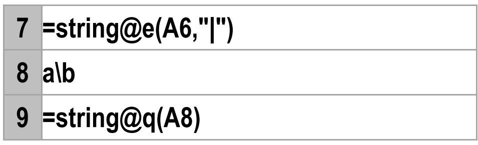
@e Inverse operation of @q. If the *expression* string contains the escape character of *format*, then slash "\" will be used to replace the *format* in the escape characters and then transfer the meaning.

Return value:

String

Example:

	A	
1	=string(123)	123
2	=string(date("2009-02-23")," MMM dd, yyyy")	二月 23, 2009
3	=string(3456.78,"$#,##0.00")	$3,456.78
4	=string(5/6,"0.00%")	83.33%
5	a b	Separate a and b with tab
6	=string@q(A5,"|")	"a|tb" Double quoted the given character string **a b**, and display the the escape character



7	=string@e(A6,"|")	**a b**
8	a\b	
9	=string@q(A8)	**"a\\b"** the default escape character is "\"

Related concepts:

- float()
- int()
- long()
- number()
- decimal()

sum()

A.sum(*x*)

Description:

Compute x with each member of the sequence and compute the summary value of the members of the new sequence

Syntax:

***A*.sum(*x*)** Equivalent to *A*.(*x*).**sum**()

Remark:

Compute *x* on sequence *A* by loop and return the summary value of members of the resulting sequence

Parameters:

A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names. The data type of the computed result of the expression is numerical value.

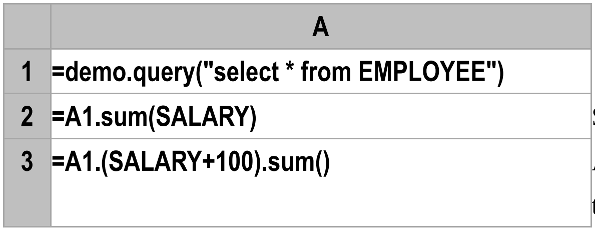
Return value:

A numerical value

Special Note:

Take a null value as zero

Example:



	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.sum(SALARY)	Sum up the salaries of all employees
3	=A1.(SALARY+100).sum()	Add 100 to the salary of each employee and then sum up all the employees' salaries

Related concepts:

A.sum()

A.sum()

Description:

Compute the summary value of members of a sequence

Syntax:

A.**sum()** Equivalent to **sum**(x_1,…,x_n)

Remark:

Compute the summary value of members in sequence *A*. Skip those members that are not numerical values

Parameters:

A A sequence

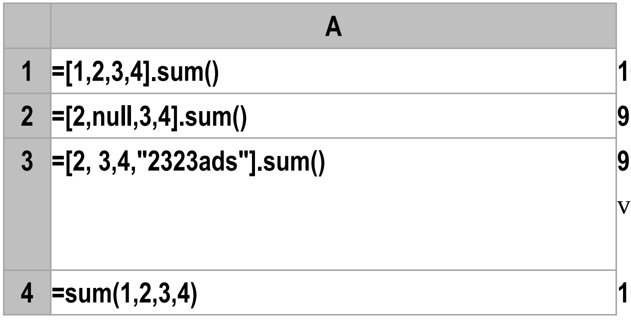
Return value:

The summary value of all members in sequence *A*

Special Note:

Take a null value as zero

Example:



	A		
1	=[1,2,3,4].sum()	10	
2	=[2,null,3,4].sum()	9	Take a null value as zero
3	=[2, 3,4,"2323ads"].sum()	9	Skip members that are not numerical values
4	=sum(1,2,3,4)	10	

Related concepts:

A.count()

A.avg()

A.min()

A.max()

A.variance()

A.sum(x)

sumif()

A.sumif()

Description:

Locate all the positions of a member in a sequence, and get the sum of the members in these positions of another sequence.

Syntax:

A.**sumif**(A_i:x_i, …)

Remark:

Locate all the positions of member x_i in A_i, acquiring the intersection of these positions and return the sum of the members in these positions of *A*

This function is quite convenient in computing constant cells. For example, for the table **Scores** in constant cells, find out the records whose **Subject** columns are "**PE**" and return the sum of the **Scores** of these records.

Parameters:

A_i a sequence

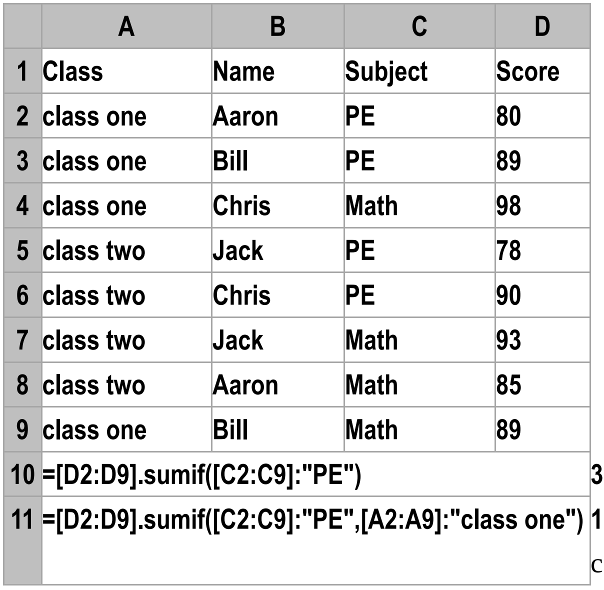
x_i the members in A_i

A the target sequence

Return value:

The sum of the members in those result positions of *A*

Example:



	A	B	C	D
1	Class	Name	Subject	Score
2	class one	Aaron	PE	80
3	class one	Bill	PE	89
4	class one	Chris	Math	98
5	class two	Jack	PE	78
6	class two	Chris	PE	90
7	class two	Jack	Math	93
8	class two	Aaron	Math	85
9	class one	Bill	Math	89
10	=[D2:D9].sumif([C2:C9]:"PE")			
11	=[D2:D9].sumif([C2:C9]:"PE",[A2:A9]:"class one")			

337, find it with a single condition

169, find it with multiple combined conditions

Related concepts:

A.countif(A_i:x_i,…)

A.avgif(A_i:x_i,…)

A.minif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

swap()

A.swap()

Description:

Generate a new sequence by swapping the member positions of two specified intervals of a sequence.

Syntax:

A.swap(*p*,*q*)

Remark:

Generate a new sequence by swapping the member positions of two specified intervals in sequence *A*, and the two intervals should not overlap each other.

Parameters:

A A sequence

p an integer sequence interval composed of positive integers, for example [1,2,3], to (1,3)

q an integer sequence interval composed of positive integers and does not have intersection with *p*, for example [4,5,6], to(4,6)

Return value:

The new sequence after swapping

Example:

	A	
1	[a,b,c,d,e,f,g.h.i.j.k]	
2	=A1.swap(to(1,3),to(4,6))	[d,e,f,a,b,c,g.h.i.j.k]
3	=A1.swap(to(1,4),to(4,6))	There are errors in **A3**: the two intervals should not overlap each other

Note:

The two intervals to be swapped in a sequence should not overlap each other.

switch()

A.switch(*F*$_i$,*A*$_i$:*x*;…)

Description:

Switch the reference field between the field value and the referenced record..

Syntax:

A.**switch**(*F*$_i$, *A*$_i$:*x*;…)

Remark:

Switch F_i between the field value and the referenced record of A_i, the field value is the primary key value of the referenced record of A_i.

If F_i represents multiple fields, the last of such fields will be switched to the referenced record of A_i. For the inverse operation, if the number of the primary keys of A_i is less than that of F_i, the replacement will be done from back to front.

Take the **Attendance** table as an example, the **employeeID** field may store either the code value of **employeeID** or the referenced record of **Employee** table. So, the **switch**() function can be used to switch between the two objects.

When there are multiple primary fields in a code table, it is necessary that F_i represents multiple fields. Take the **Student** table for example, there are two fields – **Class** and **StudentID** – in the table, students in different **Classes** share one **StudentID**; therefore, there may be both **Class** and **StudentID** in the **Scores** table**.** Under such circumstance, if you want to obtain student information by joining the **Student** table, you should join the table with both **Class** and **StudentID.**

In general, the number of F_i must be equal to the number of primary keys.

Use the index table of the foreign key if any, and create one if unavailable.

Parameters:

A	A sequence
F_i	The code fields of *r*. When A_i:*x*;… is omitted, the referenced record should be stored in $F_{i.}$ Therefore, the **switch()** function is used for switching to code values.
F_i, A_i	When A_i is available, code value is stored in F_i which is primary key value of the reference record. Therefore, the **switch()** function is used for switching to the referenced record.

The matching condition is that F_i is equal to the value of the primary key of A_i.

F_i, A_i:x When both A_i and x are available, the matching condition is that F_i is equal to the value of x of A_i

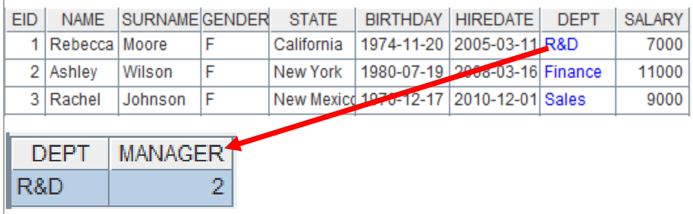
Options: (Available for F_i, A_i and F_i, A_i:x)

@i Delete this record if no corresponding value to F is found in the procedure

Return value:

The record sequence switched.

Example:

	A	
1	=demo.query("select * from DEPARTMENT")	
2	=demo.query("select * from EMPLOYEE")	
3	>A2.switch(DEPT,A1:DEPT)	
4	>A2.switch(DEPT,A1)	No fields associated to A1 is specified. The default field is Dept, and the result is the same as A3.
5	>A2.switch(DEPT)	Switch "**DEPT** " of **A2** to field values
6	>A2.switch@i(DEPT,A1)	Delete this record if no value corresponding to **DEPT** is found

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finance	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000

DEPT	MANAGER
R&D	2

Related concepts:

cs.switch()

cs.switch(F_i, A_i:x;…)

Description:

Cursor is used to switch between the key value of code field and indicator record

Syntax:

cs.**switch**(F_i,A_i:x;…)

Remark:

The F_i will either be A_i or indicator record. The key value is the primary key value of A_i indicator record.

If F_i represents multiple fields, then the last field will be replaced with the records of A_i. Contrarily, if the number of primary keys of A_i is less than that of F_i, then fill it up backwards.

General speaking, the number of F_i must be consistent with the number of primary keys.

Parameters:

cs Cursor

F_i Code field of A; If there is only the F_i parameter, only the records of code table A_i exist in $F_{i.}$ The *switch* function can be used to switch it back to the code value.

F_i, *A_i* If parameter *A_i*, then the *F_i* is the code field of *A*, and there will be code value. The *switch* function can be used to switch it back to the record of *A_i*.

The matching filter is both the *F_i* and *A_i* has the same primary key value.

F_i, *A_i:x* If there are *A_i* and *x* parameters, the first record whose x value is the same to *F_i* will be picked out from *A_i*, and assign to *F_i*.

Options:

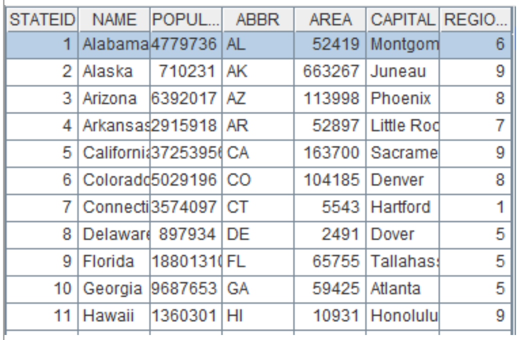
@i Connect on condition that the values are equal. By default, start connecting from the left. If no value corresponding to *F* is found, then remove this record.

