# Code reference

# Chapter1 Fundamentals[1][2]

*Fundamentals* lists code examples about topics in the following chapters – Numerical Calculation, String, Datetime, Sequence, Table Sequences and Records, Maintenance of Table Sequences & Rrecord Sequences, External Files, Database and Program Logic.

## 1.1 Numerical Calculation[3][4]

This chapter lists code examples of performing numerical calculations, including Null value judgment, Random values, Constants of various data types, Signs of numbers, Involution and evolution, Decimal truncation and rounding, Continued multiplication & factorial, Accumulated sum, Greatest common divisor & least common multiple, Permutation & combination, Pi, Trigonometric functions, Logarithmic functions, Compute expressions with different conditions, Use temp variable in expressions and Logic operations.

### 1.1.1 Null value judgment[5][6]

| | A | |
|---|---|---|
| 1 | **=null** | |
| 2 | **=0** | 0 |
| 3 | **=if(A1==null,"null","not null")** | null |
| 4 | **=if(!A1,"null","not null")** | null |
| 5 | **=if(A2!=null,"not null","null")** | not null |
| 6 | **=[,1,2,3].ifn()** | 1 |
| 7 | **=["",,,0,3].nvl()** | 0 |

### 1.1.2 Random values[7][8]

| | A | |
|---|---|---|
| 1 | **=rand()** | Get a random value between **0** and **1** |
| 2 | **=rand(1000)** | Get random integer values between **0** and **1,000** inclusive |

[1] url:jcp
[2] key:fundamentals
[3] url:szjs
[4] key:numerical calculation
[5] url:kzpd
[6] key:null,ifn
[7] url:sjs
[8] key:rand

## 1.1.3 Constants of various data types[9][10]

| | A | |
|---|---|---|
| 1 | **time** | String type "time" |
| 2 | **-3415** | 32-bit integer -3415 |
| 3 | **3.1415927** | Floating point number 3.1415927 |
| 4 | **1101022000L** | 64-bit long integer 1101022000 |
| 5 | **12345678901** | An integer exceeding the value range of a 32-bit integer will be automatically parsed into the 64-bit long integer 12345678901 |
| 6 | **35%** | 0.35, which is floating point number represented by a percentage |
| 7 | **0x33** | The value is 51, a hexadecimal long integer headed by 0x |
| 8 | **'345+6** | The value is string "345+6"; the sign ' marks a string type constant |

## 1.1.4 Signs of numbers[11][12]

| | A | |
|---|---|---|
| 1 | **=sign(45)** | Return 1 for a positive number |
| 2 | **=sign(-100.34)** | Return -1 for a negative number |
| 3 | **=sign(0)** | Return 0 for zero |
| 4 | **=abs(-4.6)** | Return the absolute value 4.6 |

## 1.1.5 Involution and evolution[13][14]

| | A | |
|---|---|---|
| 1 | **=power(2,3)** | Cube |
| 2 | **=power(-2,3)** | Cube |
| 3 | **=power(4,0.5)** | Square root |
| 4 | **=sqrt(8,2)** | Square root |
| 5 | **=power(27,1/3)** | Cube root |
| 6 | **=sqrt(8,3)** | Cube root |

[9] url:gzlxdcs
[10] key:constant
[11] url:sdfh
[12] key:sign,abs
[13] url:cfykf
[14] key:power

## 1.1.6 Decimal truncation and rounding[15][16]

| | A | |
|---|---|---|
| 1 | **=round(3451251.274,1)** | Round down to 1 decimal place |
| 2 | **=round(3451251.274,2)** | Round down to 2 decimal places |
| 3 | **=ceil(3450001.003,-2)** | Carry the remaining figures to the column of hundreds |
| 4 | **=ceil(3450001.003,2)** | Carry the remaining figures and round off to 2 decimal places |
| 5 | **=floor(3451291.234,-2)** | Round to the column of hundreds and discard all the remaining figures |
| 6 | **=floor(3451281.238,2)** | Round off to 2 decimal places and discard the remaining figure |

## 1.1.7 Continued multiplication & factorial[17][18]

| | A | |
|---|---|---|
| 1 | **=product(2, 3, 5, 7)** | 210, the value of calculating 2*3*5*7 |
| 2 | **=product([7, 4, 4])** | 112, the value of continual multiplication of the numbers in the given sequence |
| 3 | **=fact(5)** | 120, the value of factorial 5 |
| 4 | **=fact(0)** | 1 |

## 1.1.8 Accumulated sum[19][20]

| | A | |
|---|---|---|
| 1 | **=[1,2,3,4].(cum(~))** | [1,3,6,10], iterative sum |

[15] url:jqxsysswr
[16] key:round,ceil,floor
[17] url:lcyjc
[18] key:product,fact
[19] url:lj
[20] key:accum,sum

### 1.1.9 Greatest common divisor & Least common multiple[21][22]

| | A | |
|---|---|---|
| 1 | **=gcd(2000, 875, 325)** | 25, gcd of the 3 numbers |
| 2 | **=gcd([1001, 28])** | 7, gcd of the members of the sequence |
| 3 | **=gcd(20005, 1234)** | 1; the two numbers are coprime |
| 4 | **=lcm(10, 35, 28)** | 140, lcm of the 3 numbers |
| 5 | **=lcm([1001, 111])** | 111111, lcm of the members of the sequence， |

### 1.1.10 Permutation & combination[23][24]

| | A | |
|---|---|---|
| 1 | **=combin(10, 3)** | 120, the number of combinations of selecting 3 from 10 elements |
| 2 | **=combin(5, 4)** | 5, the number of combinations of selecting 4 from 5 elements |
| 3 | **=permut(10, 3)** | 720, the number of permutations of selecting 3 from 10 elements |
| 4 | **=permut(5, 4)** | 120, the number of permutations of selecting 4 from 5 elements |

### 1.1.11 Pi[25][26]

| | A | |
|---|---|---|
| 1 | **=pi()** | **π** |
| 2 | **=pi(4)** | **4*π** |

[21] url:zdgysyzxgbs
[22] key:gcd,lcm
[23] url:plhzh
[24] key:combin,permut
[25] url:pai
[26] key:pi

## 1.1.12 Infinity[27][28]

| | A | |
|---|---|---|
| 1 | **=inf()** | Positive infinity |
| 2 | **=-inf()** | Negative infinity |

## 1.1.13 Trigonometric functions[29][30]

| | A | |
|---|---|---|
| 1 | **=sin(pi(30/180))** | Sine |
| 2 | **=cos(pi()/2)** | Cosine |
| 3 | **=tan(pi()/4)** | Tangent |
| 4 | **=asin(0.5)** | Arc sine |
| 5 | **=acos(-0.5)** | Arc cosine |
| 6 | **=atan(1)** | Arc tangent |
| 7 | **=sinh(1)** | Hyperbolic sine |
| 8 | **=cosh(4)** | Hyperbolic cosine |
| 9 | **=tanh(0.5)** | Hyperbolic tangent |
| 10 | **=asinh(10)** | Inverse hyperbolic sine |
| 11 | **=acosh(10)** | Inverse hyperbolic cosine |
| 12 | **=atanh(0.5)** | Inverse hyperbolic tangent |

## 1.1.14 Logarithmic functions[31][32]

| | A | |
|---|---|---|
| 1 | **=lg(10000)** | Logarithm to base 10 |
| 2 | **=lg(8,2)** | Logarithm to base 2 |
| 3 | **=ln(1000)** | Natural logarithm |
| 4 | **=exp(A2)** | *e* to the $n^{th}$ power |

[27] url:infinity
[28] key:inf
[29] url:sjhs
[30] key:sin,cos,tan,asin,acos,atan
[31] url:dshs
[32] key:lg,ln,exp

## 1.1.15 Compute expressions with different conditions[33][34]

| | A | |
|---|---|---|
| 1 | **3000** | |
| 2 | **=if(A1>10000, A1*0.45+450, A1>5000, A1*0.15+150, A1*0.05)** | 150.0 |
| 3 | **manager** | |
| 4 | **=case(A3,"president":500,"manager":300,"employee":150)** | 300 |

## 1.1.16 Use temp variables in expressions[35][36]

| | A | |
|---|---|---|
| 1 | **=(a=1,b=a*3,b+4)** | **7** |
| 2 | **=a** | **1** |
| 3 | **=b** | **3** |

## 1.1.17 Logic operations[37][38]

| | A | |
|---|---|---|
| 1 | **=and(6,10)** | 2, bitwise AND |
| 2 | **=or(3,5)** | 7, bitwise OR |
| 3 | **=not(6)** | -7, bitwise NOT |
| 4 | **=xor(6,11)** | 13, bitwise XOR |
| 5 | **=shift(13,2)** | 3, Shift right two bit positions |
| 6 | **=shift(13,-2)** | 52, Shift left two bit positions |

# 1.2 String[39][40]

This chapter lists code examples about handling strings, including Generate a fixed-length string, Search for and replace a substring, Get part of a string, Concatenation of strings, Upper and lower-case letters identification and conversion, Remove blank spaces on both sides of a string,

[33] url:gjtjjsbtdbds
[34] key:if,case
[35] url:zbdsnsylsbl
[36] key:expression, temporary variable
[37] url:logicoperations
[38] key:bitwise,shift
[39] url:zfc
[40] key:string

Format string matching, Match format strings with ' *', Get Unicode and return characters by Unicode, Split a string into a sequence, Concatenate members of a sequence into a string, Delete certain characters from a string, Check if a string consists of alphabetic and numeric characters, Compute an expression stored in a string, Use parameters to compute a string expression, Adjuste string expression during editing, Check if it is string, Get MD5 signature string, Generate random string of specified length, Match string by regular expression, Get substring from source string, Split away words/numbers from string and Parse a string as string.

## 1.2.1 Generate a fixed-length string[41][42]

| | A | |
|---|---|---|
| 1 | **=fill(" ",10)** | **"  "** |
| 2 | **=len(A1)** | **10** |
| 3 | **=fill("ab",10)** | **abababababababababab** |

## 1.2.2 Search for and replace a substring[43][44]

| | A | |
|---|---|---|
| 1 | **=pos("abcdef","def")** | Determine the position of **"def"** in **"abcdef"** |
| 2 | **=pos("abcdefdef","def",5)** | Search from the fifth character |
| 3 | **=replace("abca","a","ABC")** | Replace **"a"** with **"ABC"** in **"abca"** |
| 4 | **=replace("abc'abc'","a","ABC")** | The substring in the single quotation marks will also be replaced |
| 5 | **=replace@q("abc'abc'","a","ABC")** | The substring in the single quotation marks will not be replaced |

## 1.2.3 Get part of a string[45][46]

| | A | |
|---|---|---|
| 1 | **=mid("abcde",2,1)** | Get the second character |
| 2 | **=mid("abcde",3,2)** | Get 2 characters from the third position |
| 3 | **=mid("abcde",2)** | Get characters from the second position to the end |
| 4 | **=left("abcdefg",3)** | The leftmost three characters |
| 5 | **=right("abcde",2)** | The rightmost two characters |

41 url:cstdczfc
42 key:fill,len
43 url:czhthzc
44 key:pos,replace
45 url:qczfcdbf
46 key:mid,left,right

## 1.2.4 Concatenation of strings[47][48]

| | A | |
|---|---|---|
| 1 | **="ab"+"cd"** | abcd |
| 2 | **="ab"/"cd"** | abcd |
| 3 | **="3"+2** | The result is **5** because the string is taken as number when computed with numbers. |
| 4 | **="ab"+1** | The result is **1** because the string that can't be converted to a number will be taken as 0. |

## 1.2.5 Upper- and lower-case letters identification and conversion[49][50]

| | A | |
|---|---|---|
| 1 | **=upper("abcdef")** | **"ABCDEF"** |
| 2 | **=upper("ABCdef")** | **"ABCDEF"** |
| 3 | **=lower("abcDEF")** | **"abcdef"** |
| 4 | **=isupper("ABC")** | **true** |
| 5 | **=islower("ABC")** | **false** |
| 6 | **=islower("aBc")** | **false** |
| 7 | **=isupper("Bc")** | **false** |

## 1.2.6 Remove blank spaces on both sides of a string[51][52]

| | A | |
|---|---|---|
| 1 | **=trim(" abc ")** | Remove spaces on both sides |
| 2 | **=trim@l(" abc ")** | Remove the spaces on the left |
| 3 | **=trim@r(" abc ")** | Remove the spaces on the right |

47 url:zfcpj
48 key:string, concatenation
49 url:dxxsbhzh
50 key:upper,lower,isupper,islower
51 url:qczfcsbdydkb
52 key:trim

## 1.2.7 Format string matching[53][54]

| | A | |
|---|---|---|
| 1 | **=like("abc123","abc1?3")** | **"?"** is used to match a single character |
| 2 | **=like("abcefg","abc*")** | **"*"** is used to match 0 or multiple characters |

## 1.2.8 Match format strings with ' *'[55][56]

| | A | |
|---|---|---|
| 1 | **=like("ab*123", "ab*1?3")** | Make **"*"** escaped with "\" to match the character itself. |
| 2 | **=like("a*bcefg","a*bc*")** | **true** |

## 1.2.9 Get Unicode and return characters by Unicode[57][58]

| | A | |
|---|---|---|
| 1 | **=asc("a")** | Get the unicode of character **"a"** |
| 2 | **=char(68)** | Get the character corresponding to code **"68"** |
| 3 | **=asc("USA")** | Get the Unicode corresponding to character **"U"** |

[53] url:ppmsc
[54] key:like
[55] url:ppdydmsc
[56] key:like
[57] url:qdzfbmhybmfhzf
[58] key:asc,char