Return value:

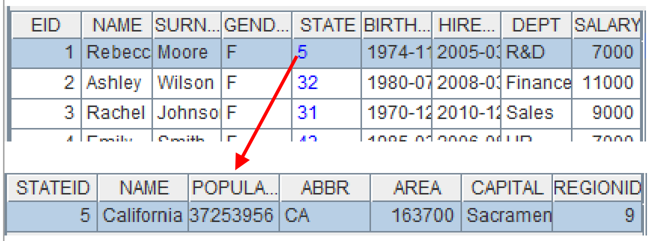
Returned cursor

Example:

	A
1	**=demo.cursor("select * from EMPLOYEE")**
2	**=demo.query("select * from STATES")**
3	**=A1.switch(STATE,A2:NAME)**
4	**=A3.fetch()**



STATEID	NAME	POPUL...	ABBR	AREA	CAPITAL	REGIO...
1	Alabama	4779736	AL	52419	Montgom	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
4	Arkansas	2915918	AR	52897	Little Roc	7
5	Californi	3725395	CA	163700	Sacrame	9
6	Colorado	5029196	CO	104185	Denver	8
7	Connecti	3574097	CT	5543	Hartford	1
8	Delaware	897934	DE	2491	Dover	5
9	Florida	1880131	FL	65755	Tallahas	5
10	Georgia	9687653	GA	59425	Atlanta	5
11	Hawaii	1360301	HI	10931	Honolulu	9



EID	NAME	SURN...	GEND...	STATE	BIRTH...	HIRE...	DEPT	SALARY
1	Rebecc	Moore	F	5	1974-1	2005-0	R&D	7000
2	Ashley	Wilson	F	32	1980-0	2008-0	Finance	11000
3	Rachel	Johnso	F	31	1970-1	2010-1	Sales	9000

STATEID	NAME	POPULA...	ABBR	AREA	CAPITAL	REGIONID
5	California	37253956	CA	163700	Sacramen	9

For any records of cursor corresponding to the "**STATE**" field in A1, convert them to the field values

Related concepts:

A.switch()

sync()

Description:

Node machine synchronization.

Syntax:

sync(*h*,*zs***)**

Remark:

From the current unit machine, synchronize the files on partition *zs* to the other unit machine *h*. The final time of the file shall prevail. Remove the extra file and correctly adjust the time for the file being synchronized. If *zs* is omitted, then all partitions will be synchronized.

Parameter:

h Sub-machine sequence.

zs Partition name.

Options:

@r From the unit machine *h*, synchronize with the current unit machine. If conflicts occur between files in *h*, then determine randomly. The conflicts here refer to as there are multiple unit machines and the number of files on the unit machines are different. Synchronize the files against the first unit machine when conflicts occur. Locate the latest file synchronization on other unit machines, and ignore the redundant files on other unit machines.

Example:

On the unit machine "**192.168.0.99:9282**", the contents in the sync.dfx cellset file is shown below. Set the cellset parameter arg1:

	A	
1	=sync("192.168.0.99:9281","1")	From the partition "1" of the current unit machine, synchronize the file to the partition "1" of the unit machine "**192.168.0.99:9281**"
2	=sync("192.168.0.99:9281")	From all partitions of the current unit machine, synchronize the file to all partitions of the unit machine "**192.168.0.99:9281**"
3	=sync@r("192.168.0.99:9281","1")	From the partition "1" of the unit machine "**192.168.0.99:9281**", synchronize the file to the partition "1" of the current unit machine

	A	
1	=callx("sync.dfx",1;"192.168.0.99:9282")	Call cellset file

system()

Description:

It is used to call the system command, and return the result once completed. For example, to open the **bat** and **exe** file.

Syntax:

system(*cmd/sh***)**

Remark:

It is presently only allowed to execute one *cmd command for each time. To execute cmd* commands for multiple times, you will need to write multiple *cmd* commands into a **bat** file, and use the *cmd* to call the **bat** file.

Parameter:

cmd cmd parameter style is set to “cmd /c +*command”***(Windows)**.

sh sh parameter style is set to “sh -c +*command”* **(Linux)**.

Option:

@p Proceed with the execution without pause

Return value:

Return the corresponding result

Example:

- **=system("cmd /C D:\\MicroInsight\\esCalc\\bin\\startup-zh.bat")**
- **=system("cmd /C D:\\MicroInsight\\DataLogic\\bin\\DataLogic.exe")**
- **=system("sh -c /test.sh")**

tan()

Description:

Compute the tangent value

Syntax:

tan(*number***)**

Remark:

The parameter *number* is in radians

Parameters:

number	Radians for which you want to compute the tangent value

Return value:

Float type

Example:

tan(pi()/2)	**1.633123935319537E16**
tan(pi(2))	**-2.4492935982947064E-16**

Related concepts:

sin()

cos()

time()

time(*datetimeExp*)

Description:

Get the time part from the datetime value

Syntax:

time(*datetimeExp***)**

Remarks:

Get the time part from *datetimeExp*, accurate to millisecond by default. The format must be consistent with the time format in the configure information. By default, the configure information will not be displayed in millisecond.

Parameters:

datetimeExp datetime

Options:

@m	Measure to minute
@s	Measure to second

Return value:

Time value

Example:

- time(now()) 16:28:26400
- time@s(now()) 16:28:260
- time@m(now()) 16:28:000

Related concepts:

date()

date(*datetimeExp*)

datetime(*datetimeExp*)

datetime()

time()

time()

Description:

Convert the string or integer to time data

Syntax:

time(*stringExp*)	Convert *stringExp* to time data type
time(*h,m,s*)	Convert *h,m,s* of integer type to time data type

Remark:

Convert the string *stringExp* or integer *h,m,s* to time data

Parameters:

stringExp A string in time format. The format must be the consistent with the one in the configure information.

h	integer
m	integer
s	integer

Return value:

Time data

Example:

- time("00:00:45") 00:00:45
- time(12,13,00) 12:13:00

Related concepts:

datetime()

date()

date(*datetimeExp*)
datetime(*datetimeExp*)
time(*datetimeExp*)

to()

A.to()

Description:

Get members from a sequence start from a specified position, so as to create a new sequence.

Syntax:

A.**to**(*a*) From sequence *A*, get a sequence composed of the first *a* members.

A.**to**(*a*,*b*) From the sequence *A*, get a sequence composed of the members from *a*th to the *b*th. If omitting *a*, then return 1 by default; If omitting *b*, then return ***A*.len()** by default. Please note that there is a comma that cannot be omitted.

Remark:

From the sequence *A*, get a sequence composed of the members from *a*th to the *b*th. If omitting *a*, then return 1 by default; If omitting *b*, then return ***A*.len()** by default.

Parameters:

A	A sequence
a	the starting integer
b	the ending integer

Return value:

Sequence

Example:

[1,5,2,6,8].to(2,3)	Return sequence **[5,2]**
[1,5,2,6,8].to(2)	Return sequence **[1,5]**

Related concepts:

to()

to()

Description:

Generate an integer sequence.

Syntax:

to(*a*,*b*) Generate an integer sequence composed of continuous integers between *a* and *b*.

to(*n*) Generate an integer sequence composed of continuous integers from **1** to *n*.

Remark:

Generate an integer sequence composed of a set of continuous integers from *a* to *b* or from 1 to *n*.

Parameters:

a	the starting integer
b	the ending integer

n *n*>0

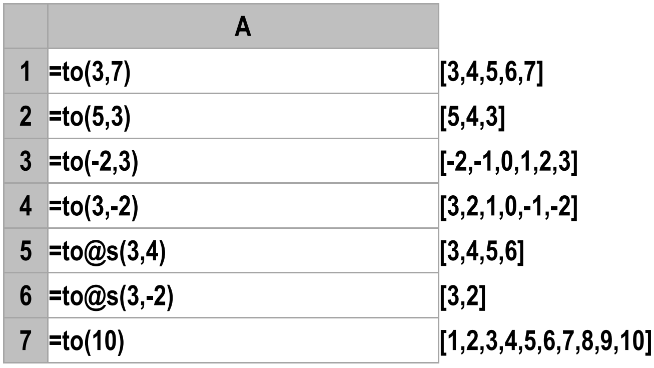
Options:

@s Generate a sequence composed of *b* integers continuously starting from *a*, If *b* is less than **0**, it is generated backward one by one in descending order.

Return value:

A continuous integer sequence

Example:



	A	
1	=to(3,7)	[3,4,5,6,7]
2	=to(5,3)	[5,4,3]
3	=to(-2,3)	[-2,-1,0,1,2,3]
4	=to(3,-2)	[3,2,1,0,-1,-2]
5	=to@s(3,4)	[3,4,5,6]
6	=to@s(3,-2)	[3,2]
7	=to(10)	[1,2,3,4,5,6,7,8,9,10]

Related concepts:

A.to()

top()

A.top()

Description:

Get the top n smallest records of sequence members

Syntax:

A.**top**(*x*,…;*n*)

Remark:

x is the expression, based on which compute for each member of a sequence. The records corresponding to the top n smallest values will be returned.

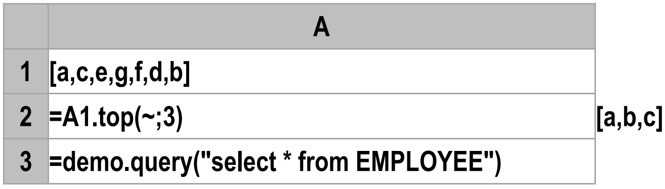
Parameter:

A	A sequence
x	Sort expression
n	Integer

Return value:

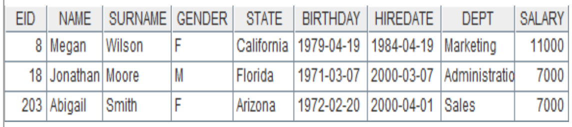
Corresponding records whose computational results of *x* are the top n smallest values

Example:



	A	
1	[a,c,e,g,f,d,b]	
2	=A1.top(~;3)	[a,b,c]
3	=demo.query("select * from EMPLOYEE")	

4	=A3.top(HIREDATE,SALARY;3)



EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
8	Megan	Wilson	F	California	1979-04-19	1984-04-19	Marketing	11000
18	Jonathan	Moore	M	Florida	1971-03-07	2000-03-07	Administratio	7000
203	Abigail	Smith	F	Arizona	1972-02-20	2000-04-01	Sales	7000

Get the data of top 3 employees who were employed at the earliest dates with still the lowest salary.

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.psort()

A.ptop()

topx()

A.topx()

Description:

Get the top *n* smallest values of the sequence.

Syntax:

A. **topx**(*x*;*n*)

Remark:

x is the expression based on which compute for each member of a sequence. Then, the resulting top n smallest values will be returned.

Parameter:

A	A sequence
x	Sort expression
n	Integer

Options:

@s	*x* is the sequence to compute the top n smallest values of members of *x*

Return value:

The top n smallest values resulting from the formula

Example:

	A	
1	[2,222,22,122,2222]	
2	=A1.topx(~*2;2)	[4,44]
3	=demo.query("select * from SCORES")	
4	=A3.topx(SCORE+10;3)	[61,61,62]
5	[[1,5],[4,6],[7,9,2]]	
6	=A5.topx@s(~;3)	[1,2,4]

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.psort()

A.ptop()

A.top()

trim()

Description:

Remove the space on both ends of a string

Syntax:

trim(*s*)

Remark:

Remove the space on both ends of a string *s*.

Parameters:

s Source string from which you want to remove the space

Options:

@l Remove the spaces on the left of the string *s*, and the option is letter **l**

@r Remove the spaces on the right of string *s*,

The default actions are to remove the space on both ends.

Return value:

String

Example:

–	**trim(" abc ")**	**"abc"**
–	**trim(" a bc ")**	**"a bc"**
–	**trim@l(" abc def ")**	**"abc def "**
–	**trim@l("def abc ")**	**"def abc "**
–	**trim@r(" abc def ")**	**" abc def"**
–	**trim@r("def abc ")**	**"def abc"**

true

Description:

Logical constants. True value

Syntax:

true

Remark:

It can be used directly in the constant cell or expression.