## 1.2.10 Split a string into a sequence[59][60]

| | A | |
|---|---|---|
| 1 | **="12345678"** | |
| 2 | **=len(A1)** | |
| 3 | **=A1.split()** | Split A1 into a sequence of characters |
| 4 | **="a,[b,c],d"** | |
| 5 | **=A4.split@cp()** | Return a sequence which consists of three members **a** ,**[b,c], d** where the member **[b,c]** is a sequence |
| 6 | **=A4.split@c()** | Return a sequence which consists of three members **a** ,**[b,c], d** where the member [**b,c]** is a string instead of a sequence |
| 7 | **=A4.split@cb()** | Return a sequence which consists of four members **a** ,**[b,c], d.** The quotation marks and brackets will not be matched. |
| 8 | **="a;[b;c];d".split@p(";")** | Use "**;**" as the delimiter |
| 9 | **="a b c"** | |
| 10 | **=A1.split@t()** | Return a sequence consisting of a, b, c and perform trim on both sides |
| 11 | **="a1b2c".split@r("(\\d)")** | Return a sequence consisting of a, b, c where the delimiter is a regular expression |

## 1.2.11 Concatenate members of a sequence into a string[61][62]

| | A | |
|---|---|---|
| 1 | **=[1,"abc,def",[2,4],"{7,8}"]** | |
| 2 | **=A1.concat@cq()** | Use **","** as the delimiter to concatenate members , in which strings will be quoted |
| 3 | **=A1.concat()** | String members don't need to be quoted and separated when being concatenated |
| 4 | **=A1.concat("&")** | Use **"&"** as the delimiter |

[59] url:jzfcccxl
[60] key:len,split
[61] url:jxlpczfc
[62] key:concat

## 1.2.12 Delete certain characters from a string[63][64]

| | A | |
|---|---|---|
| 1 | abcda123efag | |
| 2 | =replace(A1,"123","") | Remove "123" |
| 3 | =replace(A1,"a","") | remove "a" |

## 1.2.13 Check if a string consists of alphabetic and numeric characters[65][66]

| | A | |
|---|---|---|
| 1 | 2345$#dfAgsdf23* | |
| 2 | =len(A1) | 16 |
| 3 | =A1.split() | Split A1 into a sequence of characters |
| 4 | =A3.count(!isdigit(~) && !isalpha(~))==0 | Check if it is a alphanumeric string |

## 1.2.14 Compute an expression stored in a string[67][68]

| | A | |
|---|---|---|
| 1 | =eval("1+5") | Compute 1 + 5 |
| 2 | =eval($[A1+2]) | Compute A1 + 2 |

## 1.2.15 Use parameters to compute a string expression[69][70]

| | A | |
|---|---|---|
| 1 | =eval("?+5",3) | Equivalent to "3 + 5" |
| 2 | =eval("(?1+1)/?2",3,4) | Equivalent to "(3 + 1)/4" |

63 url:czfcscbxydzf
64 key:replace
65 url:pdzfcsfszmhszgc
66 key:len,split,count,isdigit,isalpha
67 url:jscczzfczdbds
68 key:eval
69 url:jszfcbdsssycs
70 key:eval

## 1.2.16 Adjust string expressions during editing[71][72]

| | A |
|---|---|
| 1 | **=$[B1+4]** |

Strings written in the format of **$[]** will adjust themselves during editing

## 1.2.17 Check if it is string[73][74]

| | A | |
|---|---|---|
| 1 | **=ifstring("abc")** | true |

## 1.2.18 Get MD5 signature string[75][76]

| | A | |
|---|---|---|
| 1 | **=md5("abced")** | EBB080AFAAC3A990AD3F1D0F21742FAC |

## 1.2.19 Generate a random string of specified length[77][78]

| | A | |
|---|---|---|
| 1 | **=rands("abc",5)** | baaca |

[71] url:kszbjgcbqdbdszfc
[72] key:editing, adjust,expression string
[73] url:checkifitisstr
[74] key:check,string
[75] url:getMDBsignaturestr
[76] key:MD5,signature
[77] url:generaterandstrofspecifiedlen
[78] key:rand,len

## 1.2.20 Match string with regular expression[79][80]

| | A | |
|---|---|---|
| 1 | **4,233,a,test** | |
| 2 | **=A1.regex("(\\d),([0-9]*),([a-z]),([a-z]*)")** | Return a sequence of [4, 233, a, test] |
| 3 | **4,233,A,tEst** | |
| 4 | **=A3.regex@c("(\\d),([0-9]*),([a-z]),([a-z]*)")** | Return a sequence consisting of [4, 233, A, tEst]; case-insensitive |
| 5 | **小明,中国** | |
| 6 | **=A5.regex@u("(\\u5c0f\\u660e),(\\u4e2d\\u56fd)")** | Return a sequence of [小明,中国], where the elements are Chinese chracters |

## 1.2.21 Get substring from source string[81][82]

| | A | |
|---|---|---|
| 1 | **=substr("abcdef","cd")** | ef, return string after the specified substring |
| 2 | **=substr@l("abcdef","cd")** | ab, return string before the specified substring |
| 3 | **=substr@q("ab\'cdef\'","cd")** | null, ignore quoted string |

[79] url:matchstrwithregexp
[80] key:string,reg expression
[81] url:getsubstrfromsourcestr
[82] key:substring,source string

## 1.2.22 Split away words/numbers from string[8384]

| | A | |
|---|---|---|
| 1 | **4℃,23,a,test?my_file 57b** | |
| 2 | **=A1.words()** | a,test,my,file,b; English words are split away from string |
| 3 | **=A1.words@d()** | 4,23,57;numbers are split away from string |
| 4 | **=A1.words@a()** | 4,23,a,test,my,file,b,57; both English words and numbers are split away from string |
| 5 | **=A1.words@i()** | a,test,my,file,b57; neiigboring English letters and numbers are treated as a whole |
| 6 | **=A1.words@w()** | 4,℃, ,, 23, ,, a, ,, test, ?, my, _, file, , b,57; all characters are split away |

## 1.2.23 Parse a string as string[8586]

| | A | |
|---|---|---|
| 1 | **="10:20:30"** | |
| 2 | **=parse(A1)** | 10:20:30, time type |
| 3 | **="10℃"** | |
| 4 | **=parse@n(A3)** | 10, Only parse the number in a number headed string |

# 1.3 Datetime[8788]

This chapter lists code examples about handling datetime, including Get the current date and time, Get different parts of a datetime value, Compose a datetime/date/time value, The datetime which is a certain time period before/later, Interval between two datetimes, Find what day a date is, The first and last day of a week/month/quarter, The number of days in a month/quarter/year, Generate a datetime sequence by fixed intervals, Get the second and last Fridays in a month/quarter/year and the total number of Fridays in this period, Check if it is date, Get age and Work days calculations.

[83] url:splitawaywordsnumbers
[84] key:split away, word, number, string
[85] url:parsestrasstr
[86] key:parse,string
[87] url:rqsj
[88] key:datetime

## 1.3.1 Get the current date and time[8990]

| | A | |
|---|---|---|
| 1 | **=now()** | |
| 2 | **=now@d()** | Return a date type value |
| 3 | **=now@m()** | Accurate to the minute |

## 1.3.2 Get different parts of a datetime value[9192]

| | A |
|---|---|
| 1 | **=now()** |
| 2 | **=year(A1)** |
| 3 | **=month(A1)** |
| 4 | **=day(A1)** |
| 5 | **=time(A1)** |
| 6 | **=hour(A1)** |
| 7 | **=minute(A1)** |
| 8 | **=second(A1)** |

## 1.3.3 Compose a datetime/date/time value[9394]

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **1989** | **'02** | **'01** | **'02** | **34** | **55** |
| 2 | **=string(A1)+"-"+string(B1)+"-"+string(C1)+" "+string(D1)+":"+string(E1)+":"+string(F1)** | | | | | |
| 3 | **=datetime(A2)** | | | | | |
| 4 | **=datetime(A2,"yyyy-MM-dd HH:mm:ss")** | | | | | |
| 5 | **=date(A1,int(B1),int(C1))** | | | | | |
| 6 | **=time(int(D1),E1,F1)** | | | | | |
| 7 | **=datetime(A1,int(B1),int(C1),int(D1),E1,F1)** | | | | | |
| 8 | **=datetime(A5,A6)** | | | | | |

89 url:hqdqrqhsj
90 key:now
91 url:qdrqsjdgbf
92 key:year,month,day,hour,minute,second
93 url:ybdpcrqsj
94 key:string,int,date,time,datetime

## 1.3.4 The datetime which is a certain time period before/later[9596]

| | A | |
|---|---|---|
| 1 | 2006-07-05 | |
| 2 | =elapse(A1,5) | 5 days later |
| 3 | =elapse("1972-11-08 10:20:30",-10) | 10 days before |
| 4 | =elapse@s(datetime(A1),5) | 5 seconds later |
| 5 | =elapse@s("1972-11-08 10:20:30",-10) | 10 seconds before |
| 6 | =elapse@m(A1,-1) | 1 month before |
| 7 | =elapse@y(A1,-1) | 1 year before |

## 1.3.5 Interval between two datetimes[9798]

| | A | B | |
|---|---|---|---|
| 1 | 2010-5-01 23:20:15 | 2010-05-03 01:01:01 | |
| 2 | =interval(A1,B1) | | The number of days between two datetimes |
| 3 | =interval@s(A1,B1) | | The number of seconds between two datetimes |
| 4 | =interval@y(A1,"2001-01-01") | | The number of years between two dates |
| 5 | =interval@m(A1,"2001-01-01") | | The number of months between two dates |
| 6 | =interval@ms(A1,now()) | | The number of milliseconds between two datetimes |
| 7 | =interval(A1,B1) | | The number of days between two datetimes |
| 8 | =interval@s(A1,B1) | | The number of seconds between two datetimes |

95 url:xjmgsddrqsj
96 key:elapse
97 url:lgrqsjdjg
98 key:interval

## 1.3.6 Find what day a date is[99][100]

| | A |
|---|---|
| 1 | 2005-01-08 |
| 2 | **=day@w(A1)** |

Get what day the date is; 1 stands for "**Sunday**"

## 1.3.7 The first and last day of a week/ month/quarter[101][102]

| | A | |
|---|---|---|
| 1 | 2006-03-06 | |
| 2 | **=pdate@w(A1)** | The first day of the week |
| 3 | **=pdate@we(A1)** | The last day of the week |
| 4 | **=pdate@q(A1)** | The first day of the quarter |
| 5 | **=pdate@qe(A1)** | The last day of the quarter |
| 6 | **=pdate@m(A1)** | The first day of the month |
| 7 | **=pdate@me(A1)** | The last day of the month |

## 1.3.8 The number of days in a month/quarter/a year[103][104]

| | A | |
|---|---|---|
| 1 | 2007-08-08 | |
| 2 | **=days(A1)** | The number of days in the month in **A1** |
| 3 | **=days@y(2006)** | The number of days in the year **2006** |
| 4 | **=days@y(A1)** | The number of days in the year in A1 |
| 5 | **=days@q(A1)** | The number of days in the quarter in **A1** |

[99] url:xqj
[100] key:day
[101] url:zyjddythzhyt
[102] key:pdate
[103] url:mymjmndts
[104] key:days

## 1.3.9 Generate a datetime sequence by fixed intervals[105][106]

| | A | |
|---|---|---|
| 1 | 2000-08-10 12:00:00 | |
| 2 | =periods@y(A1,now(),1) | Set 1 year as the interval unit |
| 3 | =periods@q(A1,now(),1) | Set 1 quarter as the interval unit |
| 4 | =periods@m(A1,now(),2) | Set 2 months as the interval unit |

## 1.3.10 Get the second and last Fridays in a month/quarter/year and the total number of Fridays in this period[107][108]

| | A | |
|---|---|---|
| 1 | =now() | |
| 2 | =pdate@m(A1) | The start date of the month in **A1** |
| 3 | =pdate@me(A1) | The end date of the month in **A1** |
| 4 | =periods(A2, A3,1) | The sequence of dates between the first day and the last day in the month |
| 5 | =A4.select(day@w(~)==6) | The sequence of Fridays in **A4** |
| 6 | | An alternative method |
| 7 | =elapse(A2,(d=day@w(A2),if(d==7,6,6-d))) | Get the first Friday |
| 8 | =periods@x(A7,A3,7) | Get the sequence of Fridays, which doesn't contain the end date of the month got in A3 |
| 9 | =A8(2) | Get the second Friday |
| 10 | =A8.m(-1) | Get the last Friday |
| 11 | =A8.len() | Get the number of Fridays |

[105] url:cszqjgdrqsjxl
[106] key:periods
[107] url:myjnddeghzhygxqwgyjgxqw
[108] key:now,pdate,periods,elapse,len

## 1.3.11 Check if it is date[109][110]

| | A | |
|---|---|---|
| 1 | **=ifdate("2020-04-24")** | false |
| 2 | **=ifdate(date("2020-04-24"))** | true |

## 1.3.12 Get age[111][112]

| | A | |
|---|---|---|
| 1 | **1995-3-31** | |
| 2 | **=now@d()** | 2020-03-30 |
| 3 | **=age(A1)** | 24, accurate to day |
| 4 | **=age@m(A1)** | 25, accurate to month |
| 5 | **=age@y(A1)** | 25, accurate to year |

## 1.3.13 Workdays calculations[113][114]

| | A | |
|---|---|---|
| 1 | **2020-4-24** | |
| 2 | **2020-5-10** | |
| 3 | **[2020-4-26,2020-5-1,2020-5-4,2020-5-5,2020-5-9]** | A sequence of dates according to public holidays |
| 4 | **=workday(A1,1)** | 2020-04-27, the date one non-week-day after A1's date |
| 5 | **=workday(A1,1,A3)** | 2020-04-26, the date one workday after A1's date |
| 6 | **=workdays(A1,A2)** | A sequence of non-week-days between A1's date and A2's date |
| 7 | **=workdays(A1,A2,A3)** | A sequence of workdays between A1's date and A2's date |

109 url:checkifitisdate
110 key:check,date
111 url:getage
112 key:age
113 url:workdayscal
114 key:workday

## 1.4 Sequence[115][116]

This chapter lists code examples of handling sequences, including Check if it is a sequence, Get a sequence member or a sub-sequence in reverse direction, Get a sequence member or a sub-sequence in cycles, Get a sub-sequence but report no error if a specified position is beyond range, Generate a fixed-length sequence consisting of duplicate members, Duplicate a sequence (repeatedly) to generate a new sequence, Generate a continuous interger sequence, Exchange positions of member groups of a sequence, Insert one or multiple members to a sequence, Delete one or multiple members from a sequence, Modify one or multiple members of a sequence, Modify sequence member(s) at specified position(s) and supply values at the beyond-range position(s), Insert a sequence as a member into another sequence, Compare sequences in ASCII dictionary mode, Get inversed sequence, Get & count unique members in a sequence, Stretch sequence to specified length and Sequence-related logic operations.