Example:

	A	B
1	=null	

Assign null to **A1**

2	=A1==null		Judge if **A1** is null
3	if A2==true	>a=4	If **A2** is true, then assign **4** to **a**
4	else	>a=3	Otherwise assign **3** to **a**

Related concepts:

null

false

union()

A.union()

Description:

Compute the union of members in a sequence whose members are sequences

Synatax:

A.union()

Remark:

Compute the union of members in sequence *A* whose members are sequences so as to get a sequence in which members won't appear repeatedly. Duplicate members in the same sub-sequence are regarded as independent ones

Parameters:

A A sequence whose members are sequences

Return value:

A new sequence created through the union of sequence *A*

Example:

	A	
1	=[[1,2,3,4,5],[3,7,8]].union()	**[1,2,3,4,5,7,8]** "3" only appear once
2	=[[1,2,2],[3,4,4],4].union()	**[1,2,2,3,4,4]** "4" only appear once. Duplicate members in the same sub-sequence are regarded as independent ones
3	=[[1,2,2],[2,2,2,3],2].union()	**[1,2,2,2,3]** There are three "2" in the second sub-sequence, so the final result also include three "2"

Related concepts:

A.conj()

A.diff()

A.isect()

A.union(*x*)

Description:

Compute *x* with each member of the sequence whose members are sequences, and then perform union operation on members of the new sequence

Synatax:

A.**union(x)**

Remark:

Compute *x* on sequence *A*, whose members are sequences, by loop, and then perform union operation on members of the resulting sequence

Parameters:

A A sequence whose members are sequences

x An expression that returns a sequence

Return value:

A new sequence created through the union of sequence *A*

Example:

	A	
1	=demo.query("select * from EMPLOYEE where GENDER = 'M' order by NAME")	
2	=demo.query("select * from EMPLOYEE where GENDER = 'F' order by NAME")	
3	=[A1,A2].union(~.(NAME))	**[Rebecca,Ashley,...]** Common members of A1 and A2 only appear once, and duplicate members in the same sub-sequence are regarded as independent ones

Related concepts:

A.union()

update()

db.update()

Description:

To update tables by a database connection.

Syntax:

db.**update**(*A*,*tbl*,F_i:x_i,…;*P*,...)

Remark:

Update F_i of the *tbl* table with the value of x_i, which is computed against *A*. If x_i is omitted, use the field of *A* whose name is the same as F_i instead.

Parameters:

db	Database connection
A	A sequence / record sequence
tbl	The name of a table in the database
F_i	A field in *tbl*
x_i	An expression which will be computed against *A* and then will be used as the new value of F_i.
P	The primary key of *tbl*. If omitted, these keys will be retrieved from *tbl*. If retrival is failed, then use *A*

Options:

@u	Generate UPDATE statement
@i	Generate INSERT statement

@a Clear the target table before the update is executed

@k After the execution is completed, the transaction won't be committed. If this option is omitted, the transaction will be committed.

@l The first field is an auto-incremental field with no corresponding formula

Return value:

None

Example:

After the update is completed, the transaction will be committed

	A	
1	=demo.query("select * from EMPLOYEE")	
2	>A1.modify(1,1:EID,"Aaron":NAME)	
3	>demo.update(A1, EMPLOYEE,EID:EID,NAME:NAME;EID)	**EID** is a primary key. The primary key fields must be included in the update fields.

upper()

Description:

Convert all characters to upper case

Syntax:

upper(*s***)**

Remark:

Convert all characters to upper case

Parameters:

s Source string to be converted to upper case

Return value:

String

Example:

- **upper("ABCdef")** **"ABCDEF"**
- **upper("abcDEF")** **"ABCDEF"**

Related concepts:

lower()

v.v()

Description:

Obtain the primary key value of a reference field.

Syntax:

v.**v()**

Remark:

If *v* is a record, the **pkey()** value of the record will be returned; otherwise, *v* itself will be returned.

This function is used to get the primary key value of a reference field. Because a reference field can switch between a referenced record and a primary key value by using the **switch()**, the **v()** function is used to distinguish whether there is a record or a primary key value.

v may be a record, a sequence or a number.

Parameters:

v a record, a sequence or a number.

Return value:

The primary key value of a reference field

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1(1).EID.v()	**1**; because the value of the **EID** field is a single number, the value itself is returned.
3	=demo.query("select * from DEPARTMENT ").primary(MANAGER)	
4	>A1.switch(DEPT,A3:DEPT)	Switch the **DEPT** field into a reference field pointing to the record of **DEPARTMENT**.
5	=A1(1).DEPT.v()	**2**; here, **DEPT** is a reference field whose value is record, therefore, the primary key value of the referenced record will be returned
6	>A3.primary(DEPT,MANAGER)	Set **DEPT** and **MANAGER** as the primary key of **A3**
7	=A1(1).DEPT.v()	**[R&D,2]**; return the value of the primary key of the record pointed by "**DEPT**". Because there are two primary fields, a sequence will be returned

Related concepts:

r.pkey()

T.primary()

variance()

A.variance()

Description:

Compute the variance value of the non-null members in a sequence

Syntax:

A.**variance()**

Remark:

Compute the variance value of the non-null members in sequence *A*. Note: This function doesn't apply to a sequence whose members are of non-numerical values

Parameters:

A *n* sequences

Return value:

The variance value of the non-null members in sequence *A*

Example:

	A	
1	=[1,2,3].variance()	0.66666

Related concepts:

A.sum()

A.avg()

A.min()

A.max()

A.count()

A.variance(x)

A.variance(*x*)

Description:

Compute *x* with each member of the sequence and then compute the variance value of the members of the new sequence

Syntax:

A.**variance**(*x*) Equivalent to *A*.(*x*).**variance()**

Remark:

Compute *x* on sequence *A* by loop and then compute the variance value of members in the resulting sequence

Parameters:

A A sequence

x Generally an expression of a single field name, or a legal expression composed of multiple field names. The computed result of the expression is of numerical value

Return value:

A numerical value

Example:

	A	
1	=demo.query("select * from EMPLOYEE")	
2	=A1.variance(SALARY)	Compute variance value of employees' salaries
3	=A1.(SALARY+100).variance()	Add 100 to the salary of each employee and then compute the variance value of all the employees' salaries

Related concepts:

A.variance()

words()

s.words()

Description:

Select the English words out of a string

Syntax:

s.**words()**

Remark:

Select the English words out of a string as a sequence of strings and retrun it; other characters will be ingnored.

Options:

@d	Select the numbers out of the string *s*
@a	Select both the English words and the numbers out of the string *s*

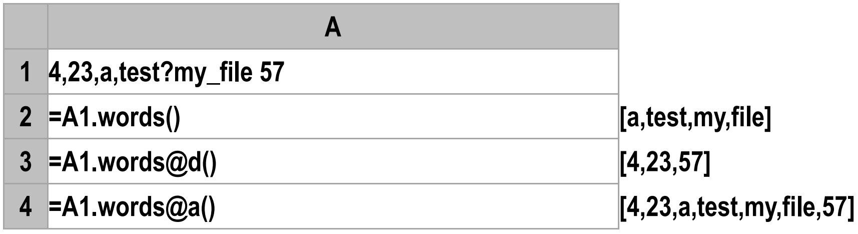
Parameters:

s	A string

Return value:

A sequence of strings

Example:



	A	
1	4,23,a,test?my_file 57	
2	=A1.words()	[a,test,my,file]
3	=A1.words@d()	[4,23,57]
4	=A1.words@a()	[4,23,a,test,my,file,57]

workday()

Description:

Compute a date time of n workdays from the specified date

Syntax:

workday (*t*,*k*,*h*)

Remark:

Compute a date of k workdays to the date t. The h is (not) a holiday sequence, that is, the member of h is either weekend or holidays. If it is weekend, then swap this day with a workday.

Parameters:

t	date
k	integer
h	time sequence

Return value:

Date time

Example:

- **workday(date("2011-11-07"),25,[date("2011-12-03"),date("2011-12-31")])** **2011-12-09**

– **workday(date("2011-11-07"),25,[date("2011-11-30"),date("2011-12-31")])** **2011-12-13**

write()

f.write()

Description:

Write string or a sequence into a file object. The write-in string can be binary data.

Syntax:

f.**write**(*s*)

f.**write**(*A*) Convert every member in the sequence *A* to a string and write to the file *f*; each member occupies one row.

Remark:

Write a file by overwrite; in other words, the original contents in the file *f* will be replaced with the string *s* or the string sequence *A*.

Parameters:

s	a string
f	a file object
A	a sequence

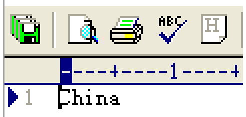
Options:

@a Append contents into a file, instead of overwriting. If contents exist in the file before appending, then new a line (carriage return) and append.

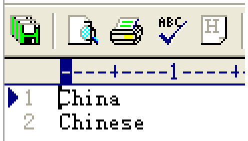
@b Write a binary file without adding the carriage return automatically

Example:

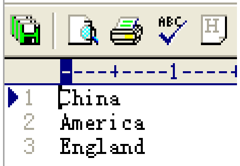
	A
1	=file("D:/tmp.txt")
2	>A1.write("China")
3	>A1.write@a("Chinese")
4	=["China","America","England "]
5	>A1.write(A4)
6	>A1.write(string(now())+":start")
7	>A1.write@a(string(now())+":end"))



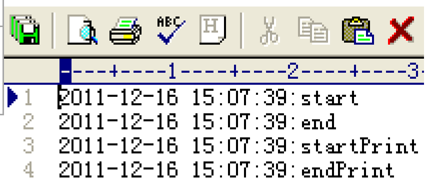
```
China
```



```
China
Chinese
```



```
China
America
England
```



```
2011-12-16 15:07:39:start
2011-12-16 15:07:39:end
2011-12-16 15:07:39:startPrint
2011-12-16 15:07:39:endPrint
```

Use the **write@a** function to compose a log

8	>A1.write@a(string(now())+":start Print")	
9	>A1.write@a(string(now())+":endPrint")	
10	=file("D:/test.txt")	A text file with binary data
11	=file("D:/result.png")	
12	=A10.read@b()	Read-in test.txt in binary format
13	=A11.write@b(A12)	Generate.png result in binary format

Related concepts:

f.read()

xjoin()

Description:

Cross-join multiple sequences together.

Syntax:

xjoin(A_i:F_i,x_i;…)

Remark:

Cross-join every member of sequence A_i unconditionally to create a new TSeq, which is composed of F_i, … fields. Each F_i will reference a member of the original sequence A_i. During the cross multiplication, select members satisfying criterion x_i from A_i.

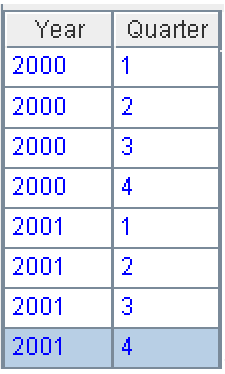
Parameters:

F_i	Field names of the resulting TSeq
A_i	The sequence on which cross-join is to be performed
x_i	Criterion of filtering expression

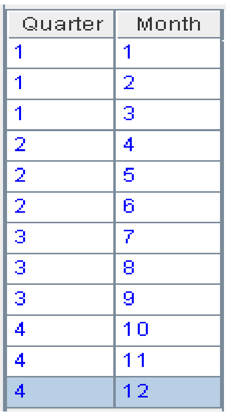
Return value:

A new TSeq, each member of which will reference a member of the original RSeq.

Example:

	A	
1	=create(Year).record([2000,2001])	
2	=create(QuarterID, QuarterNmae). record([1,"one",2,"two",3,"three",4,"four"])	
3	=xjoin(A1:Year;A2:Quarter)	 Cross-multiply the members together

Year	Quarter
2000	1
2000	2
2000	3
2000	4
2001	1
2001	2
2001	3
2001	4

4	=create(Month).record(to(12))		
5	=xjoin(A2:Quarter;A4:Month,Month>(Quarter. QuarterID-1)*3 && Month<=Quarter. QuarterID*3)		Dynamically compute the expression for filtering by cross multiplication.For the criterion of expression, you can refer to the current value of last field.