### 1.4.1 Check if it is a sequence[117][118]

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

### 1.4.2 Get a sequence member and a sub-sequence in reverse direction[119][120]

| | A | |
|---|---|---|
| 1 | **=[1,2,3,4,5,6].m(-3)** | **4** |
| 2 | **=[1,2,3,4,5,6].m([-2,-3])** | **[5,4]** |

115 url:xl
116 key:sequence
117 url:pdsfsxl
118 key:ifa
119 url:dqxlcyhzxl
120 key:m

### 1.4.3 Get a sequence member or a sub-sequence in cycles[121][122]

| | A | |
|---|---|---|
| 1 | **[1,2,3,4,5,6]** | |
| 2 | **=A1.m@r(10)** | **4** |
| 3 | **=A1.m@r([1,5,10])** | **[1,5,4]** |

### 1.4.4 Get a sub-sequence but report no error if a specified position is beyond range[123][124]

| | A | |
|---|---|---|
| 1 | **=[1,2,3,4,5,6].m@0([10,1,4,5])** | **[1,4,5];**out-of-range members are not included |

### 1.4.5 Generate a fixed-length sequence consisting of same members[125][126]

| | A | |
|---|---|---|
| 1 | **=5*[1]** | **[1,1,1,1,1]** |
| 2 | **=3.("a")** | **[a,a,a]** |

### 1.4.6 Duplicate a sequence (repeatedly) to generate a new sequence[127][128]

| | A | |
|---|---|---|
| 1 | **=[1,2,3]** | |
| 2 | **=3*A1** | **[1,2,3,1,2,3,1,2,3]** |

[121] url:xhqxlcyhzxl
[122] key:m
[123] url:qczxldyjbbc
[124] key:m
[125] url: csytycygcddcxl
[126] key:fixed length sequence, same members
[127] url:fzdbxlcsxxl
[128] key:duplicate, new sequence

## 1.4.7 Generate a continuous integer sequence[129][130]

| | A | |
|---|---|---|
| 1 | **=to(8)** | **[1,2,3,4,5,6,7,8]** |
| 2 | **=to(3,5)** | The result is **[3,4,5]** which is from **3** to **5** |
| 3 | **=to@s(3,5)** | The result is **[3,4,5,6,7]** which counts **5** numbers forward from **3** |
| 4 | **=to@s(7,-3)** | The result is **[7,6,5]** |

## 1.4.8 Exchange positions of member groups of a sequence[131][132]

| | A | |
|---|---|---|
| 1 | **=[1,2,3,4,5,6,7,8].swap([2,3,4],[6,7])** | **[1,6,7,5,2,3,4,8]** |

## 1.4.9 Insert one or multiple members to a sequence[133][134]

| | A | |
|---|---|---|
| 1 | **=[1,2,3,4].insert(0,5)** | [1,2,3,4,5]. Insert the members at the end |
| 2 | **=[1,2,3,4].insert(1,5)** | [5,1,2,3,4]. Insert the members at the beginning |
| 3 | **=[1,2,3,4].insert(3,[5,6])** | [1,2,5,6,3,4]. Insert multiple members |
| 4 | **[1,2,3,4]** | [1,2,3,5,6,4**]**, the new A4 after A5 is executed |
| 5 | **=A4.insert@n(4,[5,6])** | **[**5,6**]**, return the newly-added member |

## 1.4.10 Delete one or multiple members from a sequence[135][136]

| | A | |
|---|---|---|
| 1 | **=[11,12,13,14].delete(2)** | Delete one member and the result is **[11,13,14]** |
| 2 | **=[11,12,13,14].delete([2,4])** | Delete multiple members and the result is **[11,13]** |

[129] url:cslxdslqj
[130] key:to
[131] url:jhxldbfcy
[132] key:swap
[133] url:zxlzcrdgcy
[134] key:insert
[135] url:scxlzddgcy
[136] key:delete

## 1.4.11 Modify one or multiple members of a sequence[137][138]

| | A | |
|---|---|---|
| 1 | **=[11,12,13,14]** | |
| 2 | **>A1(2)=6** | The value in **A1** is **[11,6,13,14]** |
| 3 | **>A1([3,4])=[7,8]** | The value in **A1** is **[11,6,7,8]** |
| 4 | **>A1.run(~=~%2)** | The value in A1 is [1,0,1,0] |

## 1.4.12 Modify sequence member(s) at specified position(s) and supply values at beyond-range position(s)[139][140]

| | A | |
|---|---|---|
| 1 | **=[11,12,13,14,15].modify(2,6)** | **[11,6,13,14,15]** |
| 2 | **=[11,12,13,14,15].modify(10,10)** | **[11,12,13,14,15,null,null,null,null,10]** |
| 3 | **=[11,12,13,14,15].modify(2,[7,8,9])** | **[11,7,8,9,15]** |

## 1.4.13 Insert a sequence as a member into another sequence[141][142]

| | A | |
|---|---|---|
| 1 | **[1,2,3,4]** | |
| 2 | **[5,6,7,8]** | |
| 3 | **=A1.insert(3,[A2])** | **[1,2,[5,6,7,8],3,4]** |

[137] url:xgxlddgcy
[138] key:modify, sequence, multiple
[139] url:xgzdwzdxlcyyjzzdb
[140] key:modify
[141] url:jxlzgzwcycrlyxl
[142] key:insert

## 1.4.14 Compare sequences in ASCII dictionary mode[143][144]

| | A | |
|---|---|---|
| 1 | **=cmp(["a","b","c"],["d","e","f"])** | **-1**, ASCII code for "a" is 1 less than that for "d" |
| 2 | **=cmp(["d","b","c"],["a","e","f"])** | **1**, ASCII code for "d" is 1 more than that for "a" |

## 1.4.15 Get inversed sequence[145][146]

| | A | |
|---|---|---|
| 1 | **[1,3,5,2,4,6]** | |
| 2 | **=A1.rvs()** | **[6,4,2,5,3,1]** |

## 1.4.16 Get & count unique members in a sequence[147][148]

| | A | |
|---|---|---|
| 1 | **=["a","c","d","e","f","a","a","b"]** | |
| 2 | **=A1.id()** | [a,b,c,d,e,f], remove duplicates and sort members |
| 3 | **=A1.id@o()** | [a,c,d,e,f,a,b], remove neighboring duplicate only |
| 4 | **=A1.id@u()** | [a,c,d,e,f,b], remove duplicates but won't perform sort |
| 5 | **=A1.icount()** | 6, count members after duplicates are deleted |
| 6 | **=A1.icount@o()** | 7, count members after neighboring duplicates are deleted |