Quarter	Month
1	1
1	2
1	3
2	4
2	5
2	6
3	7
3	8
3	9
4	10
4	11
4	12

Note:

Difference between join() and xjoin():

join() join the sequences of A_i according to the criterion that the related fields/related expressions x_i is equal to x_1, and create a table sequence that has F_i,… as its fields. In this occasion, x_i is the related fields or the related expressions.

xjoin() cross-join the sequences of A_i unconditionally to create a table sequence that has F_i,… as its fields. It will remove records that cannot satisfy criterion x_i from each sequence of A_i during the cross multiplication. x_i is the filtering criterion.

Related concepts:

pjoin()

join()

year()

Description:

Get the year from a date

Syntax:

year(*dateExp*)

Remark:

Get the year from the date *dateExp*

Parameters:

dateExp	Expression whose result is a date or date time

Return value:

Integer

Example:

year(datetime("19800227","yyyyMMdd"))	1980
year("1972-11-08 10:20:30")	1972
year(datetime("2006-01-15 13:20:45"))	2006

Related concepts:

month()

day()

hour()

minute()

second()

millisecond()

Alignment Arithmetic Operation

Description:

Generate a new sequence by Alignment Arithmetic Operation between two sequences which are of the same length, such as aligning add, aligning subtract, aligning multiply and so on.

Syntax:

A++B	[A(1)+B(1),A(2)+B(2),...], Aligning add
A--B	[A(1)-B(1),A(2)-B(2),...], Aligning subtract
A**B	[A(1)*B(1),A(2)*B(2),...], Aligning multiply
A//B	[A(1)/B(1),A(2)/B(2),...], Aligning divide
A%%B	[A(1)%B(1),A(2)%B(2),...], Aligning division and get the remainder
A\\B	[A(1)\B(1),A(2)\B(2),...], Aligning division and get the integer value

Remark:

Alignment Arithmetic Operation means calculate the two members in the same position of *A* and *B* one by one, and thus generate new members for the new sequence. For example, *A*++*B* indicates [A(1)+B(1),A(2)+B(2),...].

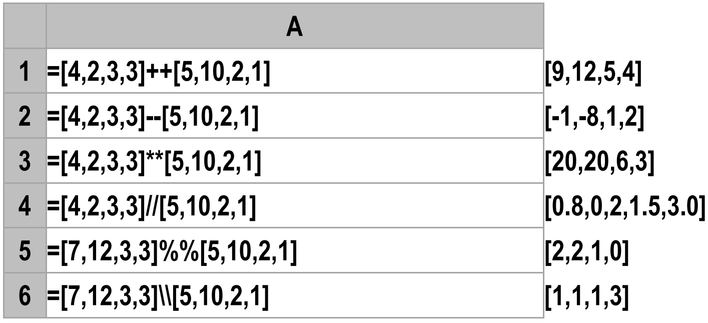
Parameters:

A	an *n* sequence
B	an *n* sequence

Return value:

A new sequence after the aligning computation

Example:



	A	
1	=[4,2,3,3]++[5,10,2,1]	[9,12,5,4]
2	=[4,2,3,3]--[5,10,2,1]	[-1,-8,1,2]
3	=[4,2,3,3]**[5,10,2,1]	[20,20,6,3]
4	=[4,2,3,3]//[5,10,2,1]	[0.8,0,2,1.5,3.0]
5	=[7,12,3,3]%%[5,10,2,1]	[2,2,1,0]
6	=[7,12,3,3]\\[5,10,2,1]	[1,1,1,3]

Related concepts:

Difference sequence

Intersection sequence

Union sequence

Multiply sequence

Concatenate sequence

cmp()

Arithmetic Operation

Description:

Perform the four arithmetic operations on two members.

Syntax:

x+y

x-y

x*y

x/y

Remark:

If both the first-progression computation (addition and subtraction) and the second-progression computation (multiplication and division) appear in the same formula, then the operational order will be first the multiplication and division, and then the addition and subtraction.

In case there is any bracket, firstly work out the factors inside the bracket and then those outside. For the same progression, work it out from left to right.

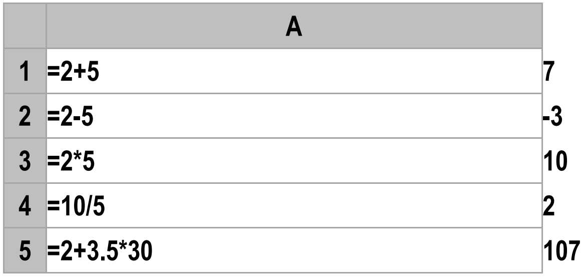
Parameters:

x	Numeric
y	Numeric

Return value:

Numeric

Example:



	A	
1	=2+5	7
2	=2-5	-3
3	=2*5	10
4	=10/5	2
5	=2+3.5*30	107

Related concepts:

cmp()

Assignment

Description:

Assign value to a member of a sequence.

Syntax:

A(*i*)=*x* Assign the *i*th member of sequence *A* with the value *x*. If *i* exceeds the number of members of sequence *A*, then an error will be raised.

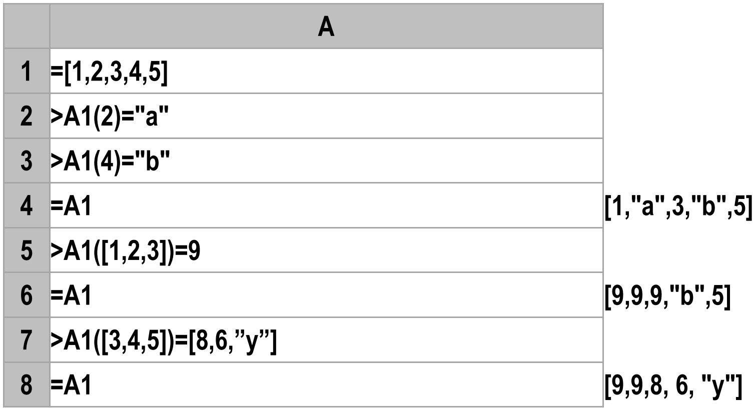
A(*p*)=*x* Assign all members of ISeq *p* with the value *x*. If the member of *p* exceeds the number of members of sequence *A*, then an error will be raised.

A(*p*)=X Respectively and correspondingly assign every members of ISeq *p* with the data from ISeq *X*. The length of ISeq must be the same, or an error will be raised.

Parameters:

A	Sequence
i	Sequence number of the member in the sequence
x	Data you are about to assign to the member
p	Sequence, which is composed of sequence number
X	ISeq, of which the data is to be assigned to the members

Example:



	A	
1	=[1,2,3,4,5]	
2	>A1(2)="a"	
3	>A1(4)="b"	
4	=A1	[1,"a",3,"b",5]
5	>A1([1,2,3])=9	
6	=A1	[9,9,9,"b",5]
7	>A1([3,4,5])=[8,6,"y"]	
8	=A1	[9,9,8, 6, "y"]

Batch computation

Description:

To compute a series of expressions one by one in an automated fashion, and return the result of the last expression.

Syntax:

(x_1,x_2,…,x_k)

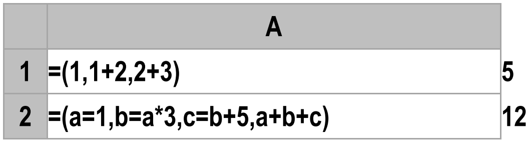
Remark:

The later expressions can refer to the variable value computed and assigned by the preceding expressions.

Parameters:

x_k	Expressions you want to compute in a batch

Example:



	A	
1	=(1,1+2,2+3)	5
2	=(a=1,b=a*3,c=b+5,a+b+c)	12

Comparison operation

Description:

The comparison operation between two numerical values, sequences, or characters

Syntax:

x==*y*	If values of two operands are the same, then the result is true. Otherwise, it is false.
x! =*y*	If values of two operands are not the same, then the result is true. Otherwise, it is false.

x<*y* If x is less than y, then the result is true. Otherwise, it is false.

x>*y* If x is greater than y, then the result is true. Otherwise, it is false.

x<=*y* If x is not greater than y, then the result is true. Otherwise, it is false.

x>=*y* If x is not less than y, then the result is true. Otherwise, it is false.

Remark:

The comparison operation requires the numeric or character operands, and the result is the boolean value. For those Not Less Than (or Not Greater Than) comparison operators, the result value is false only when both the Greater and Equal relations are not true, and the result is still true as long as either of the relations is true.

If *x* and *y* are sequences, then *x* and *y* will be compared in one-to-one relation following the sequential order of members. Return the result according to the first unequal align comparison of members, return true if the conditions are met. Otherwise, return false. However, for the align comparison ==, the true will only be returned when all members are equal in the align comparison. Otherwise, return false. If the numbers of members in sequence *x* and *y* are not the same, and their beginning members are the same, then the value wth less members will be smaller.

Parameter:

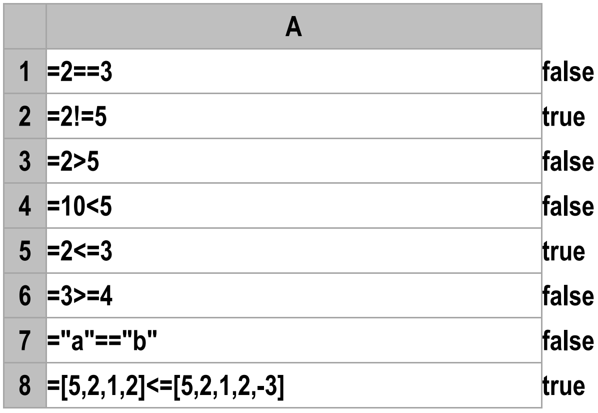
x Sequence, numerical, or character data

y Sequence, numerical, or character data

Return value:

true/false

Example:



	A	
1	=2==3	false
2	=2!=5	true
3	=2>5	false
4	=10<5	false
5	=2<=3	true
6	=3>=4	false
7	="a"=="b"	false
8	=[5,2,1,2]<=[5,2,1,2,-3]	true

Compound assignment

Description:

The value of a certain variable will undergo a certain computation with an expression to assign to the variable itself.

Syntax:

a?=*x*

Remark:

A variable value will undergo a certain computation with an expression to assign to the variable itself.

Parameters:

a Variable name

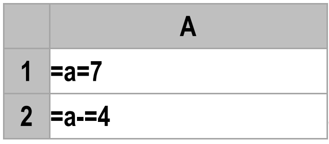
x Legal expression

? Operators, support the operator +, -, *, /, \, and %

Return value:

New variable

Example:



	A	
1	=a=7	
2	=a-=4	3

Concatenate sequence

Description:

Concatenate two sequences so as to generate a new sequence.

Syntax:

A|*B*

Remark:

Concatenate the members (or single values) of two sequences in proper order to compose a new sequence, that is, [*A*(1),...,*A*(*n*),*B*(1),…,*B*(*m*)].

Parameters:

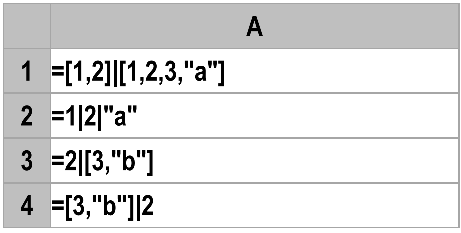
A an *n* sequence or a single value; When it is a single value, it is regarded as [*A*]

B an *m* sequence or a single value; When it is a single value, it is regarded as [*B*]

Return value:

The new sequence after concatenating the two sequences *A* and *B*

Example:



	A	
1	=[1,2]|[1,2,3,"a"]	[1,2,1,2,3,a], the same member appears repeatedly
2	=1|2|"a"	[1,2,a], the single value can also be concatenated
3	=2|[3,"b"]	[2,3,b]
4	=[3,"b"]|2	[3,b,2]

Related concepts:

Difference sequence

Intersection sequence

Union sequence

Multiply sequence

Alignment Arithmetic Operation

cmp()

Difference sequence

Description:

Generate a new sequence by subtracting members from a sequence.

Syntax:

*A**B*

*A**x*

Remark:

Generate a new sequence by subtracting the members (or single values) of *B* from sequence *A* in proper order

Parameters:

A	an *n* sequence
B	an *m* sequence or a single value; when it is a single value, it is regarded as **[***B***]**

Return value:

The new sequence by subtracting the members (or single values) of *B* from sequence *A* in proper order.