143 url:zdsbjxldx
144 key:cmp
145 url:getinversedseq
146 key:inversed,sequence
147 url:getcountuniquemembers
148 key:count,unique,member

## 1.4.17 Stretch sequence to specified length[149][150]

| | A | |
|---|---|---|
| 1 | **[a,b,c,d,e,f]** | |
| 2 | **=A1.pad(null,9)** | [a,b,c,d,e,f,null,null,null], add members on the right side |
| 3 | **=A1.pad@l(null,9)** | [null,null,null,a,c,d,e,f,a,b], add members on the left side |

## 1.4.18 Sequence-related logic operations[151][152]

| | A | |
|---|---|---|
| 1 | **=[2<10,3>4,1!=1]** | **[true,false,false]** |
| 2 | **=A1.cand()** | **false** |
| 3 | **=[20,1<2].cand()** | **true** |
| 4 | **=[null,1<2].cand()** | **false** |
| 5 | **=[1!=1,3<2].cor()** | **false** |

## 1.5 Table Sequences and Records[153][154]

This chapter lists code examples of handling table sequences and records, including Create an empty table sequence, Judge if it is a record or a table sequence, List fields of a record, Access record fields and assign values to them, Get field number and number of fields, Fill members of a sequence into a record as field values, Get field value from a record by field number, Modify field value in a record by field number, Modify data structure of a table sequence, Replace record field values with fields, insert record(s) at specified position, Row-to-column & column-to-row transpositions, Delete record(s) and Reset table sequence.

## 1.5.1 Create an empty table sequence[155][156]

| | A | |
|---|---|---|
| 1 | **=create(fld1,fld2,fld3)** | Create an empty table sequence |
| 2 | **=A1.create()** | Create another empty table sequence using the structure of **A1** |

149 url:stretchseqtospecifiedlen
150 key:stretch,length
151 url:seqrelatedlogicoperations
152 key:sequence,logic
153 url:xbhjl
154 key:table sequence, record
155 url:cjkxb
156 key:create

## 1.5.2 Judge if it is a record or a table sequence[157][158]

| | A | |
|---|---|---|
| 1 | **[1,2,3]** | |
| 2 | **=create(fld1).record(A1)** | |
| 3 | **=ifr(A1(1))** | **false**, Judge if it is a record |
| 4 | **=ifr(A2(1))** | **true**, Judge if it is a record |
| 5 | **=ift(A1)** | **false**, Judge if it is a table sequence |
| 6 | **=ift(A2)** | **true**, Judge if it is a table sequence |

## 1.5.3 List fields of a record[159][160]

| | A | |
|---|---|---|
| 1 | =*r*.**fname()** | All fields in the table sequence in which the record resides |

## 1.5.4 Access record fields and assign values to them[161][162]

| | A | |
|---|---|---|
| 1 | =*r*.**Name** | Access by field name |
| 2 | =*r*.**#2** | Access by field number |
| 3 | >*r*.**#2="Tom"** | Assignment |
| 4 | >*r*.**Name="Tom"** | Assignment |

## 1.5.5 Get field number and number of fields[163][164]

| | A | |
|---|---|---|
| 1 | **=create(name, gender, job, age)** | |
| 2 | **=A1.fno(gender)** | 2, field number |
| 3 | **=A1.fno()** | 4, number of fields |

157 url:pdsfjlhxb
158 key:create,record,ifr,ift
159 url:lcjldzd
160 key:fname
161 url:fwjldzdhdqfz
162 key:access of fields, value assignment
163 url:qdzddxhhgs
164 key:create,fno

## 1.5.6 Fill members of a sequence into a record as field values[165][166]

| | A | |
|---|---|---|
| 1 | >*r*.**record([1,2,3])** | Fill members into the record in order |

## 1.5.7 Get field value from a record by field number[167][168]

| | A | |
|---|---|---|
| 1 | =*r*.**field(***i***)** | Get value of the *i*th field from record *r* |

## 1.5.8 Modify field value in a record by field number[169][170]

| | A | |
|---|---|---|
| 1 | >*r*.**field(***i***, *x*)** | Modify value of the *i*th field in record *r* into *x* |

## 1.5.9 Modify data structure of a table sequence[171][172]

| | A | |
|---|---|---|
| 1 | **=create(number,name,birthday)** | |
| 2 | **>A1.rename(number:id)** | Modify a field name |
| 3 | **>A1.rename(birthday)** | Delete field names |

[165] url:jxlyctrjlzwzdz
[166] key:record
[167] url:gjzdxhqcjldzdz
[168] key:field
[169] url:zdzdxhxgjl
[170] key:field
[171] url:xgxbdsjjg
[172] key:create,rename

## 1.5.10 Replace field values of a record with fields[173][174]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** | |
| 2 | **=A1.group(NAME)** | |
| 3 | **=A2.new(NAME,~.select@1(EVENT:"BalanceBeam").SCORE:BalanceBeam,~.select@1(EVENT:"Floor").SCORE:Floor)** | Replace record field values with fields |

## 1.5.11 Row-to-column & column-to-row transposition[175][176]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** | |
| 2 | **=A1.pivot(NAME;EVENT,SCORE)** | Row to column |
| 3 | **=A2.pivot@r(NAME;EVENT,SCORE)** | Column to row |

## 1.5.12 Reset table sequence[177][178]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.reset()** | Clean up table sequence |

# 1.6 Maintenance of Table Sequences & Record Sequences[179][180]

This chapter lists code examples of handling data maintenance in table sequences and record sequences, including Copy a table sequence entirely, Insert one or multiple null or non-null records into a table sequence, Remove one or multiple records from a table sequence, Modify field

[173] url:jjldzdzhczd
[174] query,group,new,select
[175] url:rowcoltransposition
[176] key:row,column,transposition
[177] url:resettseq
[178] key:reset,table sequence
[179] url:xbpldsjwh
[180] key:maintenance of table sequences & record sequences

values of one or multiple records, Modify the record at a specified position and fill up the position automatically if it is beyond range, Add a calculated column to a table sequence, Create a new table sequence based on the specified table sequence/record sequence, Combine table sequences or split a table sequence, Insert a sequence into the newly-created table sequence to generate new records, Get field values of a table sequence/record sequence and join them into a sequence, Add sequence members into a table sequence/record sequence as field values, Add sequence members to a table sequence/record sequence vertically as field values, Copy field values of a table sequence/record sequence into another table sequence/record sequence, and Modify values of specified field.