Example:

	A	
1	=[1,2,3,4]\[2,3]	[1,4]
2	=[1,2,3,3]\[1,3]	[2,3], the same members will not be subtracted repeatedly
3	=[1,2,3]\3	[1,2]

Related concepts:

Concatenate sequence

Intersection sequence

Union sequence

Multiply sequence

Alignment Arithmetic Operation

cmp()

Empty sequence

Description:

A sequence having no member.

Remark:

A sequence having no member, for example: **[]**.

Escape character

Description:

A character that invokes an alternative interpretation on special characters in a string.

Syntax:

\*s*

Remark:

The special character such as \ (not an escape character, simply a backslash), **"**, **'**, newlines, etc., needs an alternative interpretation in a string, or it will be misunderstood.

The double quotation mark in **$[***string***]** does not need to use the escape character; while the double quotation mark in **"***string***"** needs to use the escape character.

We support the following escape character:

\t

\r

\n

Parameters:

s	The special character that needs an alternative interpretation such as \ (not an escape character, simply a backslash), **"**, **'**, newlines, etc.

Return value:

A string that contains the corresponding special character converted from the escape sequences.

Example:

Common examples:

– **"\\\\"**	backslash
– **"\\n"**	newlines

Double quote in **$[]** does not need an escape character, while double quote in **""** does.

– **$[a"s]**

– **"a\\"s"**

Related concepts:

String

Hexadecimal long integer

Description:

Hexadecimal long integer

Syntax:

0x12

Remark:

Those starting with “0x” are the hexadecimal long integers

Example:

– **0x2345**	**9029**

Related concepts:

Long integer

Identifier

Description:

Identifier (commonly referred to as variable name) is a defined identifier or defined variable name.

Syntax:

string1

'*string2*'

Remark:

Normally the variable name can be used directly without having to be defined specially. But if the variable name contains the space, equal sign, and other misleading characters, then the variable name will need to be defined specially. This is same to the principle of handling space in DOS command.

Parameters:

string1	Ordinary identifier
'string2'	If the string contains any misleading characters such as **=**, space, **+**, and **-** , the string must be enclosed in single quotation marks.

Return value:

Variable using *string1* or '*string2*' as variable name.

Example:

	A	
1	>arg1=5	Common identifier
2	>'a b'=4	Identifier that contains a space
3	>'a=b'=3	Identifier that contains equal signs
4	=arg1	5
5	='a b'	4
6	='a=b'	3

Intersection sequence

Description:

Generate a new sequence which is composed of common members from two sequences.

Syntax:

A^*B*

Remark:

Generate a new sequence which is composed of members both in *A* and *B*

Parameters:

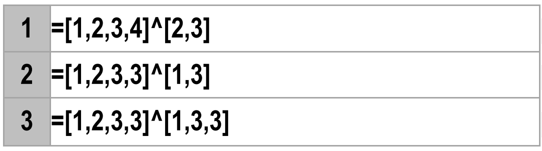
A	an *n* sequence
B	an *m* sequence

Return value:

The new sequence which is composed of members both in *A* and *B*

Example:

	A



1	=[1,2,3,4]^[2,3]	**[2,3]**
2	=[1,2,3,3]^[1,3]	**[1,3]**, there is only a **3** in the common members
3	=[1,2,3,3]^[1,3,3]	**[1,3,3]**, there are two **3** in the common members

Related concepts:

Concatenate sequence

Union sequence

Multiply sequence

Alignment Arithmetic Operation

cmp()

Logic operation

Description:

Perform logical operations on the two boolean expressions

Syntax:

x&&*y* Logical AND; If both x and y are true, then the result is true. Otherwise, it is false. As long as value on the left end to the operator is false, the final result will always be false, no matter the value on the right end to the operator is true or false.

x||*y* Logical OR; The result is true as long as either x or y is true. Otherwise, it is false. As long as the value on the left end to the operator is true, the final result is always the true, no matter the value on the right end to the operator is true or false.

!*x* Logical NOT, the reverse value of the original value.

Remark:

The operand of logic operation is Boolean. If the operand is not the Boolean, then it will be converted to the Boolean. The result is a Boolean value.

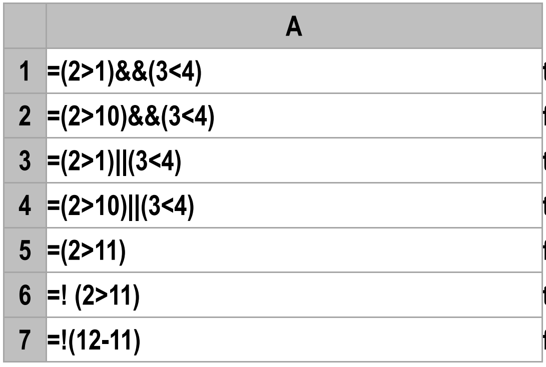
Parameter:

x	expression
y	expression

Return value:

true/false

Example:



	A	
1	=(2>1)&&(3<4)	true
2	=(2>10)&&(3<4)	false
3	=(2>1)||(3<4)	true
4	=(2>10)||(3<4)	true
5	=(2>11)	false
6	=! (2>11)	true
7	=!(12-11)	false

Long integer

Description:

Long integer

Syntax:

1L

Remark:

A long integer is represented by an integer with a tailing capital L.

Example:

- **2345L** **2345**

Related concepts:

Hexadecimal long integer

Loop Function

Description:

The introduction of common rules of loop functions.

Syntax:

A.*f*(*x*)

Remark:

Compute *A*(1).*f*(*x*),...,*A*(*n*).*f*(*x*),~=*A*(*i*) in proper order against each record of record sequence *A*, "~" in *x* is used to reference the current record *A*(*i*)

Parameters:

A	an *n* record sequence
x	an expression, "~" in which is used to reference the current record *A*(*i*).
f	a loop function, which includes aggregate function, locate function, select function, modify function and so on, most of which can be operated on a record sequence.

Return value:

The computation of the function *f*

Example:

	A	
1	=demo.query("select EID,NAME,BIRTHDAY,SALARY,1 as AGE from EMPLOYEE")	
2	=A1.run(AGE=age(BIRTHDAY))	Compute the age of the employees, and assign the result to the field **AGE**.
3	=A1.avg(AGE)	35.52

Modulus

Description:

Perform Mod operation on two Integer numbers or seek the integer value.

Syntax:

x%*y*	Get remainder
x\y	Seek integral value

Remark:

Two integers or long integers division will generate the integer and the remainder parts.

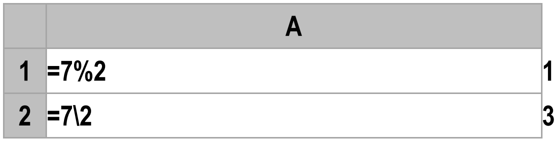
Parameters:

x	Integer or Long integer
y	Integer or Long integer

Return value:

Integer

Example:



	A	
1	=7%2	1
2	=7\2	3

Related concepts:

cmp()

Multiply sequence

Description:

Generate a new sequence by duplicating members of a sequence.

Syntax:

*A***k*   or

*k***A*

Remark:

Generate a new sequence by duplicating the members of the sequence *A* for *k* times

If *k*<=0, then generate an empty sequence.

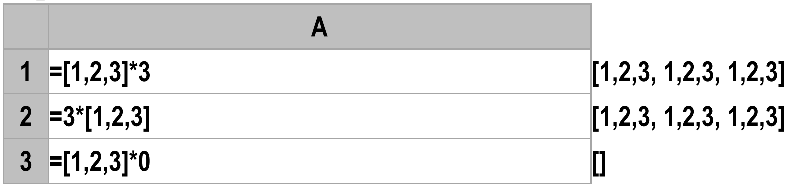
Parameters:

A	a sequence
k	an integer

Return value:

The new sequence generated by duplicating the members of the sequence *A* for *k* times.

Example:



	A	
1	=[1,2,3]*3	[1,2,3, 1,2,3, 1,2,3]
2	=3*[1,2,3]	[1,2,3, 1,2,3, 1,2,3]
3	=[1,2,3]*0	[]

Related concepts:

Difference sequence

Union sequence

Concatenate sequence

Alignment Arithmetic Operation

cmp()

Opposite number

Description:

Opposite number

Syntax:

-*a*

Remark:

Opposite number. If *a* is the date time and character string, then it can be used to sort in descending.Functions frequently used in sorting in ascending order include *cs*.sortx()、*A*.sort(*x* ;loc)、*A*.psort (*x*)、*A*.top(*x*,…;*n*), and etc. By default, they can only be used to sort in ascending order, though the sign "-" can be used to invert the fields to be sorted. And sorting the inverted fields ascendingly means sorting the original fields descendingly.

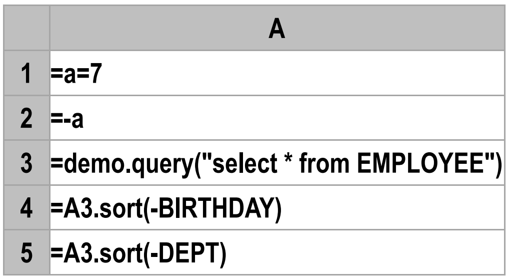
Parameters:

a variable name , data time or string

Return value:

Real number or sequence

Example:



	A	
1	=a=7	
2	=-a	-7
3	=demo.query("select * from EMPLOYEE")	
4	=A3.sort(-BIRTHDAY)	Sort by birthday descendingly
5	=A3.sort(-DEPT)	Sort by character string descendingly

String

Description:

Define a string constant.

Syntax:

"*string*" or

$[*string*]

Remark:

The expression must be double quoted when using. But the quotation mark is not required if the character string constants are defined directly.

The macro in **"***string***"** will not be replaced. When copying/pasting/inserting/deleting the row, the cell name in **"***string***"** will not change automatically.

The macro in **$[***string***]** will be replaced. When copying/pasting/inserting/deleting the row, the cell name in **$[***string***]** will change automatically.

The double quotation mark in **$[** *string***]** does not need to use the escape character; while the double quotation mark in **"***string***"** needs to use the escape character.

Parameters:

string Content of the string. Content can be any character.

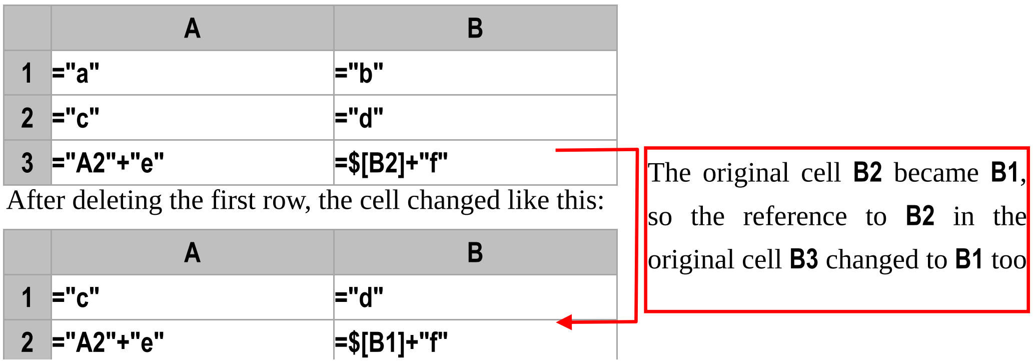
Return value:

String constant

Example:

- **"asd"**
- **$[asd]**

In the string enclosed in **$[]**, the cell name will change automatically when copying/pasting/inserting/deleting the row, while the string enclosed in **""**, the cell name will not change automatically when copying/ pasting/inserting/deleting the row.



	A	B
1	="a"	="b"
2	="c"	="d"
3	="A2"+"e"	=$[B2]+"f"

After deleting the first row, the cell changed like this:

	A	B
1	="c"	="d"
2	="A2"+"e"	=$[B1]+"f"

The original cell **B2** became **B1**, so the reference to **B2** in the original cell **B3** changed to **B1** too

The macro in **$[]** will be replaced automatically, while macro in **""** will not be replaced automatically.