## 1.6.1 Copy a table sequence entirely[181][182]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT, SCORE from GYMSCORE")** | |
| 2 | **=A1.derive()** | Copy the table sequence entirely |
| 3 | **=A1.to(,)** | Copy as a record sequence only |

## 1.6.2 Insert record(s) to Tseq at specified position[183][184]

| | A | |
|---|---|---|
| 1 | **=create(ID,NAME,AGE)** | Create an empty table sequence |
| 2 | **>A1.insert(0,10,"Lucy",20)** | Insert a record at the end |
| 3 | **>A1.insert(1,1,"Jim",19)** | Insert a record at the first row |
| 4 | **>A1.insert(2:8,#+1,"Name"+string(#+1),rand(4)+18)** | Insert eight records starting from the second row |
| 5 | **=create(id,NAME,Age)** | Create an empty table sequence |
| 6 | **>A5.insert(0:5,#+10,"Name"+string(#+10),rand(4)+18)** | Insert 5 records at the end |
| 7 | **=A1.insert@r(0:A5)** | Insert A5's records to A1 |

[181] url:wzfzxb
[182] key:query,derive,to
[183] url:insertrecordatspecifiedposition
[184] key:insert,record,position

| | | |
|---|---|---|
| 8 | **=A1.insert@f(0:A5)** | Insert namesake fields in A5 to A1 |
| 9 | **>A1.insert(2)** | Insert an empty record before the second row |
| 10 | **>A1.insert(0:10,~:ID)** | Append 10 records at the end and set ID field values |

## 1.6.3 Remove one or multiple records from a table sequence[185][186]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME, EVENT, SCORE from GYMSCORE")** | |
| 2 | **>A1.delete(2)** | Remove the second record |
| 3 | **>A1.delete([4,6,1])** | Remove multiple records |
| 4 | **>A1.delete(A1.select(SCORE<15))** | Remove records based on specified conditions |

## 1.6.4 Modify field values of one or multiple records[187][188]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT, SCORE from GYMSCORE")** | |
| 2 | **>A1.run(SCORE+2:SCORE)** | Modify all the records |
| 3 | **>A1.select(EVENT=="Vault").run(SCORE+2:SCORE)** | Modify some of the records |

185 url:cxbcdtjl
186 key:query,delete,select
187 url:xgdtjldzdz
188 key:query,run,select

## 1.6.5 Modify a record at a specified position and fill up the position automatically if the record is beyond range[189][190]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT, SCORE from GYMSCORE")** | A table sequence |
| 2 | **=A1.modify(5,"":EVENT,15:SCORE)** | Modify the fifth record and fill up the out-of-range positions automatically |

## 1.6.6 Add a calculated column to a table sequence[191][192]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,UNITPRICE, QUANTITY from RECEIPT")** | |
| 2 | **=A1.derive(UNITPRICE*QUANTITY:Amount)** | Add the "**Amount**" field |

## 1.6.7 Create a new table sequence based on the specified table sequence/record sequence[193][194]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME, ABBR, CAPITAL,POPULATION from STATES")** | |
| 2 | **=A1.new(NAME, ABBR)** | Create a new table sequence based on the original one |
| 3 | **=A1.new(NAME:State, ABBR,CAPITAL:Capital)** | Change the field names |

189 url:xgzdwzdjlyjzzdb
190 key:query,modify
191 url:wxbtjjsl
192 key:query,derive
193 url:jyzdxbplcjxxb
194 key:query,new

## 1.6.8 Combine table sequences or split a table sequence[195][196]

| | A |
|---|---|
| 1 | **=demo.query("select * from STUDENTS1")** |
| 2 | **=demo.query("select * from STUDENTS2")** |
| 3 | **=A1\|A2** |

Concatenate table sequences A1 and A2

## 1.6.9 Insert a sequence into the newly-created table sequence to generate new records[197][198]

| | A | B |
|---|---|---|
| 1 | **1** | **Tom** |
| 2 | **2** | **Jack** |
| 3 | **3** | **Andy** |
| 4 | **=create(id,name)** | |
| 5 | **=A4.record([A1:B3])** | |

## 1.6.10 Get field values of a table sequence/record sequence and join them into a sequence[199][200]

| | A | B |
|---|---|---|
| 1 | **1** | **Tom** |
| 2 | **2** | **Jack** |
| 3 | **3** | **Andy** |
| 4 | **=create(id,name)** | |
| 5 | **=A4.record([A1:B3])** | |
| 6 | **=A5.field(2)** | |

Return the same sequence as **[B1:B3]**

195 url:hbhcfxb
196 key:query
197 url:jxltrxcjdxbcsxjl
198 key:create,record
199 url:jxbxldzdzqcpcxl
200 key:create,record,field

## 1.6.11 Add sequence members into a table sequence/record sequence as field values[201][202]

| | A | B |
|---|---|---|
| 1 | 1 | Tom |
| 2 | 2 | Jack |
| 3 | 3 | Andy |
| 4 | =create(id, name).insert(1:3) | |
| 5 | =A4.paste@a([A1:B3]) | |

## 1.6.12 Add sequence members to a table sequence/record sequence vertically as field values[203][204]

| | A | B | C | |
|---|---|---|---|---|
| 1 | 1 | 2 | 3 | |
| 2 | Tom | Jack | Andy | |
| 3 | =create(id, name).insert(1:3) | | | |
| 4 | =A3.modify(1:[A1:C1],~:#1) | | | The first column |
| 5 | =A3.modify(1:[A2:C2],~:#2) | | | The second column |

## 1.6.13 Copy field values of a table sequence/record sequence into another table sequence/record sequence[205][206]

| | A |
|---|---|
| 1 | =create(name,price) |
| 2 | =demo.query("select NAME,UNITPRICE, QUANTITY from RECEIPT") |
| 3 | =A1.modify@r(0:A2) |

[201] url:jxlcytrxbplzwzdz
[202] key:create,insert,paste
[203] url:jxlcysztrxbplzwzdz
[204] key:create,insert,modify
[205] url:clxbpldzdzdlyxbxl
[206] key:create,insert,query,paste

## 1.6.14 Modify values of specified field[207][208]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.field("SALARY",[8000,9000])** | Modify the first two values of SALARY field |
| 3 | **=A1.field(9,[8000,9000])** | Modify the first two values of the 9th field |

# 1.7 External Files[209][210]

This chapter lists code examples of handling external files, including Read/write a text file, Record log in a text file, Import a text file as a table sequence, and Store a table sequence as a text file.

## 1.7.1 Read/write a text file[211][212]

| | A | |
|---|---|---|
| 1 | **=file("D:\\test.txt")** | |
| 2 | **>A1.write("USA")** | Write a string into the file |
| 3 | **=A1.read()** | Read and return the file as a string |

## 1.7.2 Record log in a text file[213][214]

| | A | |
|---|---|---|
| 1 | **=file("D:\\test.log")** | |
| 2 | **>A1.write@a(string(now())+": Start getting data")** | "**@a**" indicates appending strings at the end |

[207] url:modifyvaluesofspecifiedfield
[208] key:modify,values,field
[209] url:wbwj
[210] key:external file
[211] url:dxwbwj
[212] key:file,write,read
[213] url:ywbwjjlrz
[214] key:file,write,string

## 1.7.3 Import a text file as a table sequence[215][216]

| | A |
|---|---|
| 1 | **=file("D:\\employee.txt")** |
| 2 | **=A1.import()** |
| 3 | **=A1.import@t()** |

The first row is used as field names

## 1.7.4 Store a table sequence as a text file[217][218]

| | A |
|---|---|
| 1 | **=demo.query("select EID,NAME,STATE, GENDER, BIRTHDAY,HIREDATE,DEPT, SALARY from EMPLOYEE")** |
| 2 | **=file("D:/employee.txt")** |
| 3 | **>A2.export(A1)** |
| 4 | **>A2.export@t(A1)** |

Set field names as the title

## 1.8 Database[219][220]

This chapter lists code examples of dealing with databases, including Retrieve data from database as table sequence via SQL, Return single value result of SQL computation, Use database stored procedure to return one or multiple table sequences, Run SQL statement over a database to modify data, Use program code to connect to and disconnect from database, Manage transaction commit automatically by program code, Get database error messages, Use cursor to fetch big data in batches, and Write a table sequence/record sequence into database.