- **"${arg1}abc"** **"${arg1}abc "**
- **$[${arg1}abc]** **"Tomabc"**, (the value of **arg1** is **"Tom"**)

The double quotation mark in **$[]** does not need to use the escape character; while the double quotation mark in **""** needs to use the escape character.

- **$[a"s]**
- **"a\"s"**

Related concepts:

Escape character

String concatenation

String concatenation

Description:

Join two or more strings end-to-end.

Syntax:

x+*y*

Remark:

A string and a numeric value cannot be concatenated.

Parameters:

x	a string constant
y	a string constant

Return value:

A concatenated string formed by joining *x* and *y*.

Example:

–	"abc"+"def"	abcdef
–	"abc"+123	123. A string and a numeric value cannot be concatenated.

The writing rules of the expressions *x* in a loop function

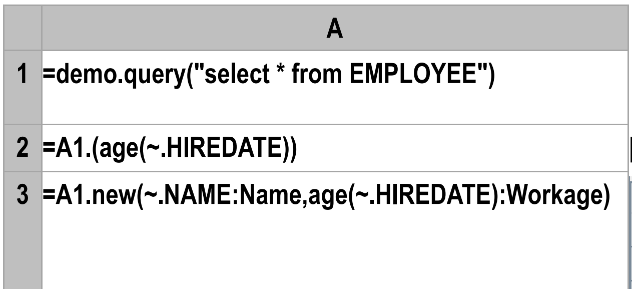
Description:

Introduce the common rules of expressions in a loop function.

Remark:

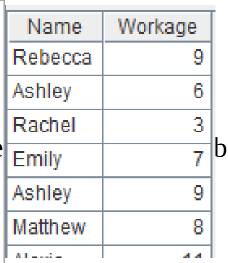
Expressions *x* in loop functions such as *r*.(*x*), *r*.run(*x*), *A*.(*x*), *A*.run(*x*) and so on, should follow the rules below:

- Basic members: ~ and *A*.~, ~ means the record being processed currently.



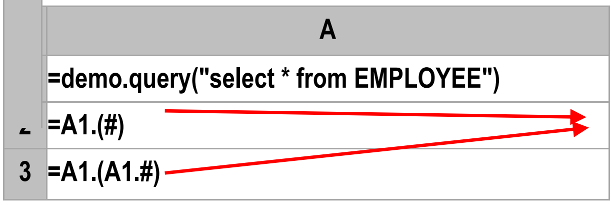
	A
1	=demo.query("select * from EMPLOYEE")
2	=A1.(age(~.HIREDATE))
3	=A1.new(~.NAME:Name,age(~.HIREDATE):Workage)

[9,6,3,7,9,8,…]



Name	Workage
Rebecca	9
Ashley	6
Rachel	3
Emily	7
Ashley	9
Matthew	8

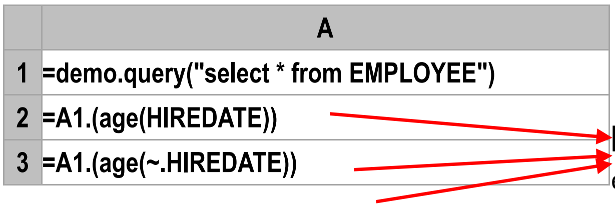
Sequence numbers: # and *A*.#, # indicates the seque basic members.



	A
	=demo.query("select * from EMPLOYEE")
2	=A1.(#)
3	=A1.(A1.#)

[1,2,3,4,5,6,7,8…], the two results are the same

- Fields: *F*, ~.*F*, *r*.*F* and *A*.*F* equal to *A*.~.*F* , and *A*.*F* equal to *A*(1).*F* if there is not a loop.



	A
1	=demo.query("select * from EMPLOYEE")
2	=A1.(age(HIREDATE))
3	=A1.(age(~.HIREDATE))

[9,6,3,7,9,8,…], the results of the tree expressions are the same.

4	**=A1.(age(A1.HIREDATE))**	
5	**=A1.HIREDATE**	**2005-03-11**, it is equal to **A1(1).HIREDATE** if there is not a loop.

- *A*[*i*] and **~**[*i*] equal to *A*(**#+***i*) in the loop, that is to count the members forward from the current member in *A* and return the ith member, if the boundary is exceeded, errors are not reported and null is returned. *F*[*i*] equal to *A*[*i*].*F*

	A	
1	**=demo.query("select ORDERID,AMOUNT," as ACCUMULATION from SALES")**	
2	**=A1.run(ACCUMULATION=AMOUNT+ACCUMULATION[-1])**	Obtain the **ACCUMULATION** of the current record by adding the **ACCUMULATION** of the last record and the current **AMOUNT**.

- *A*{*a*,*b*} and **~**{*a*,*b*} in the loop indicates a sequence composed of members between *A*(**#+***a*) and *A*(**#+***b*). *a* is **1-#** by default and *b* is *A*.**len()-#** by default.

	A	
1	**=demo.query("select STOCKID,DATE,CLOSING," as FirstThree from STOCKRECORDS")**	
2	**=A1.run(FIRSTTHREE=~{-3,-1}.(CLOSING).avg())**	**~{-3,-1}** indicates a sequence composed of three records counted backward from the current row.

- Iteration: *A1.f1*(*A2.f2*(*x*))

  The symbols in *x* are first explained to be subordinate to *A2*.

- Aggregate: *A*.*f*(*x*)

  A.*f*(*x*) equal to *A*.(*x*).*f*(), *f* in which is an aggregate function and *x* can be null.

Note:

The *x* in *A*.**count**(*x*) is a boolean expression, which is different from other aggregate function rules.

Union sequence

Description:

Generate a new sequence by merging two sequences.

Syntax:

*A***&***B*

Remark:

Generate a new sequence by merging the members (or single values) from the two sequences *A* and *B* in proper order. The common members will not appear repeatedly in the new sequence.

Parameters:

A	an *n* sequence or a single value; When it is a single value, it is regarded as [*A*]
B	an *m* sequence or a single value; When it is a single value, it is regarded as [*B*]

Return value:

The new sequence after merging the two sequences *A* and *B*

Example:

	A	
1	=[1,2,3,4]&[3,2]	[1,2,3,4]. If the members of the two sequences are not sorted in the same order, then follow the member orders of the first sequence
2	=[1,2,3,3]&[1,3]	[1,2,3,3]. The same members will not appear repeatedly, but the same members in the former sequence itself will not be deleted
3	=[1,2,3,3]&4	[1,2,3,3,4]
4	=4&[1,2,3,3]	[4,1,2,3,3]
5	=33&4	[33,4]

Related concepts:

Difference sequence

Intersection sequence

Concatenate sequence

Multiply sequence

Alignment Arithmetic Operation

cmp()

Value assignment and computation

Description:

Assign the result of an expression to a variable and return the result of the expression.

Syntax:

a=*x*

Remark:

Assign the result of expression *x* to variable *a* and return the result of the expression *x*.

Parameters:

a	The variable name
x	The valid expression

Return value:

The result of an expression

Example:

	A	
1	>time=now()	
2	>arg1=5*3	
3	=time	The current time
4	=arg1	15

Chart Element

G.draw()

Description:

Draw canvas

Syntax:

G.**draw**(*w*,*h*)

Remark:

Draw a *w*(width) * *h* (height) canvas. The format of image returned can be specified in options, and the default format is SVG.

Parameters:

G	Canvas
w	Width of canvas in pixels
h	Height of canvas in pixels

Options:

@j	Return a JPG canvas
@g	Return a GIF canvas
@p	Return a PNG canvas

Return value:

Canvas drawing

Example:

= a.draw@p(600,600)	Return a PNG canvas of 600 pixels wide and high

G.plot()

Description:

Compute the chart plotting string and its parameters, plot them onto the canvas, and return.

Syntax:

G.**plot**(e,a_i:x_i:A_i,…)

Remark:

Compute the plotting string on the canvas *G*, with the chart element *e* to plot. Assign the chart element parameter a_i with the value x_i. If the corresponding coordinate axis is available for this parameter value, then this coordinate axis is A_i.

Parameter:

G	Canvas
e	Chart element, for example, MapAxis, NumericAxis, EnumAxis, DateAxis, BackGround, Column, Line, Dot, Polygon, Text, Legend and Sector.
a_i	Parameters of chart element, for example, the name, logicData, and physicalData parameters

in the chart element of MapAxis. For the parameter introduction of each chart element, please refer to parameter chapter.

x_i parameter value

A_i Coordinate axis corresponding to the parameter value

Return value:

Chart element

Example:

```
=a.plot("Legend","legendText":A1,"legendFillColor":A1:"map1")          Legend
```

Chart Element Parameters

MapAxis

name

Name of mapping table

logicalData

Indicates the sequence is composed of logical coordinate’s value of a logical axis.

physicalData

The mapping data values corresponding to the **logicalData**, which are in a one-to-one relation to the **physicalData**. The mapping values are usually in the form of a sequence composed of the appearance properties.

NumericAxis

name

Numeric axis name

visible

Show or hide axis

location

The numeric axis will appear on here. You can set the axis as vertical axis, horizontal axis, polar axis, or angle axis.

is3D

Indicates the 3-dimensional axis. If the 3D properties is adopted for at least one coordinate axis in the coordinate system on canvas, then the other coordinate axises in the current coordinate system will take on the 3D effects consistently in chart plotting.

threeDThickRatio

Indicates the thickness ratio of 3D axis when the axises in the chart have taken on the 3D effect. This parameter will determine the **3D Thickness**, which refers to the length of forward and backward edges. The threeDThickRatio takes effect **on condition that** there is no enum axis in the coordinate system on canvas, and the axis 1 is the numeric axis. The default maximum 3D depth is 60 pixels.

To compute the overall 3D thickness based on the 3D thickness radio of axis 1, **3D Thickness Ratio * Canvas Height** will be used. If the ratio is greater than 1, then it represents the absolute pixel value. In addition, the chart will be auto-plotted with 60 pixels if the 3D thickness is greater than 60.

autoCalcValueRange

The value range of the coordinate axis will be computed automatically according to the coordinate point. If setting to true, in the value range of numeric axis, the computation range will be set automatically according to the coordinate point on the coordinate axis.

maxValue

Maximum on the customized numeric axis. In this case, the autoCalcValueRange must be set to false, and the system is not required to compute the axis scope automatically.

minValue

Minimum on the customized numeric axis. In this case, the autoCalcValueRange must be set to false, and the system is not required to compute the scope of axis.

scaleNum

The number of scales displayed on the customized numeric axis. The autoCalcValueRange must be set to false, and the scope of computation is not required to compute automatically.

format

Set the display format of the scale and label values manually, for example, ￥0.00, and $0.0.

transform

The transformation type of scales on the numeric axis. Options include not-to-transform, scale, logarithm, and index.

scale

Customize the zooming scales for numeric axis when the transformation type is scale. The default value is 1.0.

logBase

Customize the logarithmic base of the numeric axis if the transformation type is logarithm. The default value is 10.

powerExpoment

Customize the power exponent if the transformation type is exponent. The default value is 2.718.

unitFont

Indicates the font of displayed unit text in the value transformation after setting the type of transformation.

unitStyle

Sets the style of displayed text for unit. The options include the boldness, italic type, underline, and vertical text. The vertical text settings will override the underlines if setting them both.

unitSize

Size of displayed unit text

unitAngle

Set rotation angle of unit text

unitColor

Color of unit text

xStart

The position of starting point coordinate on X-axis of the canvas when the numeric axis is a horizontal axis.

If the specified value is not greater than 1 and not less than 0, it indicates the chart will take up this percentage widths/lengths across the canvas. If the specified value is greater than 1 or less than 0, then it represents the pixel values. The chart will expand across the border if the pixels are greater than the width/length of canvas.

The origin point is in the top left corner if the specified value is not less than 0; Otherwise, it will be located in the bottom right corner if the specified value is less than 0.

xEnd

The position of ending point coordinate on X-axis of the canvas when the numeric axis is a horizontal axis. Please refer to the xStart property description for the property value settings.

xPositon

The position of coordinate point on Y-axis of the canvas when the numeric axis is a horizontal axis. Please refer to the xStart property description for the property value settings.

yStart

The position of starting point coordinate on Y-axis of the canvas when the numeric axis is a vertical axis. The numeric value ascends from top to bottom. Please refer to the xStart property description for the property value settings.

yEnd

The position of ending point coordinate on Y-axis of the canvas when the numeric axis is a vertical axis. Please refer to the xStart property description for the property value settings.

yPositon

The position of coordinate point on X-axis of the canvas when the numeric axis is a vertical axis. Please refer to the description on the value forms of the xStart property for the property value settings.

polarX

The position of starting point coordinate on X-axis of the canvas when the numeric axis is a polar axis. Please refer to the description on the value forms of the xStart property for the property value settings.

polarY

The position of starting point coordinate on Y-axis of the canvas when the numeric axis is a polar axis. Please refer to the description on the value forms of the xStart property for the property value settings.

polarLength

Sets the polar axis length.

startAngle

Sets the start angle when the numeric axis is an angle axis. In plotting the circular sector pie chart, the circular sector can be customized with the startAngle and endAngle properties.

endAngle

Sets the end angle of an angle axis.

axisColor

Sets the axis line color of a numeric axis.

axisLineStyle

Sets the style of axis line. Options include the dotted line, solid line, and none.

axisLineWeight

The line weight of numeric axis.

axisArrow

Sets the arrow style of numeric axis.

title

The title name of numeric axis.

titleFont

The font of title of numeric axis.

titleStyle

The title style of numeric axis. Please refer to unitStyle.

titleSize

The title font size of numeric axis.

titleIndent

The blanks in the title of numeric axis.

titleColor

The title font color of numeric axis.

titleAngle

The title rotation angle of numeric axis.

allowLabels

Indicates if the label value will be displayed.

labelFont

The font of label value.

labelStyle

The label value style. Please refer to the unitStyle of NumericAxis.

labelSize

The font size of label value.

labelIndent

The blank in-between the label value and the axis.

labelColor

The font color of label value.

labelStep

The display style of label. The labelStep=0 indicates that all labels will be displayed, and labelStep=2 indicates a label value will be displayed every other 2 labels.

labelAngle

The rotation angle of label value

labelOverlapping

The intersecting of label. If checked, the intersecting label is allowed; Otherwise, the intersecting label is not allowed, and the intersecting label, if any, will be overridden and hidden.

scalePosition

The tick mark can be displayed on the right or top, on the left or bottom, or across axis, or none tick marks appear. Once the label is positioned, the tick mark will change accordingly.

scaleStyle

The style of tick mark, including the dotted line or the solid line.

scaleWeight

The weight of tick mark.

scaleLength

The length of tick mark.

displayStep

The display style of tick mark. If displayStep=0, then every tick mark will be displayed, and displayStep=2 indicates it becomes visible every other two.

allowRegions

Indicates if the spacer region will be displayed.

regionLineStyle

The type of spacing line in the spacer region.

regionLineColor

Color of spacing line displayed in the spacer region

regionLineWeight

Indicates the weight of spacing line displayed in the spacer region

regionColors

Color of spacer region

regionTransparent

Transparency of spacer region. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

isPolygonalRegion

Indicates whether it is a polygon spacer. This property is only valid in the polar coordinate system, and is fit for radar chart and other no-filling charts. For pie chart and other charts with fillings, such effect is not vivid. On the radar map, the shape of spacer region will change along with the number of categories.

EnumAxis

name

Enum axis name

visible

Show or hide axis

location

Enum axis will appear on here. You can set the axis as vertical axis, horizontal axis, polar axis, or angle axis.

is3D

Indicates the 3-dimensional axis. If the 3D properties is adopted for at least one coordinate axis in the coordinate system on canvas, then the other coordinate axises in the current coordinate system will take on the 3D effects consistently in chart plotting.

threeDThickRatio

Indicates the thickness ratio of 3D axis when the axises in the chart have taken on the 3D effect. This parameter will determine the **3D Thickness**, which refers to the length of forward and backward edges. The default maximum 3D thickness is 60 pixels.

To compute the overall 3D thickness based on the 3D thickness radio of the enum axis, **3D Thickness Ratio * Enum Axis Series Width** will be used. The chart will be auto-plotted with 60 pixels if the 3D thickness is greater than 60.

categories

Specifies the categories. If not specified, then select from the cross sequence of the series of categories. For example, the cross sequence =["Tom, maths","Tom, English","Tom, French","Mary, Maths","Mary,English","Mary,French"], and the ["Tom","Mary"] is auto-retrieved as the category value.

series

Specifies the series. If not specified, then select from the cross sequence of the series of categories. For example, the cross sequence =["Tom, maths","Tom, English","Tom, French","Mary, Maths","Mary,English","Mary,French"], and the ["Maths","English","French"] is auto-retrieved as the series value.

gapRatio

Represents the rate of series gap width on the category axis to the category gap, starting gap, or ending gap. By default, this value is 1.5, indicating the gap width is the 1.5 times of the basic width of series.

xStart

The position of starting point coordinate on X-axis of the canvas when the enum axis is a horizontal axis.

If the specified value is not greater than 1 and not less than 0, it indicates the chart will take up this percentage widths/lengths across the canvas. If the specified value is greater than 1 or less than 0, then it represents the pixel values. The chart will expand across the border if the pixels are greater than the width/length of canvas.

The origin point is in the top left corner if the specified value is not less than 0; Otherwise, it will be located in the bottom right corner if the specified value is less than 0.

xEnd

The position of ending point coordinate on X-axis of the canvas when the enum axis is a horizontal axis. Please refer to the xStart property description for the property value settings.

xPosition

The position of coordinate point on Y-axis of the canvas when the enum axis is the horizontal axis. Please refer to the description on the xStart property for the property value settings.

yStart

The position of starting point coordinate on Y-axis of the canvas when the enum axis is a vertical axis. The numeric value ascends from top to bottom. Please refer to the description on the xStart property for the property value settings.

yEnd

The position of ending point coordinate on Y-axis of the canvas when the enum axis is a vertical axis. Please refer to the description on the **xStart** property for the property value settings.

yPosition

The position of coordinate point on X-axis of the canvas when the enum axis is a vertical axis. Please refer to the description on the **xStart** property for the property value settings.

polarX

The position of starting point coordinate on X-axis of the canvas when the enum axis is a polar axis. Please refer to the description on the **xStart** property for the property value settings.

polarY

The position of starting point coordinate on Y-axis of the canvas when the enum axis is a polar axis. Please refer to the description on the **xStart** property for the property value settings.

polarLength

Sets the polar axis length.

startAngle

Sets the start angle when the enum axis is an angle axis. In plotting the circular sector pie chart, the circular sector can be customized with the startAngle and endAngle properties.

endAngle

Sets the end angle of an angle axis.

axisColor

Sets the axis line color of a enum axis.

axisLineStyle

Sets the style of axis line. Options include the dotted line, solid line, and none.

axisLineWeight

The axis line weight of enum axis.

axisArrow

Sets the style of arrow for enum axis.

title

The title name of enum axis.

titleFont

The title font of enum axis.

titleStyle

The title style of enum axis. Please refer to the description on uniStyle of numeric axis.

titleSize

The title font size of enum axis.

titleIndent

The blanks in the title of enum axis.

titleColor

The title font color of enum axis.

titleAngle

The title rotation angle of enum axis.

allowLabels

Indicates if the label value will be displayed.

labelFont

The font of label value.

labelStyle

The label value style. Please refer to the unitStyle of NumericAxis.

labelSize

The font size of label value.

labelIndent

The blank in-between the label value and the axis.

labelColor

The font color of label value.

labelStep

The display style of label. The labelStep=0 indicates that all labels will be displayed, and labelStep=2 indicates a label value will be displayed every other 2 labels.

labelAngle

The rotation angle of label value

labelOverlapping

The intersecting of label. If checked, the intersecting label is allowed; Otherwise, the intersecting label is not allowed, and the intersecting label, if any, will be overridden and hidden.

scalePosition

The tick mark can be displayed on the right or top, on the left or bottom, or across axis, or none tick marks appear. Once the label is positioned, the tick mark will change accordingly.

scaleStyle

The styles of tick marks, including the dotted line and the solid line.

scaleWeight

The weight of tick mark.

scaleLength

The length of tick mark.

displayStep

The display style of tick mark. If displayStep=0, then every tick mark will be displayed, and displayStep=2 indicates it becomes visible every other two.

allowRegions

Indicates if the spacer region will be displayed.

regionLineStyle

The type of spacing line in the spacer region.

regionLineColor

Color of spacing line displayed in the spacer region

regionLineWeight

Indicates the weight of spacing line displayed in the spacer region

regionColors

Color of spacer region

regionTransparent

Transparency of spacer region. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

isPolygonalRegion

Indicates whether it is a polygon spacer. This property is only valid in the polar coordinate system, and is fit for radar chart and other no-filling charts. For pie chart and other charts with fillings, such effect is not vivid. On the radar map, the shape of spacer region will change along with the number of categories.

DateAxis

name

Date axis name

visible

Show or hide axis

location

Date axis will appear on here. You can set the axis as vertical axis, horizontal axis, polar axis, or angle axis.

is3D

Indicates the 3-dimensional axis. If the 3D properties is adopted for at least one coordinate axis in the coordinate system on canvas, then the other coordinate axises in the current coordinate system will take on the 3D effects consistently in chart plotting.

threeDThickRatio

Indicates the thickness ratio of 3D axis when the axises in the chart have taken on the 3D effect. This parameter will determine the **3D Thickness**, which refers to the length of forward and backward edges. The threeDThickRatio takes effect **on condition that** there is no enum axis in the canvas coordinate system, and the axis 1 is the date axis. The default maximum 3D thickness is 60 pixels.

To compute the overall 3D thickness based on the 3D thickness radio of axis 1, **3D Thickness Ratio * Canvas Height** will be used. If the ratio is greater than 1, then it represents the absolute pixel value. In addition, the chart will be auto-plotted with 60 pixels if the 3D thickness is greater than 60.

autoCalcValueRange

Indicates that the value range of coordinate's axis will be computed automatically according to the coordinate points. If the **autoCalcValueRange** is set to true, then the value range of date axis will be set automatically according to the coordinate point on the axis.

endDate

Expiration date on date axis

beginDate

Start date on date axis

scaleScale

The unit of date axis, including year, month, day, hour, minutes, second, and millisecond.

format

The display format of scale values on the date axis, for example, yyyy/MM/dd, and yy-MM-dd.

xStart

The position of starting point coordinate on X-axis of the canvas when the date axis is a horizontal axis.

If the specified value is not greater than 1 and not less than 0, it indicates the chart will take up this percentage widths/lengths across the canvas. If the specified value is greater than 1 or less than 0, then it represents the pixel values. The chart will expand across the border if the pixels are greater than the width/length of canvas.

The origin point is in the top left corner if the specified value is not less than 0; Otherwise, it will be located in the bottom right corner if the specified value is less than 0.

xEnd

The position of ending point coordinate on X-axis of the canvas when the date axis is a horizontal axis. Please refer to the description on the **xStart** property for the property value settings.

xPostion

The position of coordinate point on Y-axis of the canvas when the date axis is a horizontal axis. Please refer to the description on the **xStart** property for the property value settings.

yStart

The position of starting point coordinate on Y-axis of the canvas when the date axis is a vertical axis. The numeric value ascends from top to bottom. Please refer to the description on the **xStart** property for the property value settings.

yEnd

The position of ending point coordinate on Y-axis of the canvas when the date axis is a vertical axis. Please refer to the description on the **xStart** property for the property value settings.

yPostion

The position of coordinate point on X-axis of the canvas when the date axis is a vertical axis. Please refer to the description on the **xStart** property for the property value settings.

polarX

The position of starting point coordinate on X-axis of the canvas when the date axis is a polar axis. Please refer to the description on the **xStart** property for the property value settings.

polarY

The position of starting point coordinate on Y-axis of the canvas when the date axis is a polar axis. Please refer to the description on the **xStart** property for the property value settings.

polarLength

Set the length of polar axis.

startAngle

Sets the start angle of angle axis if the date axis is an angle axis. When plotting the sector pie chart, you can use startAngle and endAngle to customize the sector.

endAngle

Sets the end angle of angle axis if the date axis is an angle axis.

axisColor

Sets the axis line color of a date axis.

axisLineStyle

Sets the style of axis line. Options include the dotted line, solid line, and none.

axisLineWeight

The axis line weight of date axis.

axisArrow

Sets the arrow style of date axis.

title

The title name of date axis.

titleFont

The title font style of date axis.

titleStyle

The title style of date axis. Please refer to the description on uniStyle of numeric axis.

titleSize

The title font size of date axis.

titleIndent

The blanks in the title of date axis.

titleColor

The color of title font of date axis.

titleAngle

The title rotation angle of date axis.

allowLabels

Indicates if the label value will be displayed.

labelFont

The font of label value.

labelStyle

The label value style. Please refer to the unitStyle of numeric axis.

labelSize

The font size of label value.

labelIndent

The blank in-between the label value and the axis.

labelColor

The font color of label value.

labelStep

The display interval of labels and its value is not less than 0. The labelStep=0 indicates that all labels will be displayed, and labelStep=2 indicates a label value will be displayed every other 2 labels.

labelAngle

The rotation angle of label value

labelOverlapping

The intersecting of label. If checked, the intersecting label is allowed; Otherwise, the intersecting label is not allowed, and the intersecting label, if any, will be overridden and hidden.

scaleLocation

The tick mark can be displayed on the right or top, on the left or bottom, or across axis, or none tick marks appear. Once the label is positioned, the tick mark will change accordingly.

scaleStyle

The style of tick mark, including the dotted line or the solid line.

scaleWeight

The weight of tick mark.

scaleLength

The length of tick mark.

displayStep

The display interval of scales and its value is not less than 0. The displayStep=0 indicates that all scale/tick marks will be displayed, and displayStep=2 indicates a scale/tick value will be displayed every other 2 scales/ticks.

allowRegions

Indicates if the spacer region will be displayed.

regionLineStyle

The type of spacing line in the spacer region.

regionLineColor

Color of spacing line displayed in the spacer region

regionLineWeight

Indicates the weight of spacing line displayed in the spacer region

regionColors

Color of spacer region

regionTransparent

Transparency of spacer region. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

isPolygonalRegion

Indicates whether it is a polygon spacer. This property is only valid in the polar coordinate system, and is fit for radar chart and other no-filling charts. For pie chart and other charts with fillings, such effect is not vivid. On the radar map, the shape of spacer region will change along with the number of categories.

Column

visible

Indicates if the column chart/histogram is visible

stacked

Indicates if it will be stacked. False by default, and set it to true if plotting the stacked chart.

In plotting, the data of the same category value will be stacked and displayed. **The isStacked property is only valid for the conventional column chart**.

regular

Indicates if it is a standard column. The default property of isRegularColumn is true, which refers to the standard column chart. The standard column chart is plotted on the basis of the category, series, and related data values. If setting the isRegularColumn to false, then you can plot the column chart arbitrarily. Two coordinate points are necessary for plotting each column, that is, the coordinates of the top left corner and the bottom right corner of the column. Then, the number of coordinates of axis data sequence must be even.

textOverlapping

Indicates if the text overlapping is allowed. If this option is selected, the text can be displayed with overlaps. Otherwise, the overlapping text will be overridden and hidden.

shadow

Indicates if the chart element will take on the shadow effect.

convexEdge

Indicates if it is a protruding border. The protruding border can only display for the plane and 3D columns.

transparent

Sets the degree of transparency for column chart/histogram. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

axis1

The name of axis 1, which can be the name of numerical axis, enum axis, or date axis.

data1

Indicates that the sequence is composed of the logical coordinate values on axis 1. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

axis2

The name of axis 2, which can be the name of numerical axis, enum axis, or date axis.

data2

Indicates that the sequence is composed of the logical coordinate values on axis 2. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

columnWidth

The width of column. The decimal values are percentages, and values greater than 1 are pixels. If the value of columnWidth is pixel, then the column width will not change when reaching the pixel maximum, and there is no interval between columns in this point. The column width is co-determined by the axis length, number of categories, and number of series.

columnShape

Column type, especially the square column, 3D square column, and round column

borderStyle

Column borderline type

borderWeight

Column borderline weight

borderColor

Column borderline color

fillColor

Fill color of column

text

Label text of column

textFont

The label text font

textStyle

Label text style of bold, italic, underline, and vertical text.

textSize

Label text font size

textColor

Identify the color of text

horizontalAlign

Identify the alignment of text in horizontal direction

verticalAlign

Identify the alignment of text in vertical direction

Line

endToHead

Indicates if it is an end-to-end closed line

isContinuousLine

Indicates if it is a continuous line. For those non-continuous lines, the number of data points must be an even number. The program will draw a straight line connecting these two points automatically.

visible

Indicates if the line chart element is visible

transparent

Sets the transparency for broken lines. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

textOverlapping

Indicates if the text overlapping is allowed. If this option is selected, the text can be displayed with overlaps. Otherwise, the overlapping text will be overridden and hidden.

shadow

Set the shadow effect for line chart element

axis1

The name of axis 1, which can correspond to the name of numeric axis, enum axis, or date axis.

data1

Indicates that the sequence is composed of the logical coordinate values on axis 1. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

axis2

The name of axis 2, which can correspond to the name of numeric axis, enum axis, or date axis.

data2

Indicates that the sequence is composed of the logical coordinate values on axis 2. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

markerStyle

The shape of break point.

lineStyle

Style of broken line.

lineWeight

Thickness of broken line.

lineColor

Color of broken line.

markerRadius

Radius of break point in pixels.

markerColor

Fill color of break point.

text

Label text of data point on the line.

textFont

Label font.

textStyle

The label text font style. Please refer to textStyle of column chart/histogram.

textSize

Label text size

textColor

Label text color

Dot

visible

Indicates if the point chart element is visible

transparent

Set the transparency for the point chart The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

textOverlapping

Indicates if the label text overlapping is allowed. If this option is selected, the text can be displayed with overlaps. Otherwise, the overlapping text will be overridden and hidden.

shadow

Indicates if the point chart element will take on the shadow effect.

axis1

The name of axis 1, which can correspond to the name of numeric axis, enum axis, or date axis.

data1

Indicates that the sequence is composed of the logical coordinate values on axis 1. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

axis2

The name of axis 2, which can correspond to the name of numeric axis, enum axis, or date axis.

data2

Indicates that the sequence is composed of the logical coordinate values on axis 2. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

markerStyle

The point style of data in the point.

lineStyle

The border line style of point chart element

lineWeight

The border line weight of point chart element

lineColor

The border line color of point chart element.

markerRadius

The radius of point in pixels.

markerColor

The color of point.

text

The label text of point chart element.

textFont

The label text font.

textStyle

The label text font style. Please refer to textStyle of column chart/histogram.

textSize

Label text size

textColor

Label text color

Polygon

visible

Indicates if the polygon is visible.

isTrapezia

Indicates if it is trapezoid. If checked, it is a trapezoid. In other words, the shadow of the line connecting two neighboring points on the logical axis is a trapezoid. If this option is not checked, then it will be a polygon. In other words, a closed polygon will be formed with lines connecting the neighboring points from end to end.

isContinuousTrapezia

Indicates if it is a continuous trapezia on condition that the isTrapezia is true. If the isContinuousTrapezia property is not checked, then a trapezoid can be composed of two numbers. In this case, the number of coordinates on the axis data sequence must be even.

transparent

The polygon chart elements are often used for plotting the area chart. For multiple series, the chart elements may be intersected. In this case, the chart element can be set as translucency. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

textOverlapping

Indicates if the text overlapping is allowed. If this option is selected, the text can be displayed with overlaps. Otherwise, the overlapping text will be overridden and hidden.

fillColor

The fill color for polygon

axis1

The name of axis 1, which can correspond to the name of numeric axis, enum axis, or date axis.

data1

Indicates that the sequence is composed of the logical coordinate values on axis 1. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

axis2

The name of axis 2, which can correspond to the name of numeric axis, enum axis, or date axis.

data2

Indicates that the sequence is composed of the logical coordinate values on axis 2. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

borderStyle

The style of polygon border.

borderWeight

The weight of polygon border.

borderColor

The color of polygon border.

text

The label text of polygon.

textFont

The label text font of polygon.

textStyle

The label text style of polygon. Please refer to textStyle of column charts or histogram.

textSize

The label text font of polygon.

textColor

The label text font color of polygon.

Sector

visible

Indicates if the sector/ring form is visible

isAccumulate

isAccumulate=false indicates no cumulating, which means it is a non-commutative free ring. A sector or ring is determined by two sets of coordinates, that is, the inner diameter point of starting side and the outer diameter point of ending side.

isAccumulate=true indicates cumulating. One set of coordinates determine a sector or ring. All series of a type will cumulate automatically and form a complete circle or circular sector.

textOverlapping

Indicates if the text overlapping is allowed. If this option is selected, the text can be displayed with overlaps. Otherwise, the overlapping text will be overridden and hidden.

transparent

Set the transparency of circular sector or ring form. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

axis1

The name of axis 1, which can correspond to the name of numeric axis, enum axis, or date axis.

data1

Indicates that the sequence is composed of the logical coordinate values on axis 1. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

axis2

The name of axis 2, which can correspond to the name of numeric axis, enum axis, or date axis.

data2

Indicates that the sequence is composed of the logical coordinate values on axis 2. When plotting, the logical coordinate value will be converted to the physical coordinate value automatically.

borderStyle

Borderline type

borderWeight

Borderline weight

borderColor

Borderline color

fillColor

Fill color for circular sector/ring form

text

The label text contents for circular sector/ring

textFont

The label text font.

textStyle

The label text style. Please refer to textStyle of column charts or histogram.

textSize

Label text size

textColor

Label text color

textLineStyle

Label text line style

textLineWeight

Label text line weight

textLineColor

Label text line color

BackGround

backColor

The backcolor of canvas.

transparent

The transparency settings of background chart element. The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

visible

Indicates if the background chart element is visible

Text

text

The contents of text.

visible

Indicates if the legend is visible.

x

The starting X-coordinates of text chart element. Please refer to the description on the **xStart** property of the numeric axis.

y

The starting Y-coordinates of text chart element. Please refer to the description on the **xStart** property of the numeric axis.

textFont

The font of text.

textStyle

The style of text. Please refer to the description on unitStyle property of the numeric axis.

textSize

The text font size.

textColor

The color of text font.

text Angle

The rotation angle of text.

hAlign

The horizontal alignment style of text contents to the center coordinates.

vAlign

The vertical alignment style of text contents to the center coordinates.

Legend

legendText

The text contents of legend.

backColor

The backcolor.

columns

The number of columns in the legend layout are. This property can be used to set the legend orientation of either landscape or portrait.

transparent

Sets the transparency of background of legends The value range for the degree of transparency is [0,1], in which the 0 represents the 100% transparency (i.e. totally transparent), and the 1 represents opaqueness. The values higher than 1 give the same totally opaque effect as 1, and the values less than 0 represent the total transparency as well.

iconWidth

The width of icons in the legends in pixels

edgeIndent

The blank left intentionally and positioned against the legend borders regarding the legend position.

visible

Indicates if the legend is visible.

x

The X-coordinates of legends. For the value, please refer to description on the **xStart** property of the numeric axis.

y

The y-coordinates of legends. The value reference the xStart property statement in the NumericAxis description.

width

The width of legend area.

height

The height of legend area.

borderStyle

The border style of legend area.

borderWeight

The border weight of legend area.

borderColor

The border color of legend area.

textFont

The text font of legend.

textStyle

The text font style of legend. Please refer to textStyle of column chart/histogram.

textSize

The text font size of legends

textColor

The text font color of legends.

legendType

The type of legend in the legend area, including rectangle, dot, line, dotted line, or none.

legendLineStyle

The border line style of legend in the legend region.

legendLineWeight

The border line weight of icons in the legend region.

legendLineColor

The border line color of icons in the legend region.

legendFillColor

The fill color of icons in the legend area

legendMarkerShape

The style of legend point in the legend area. The setting is valid when the legendType is the dots or dotted lines.