[215] url:cwbwjzdqxb
[216] key:file,import
[217] url:ywbwjbcxb
[218] key:file,query,export
[219] url:sjk
[220] key:database

### 1.8.1 Retrieve data from database as table sequence via SQL[221][222]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=demo.query("select * from EMPLOYEE where EID=?",1)** |

### 1.8.2 Return single value result of SQL computation[223][224]

| | A |
|---|---|
| 1 | **=demo.query@1("select count(*) from EMPLOYEE")** |
| 2 | **=demo.query@1("select count(*) from EMPLOYEE where SALARY>?",10000)** |

### 1.8.3 Use database stored procedure to return one or multiple table sequences[225][226]

| | A | |
|---|---|---|
| 1 | **=*db*.proc("{call proc1(?,?)}",:101:"o":a,:101:"o":b)** | Execute the stored procedure and return 2 table sequences |
| 2 | **=A1(1)** | The first table sequence |
| 3 | **=A1(2)** | The second table sequence |
| 4 | **=a** | Use a variable name to access the first table sequence |

[221] url:csjkzySQLdrxb
[222] key:query
[223] url:fhySQLjsddzjg
[224] key:query
[225] url:ysjkdccgcfhdgxb
[226] key:proc

## 1.8.4 Run SQL statement over database to modify data[227][228]

| | A | |
|---|---|---|
| 1 | **>demo.execute("update SCORES set SCORE=? where STUDENTID=10",90)** | Update |
| 2 | **=demo.query("selct * from LIQUORSNEW")** | |
| 3 | **>demo.execute(A2,"update LIQUORS set STOCK=? where NAME=?", wineStock, wineName)** | Update in batches |
| 4 | **>demo.execute([1,3,5],"delete from product where productnumber=?",~)** | Delete in batches |
| 5 | **>demo.execute(A2,"insert into LIQUORS (LID, NAME, TYPE, PRODUCTION, STOCK) values (?,?)",wineID, wineName, wineType, wineProduction, wineStock)** | Insert in batches |

## 1.8.5 Use program code to connect to and disconnect from database[229][230]

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

## 1.8.6 Manage transaction commit automatically by program code[231][232]

| | A | B | |
|---|---|---|---|
| 1 | **=connect@e("demo")** | | Establish a connection |
| 2 | **=A1.execute@k(….)** | | |
| 3 | **=A1.error()** | | Read the error message generated by the previous database transaction execution |
| 4 | **if A3==0** | **>A1.commit()** | Commit if there are no errors |
| 5 | **else** | **>A1.rollback()** | Roll back if there are errors |
| 6 | **>A1.close()** | | Close the connection |

[227] url:lsjkzxSQLyjyxgsj
[228] key:execute,query
[229] url:ycxdmljhdksjk
[230] key:connect,close
[231] url:ycxdmzxglswdtj
[232] key:connect,execute,error,commit,rollback

## 1.8.7 Get database error messages[233][234]

| | A | |
|---|---|---|
| 1 | **=connect@e("demo")** | |
| 2 | **…** | |
| 3 | **=A1.error()** | Error code |
| 4 | **=A1.error@m()** | Error messages |

## 1.8.8 Use cursor to fetch big data in batches[235][236]

| | A | B | C | |
|---|---|---|---|---|
| 1 | **=demo.cursor("select * from STOCKRECORDS")** | | | |
| 2 | **for** | | | |
| 3 | | **=A1.fetch(1000)** | | Fetch **1,000** records and return them as a table sequence |
| 4 | | **if B3==null** | **break** | Break the loop when the data retrieving is finished |
| 5 | | **…** | | |

[233] url:qdsjkcwxx
[234] key:connect,error
[235] url:syybfpdrjdsj
[236] key:cursor,fetch,for,break

## 1.8.9 Write a table sequence/record sequence into database[237][238]

| | A | |
|---|---|---|
| 1 | **=demo.query("select ID, NAME,GENDER,AGE from STUDENTS")** | |
| 2 | **=A1.keys(ID)** | |
| 3 | **=demo.update(A1,STUDENTS1,ID, NAME)** | |
| 4 | **=demo.update@u(A1,STUDENTS1,ID, NAME)** | Generate "**update**" only |
| 5 | **=demo.update@i(A1,STUDENTS1, ID, NAME)** | Generate "**insert**" only |
| 6 | **=demo.update@a(A1,STUDENTS1, ID,NAME)** | Empty the target table before inserting data |

# 1.9 Program Logic[239][240]

This chapter lists code examples about program logic, including Join data in a cellset program into a sequence, Implement switch/case-like structure, Get loop count, Proceed to/exit outer loop, Delete used variables to free memory, Pass multiple parameters to subroutine, Return multiple values by subroutine, Write multi-line comments, Write a long statement in multiple cells, and use macro in code.

## 1.9.1 Join data in a program cellset into a sequence[241][242]

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **1** | **2** | **3** | **4** |
| 2 | **=[A1:D1]** | | | |

[237] url:jxbplxrsjk
[238] key:query,keys,update
[239] url:cxlj
[240] key:program logic
[241] url:jwgndjpcxl
[242] key:cellset, sequence

## 1.9.2 Implement switch/case-like structure[243][244]

| | A | B | |
|---|---|---|---|
| 1 | **=80** | | |
| 2 | **if A1>=90** | | |
| 3 | | **> A10="excellent"** | |
| 4 | **else if A1>=80** | | |
| 5 | | **> A10="good"** | |
| 6 | **else if A1>=60** | | |
| 7 | | **> A10="pass"** | |
| 8 | **else** | | |
| 9 | | **> A10="fail"** | |
| 10 | | | **good** |

## 1.9.3 Get loop count[245][246]

| | A | B | C |
|---|---|---|---|
| 1 | **for** | **if #A1==10000** | **break** |

## 1.9.4 Proceed to /exit outer loop[247][248]

| | A | B | C | D | |
|---|---|---|---|---|---|
| 1 | **for [3,2,1]** | | | | |
| 2 | | **for [5,1,3]** | | | |
| 3 | | | **if A1>B2** | **next A1** | Proceed to the next loop |
| 4 | | | **if A1== B2** | **break A1** | Exit the outer loop |

[243] url:sxlsscdjg
[244] key:if,else
[245] url:qddqxhyjxdcs
[246] key:for,break
[247] url:jxtcwcxh
[248] key:for,next,break

## 1.9.5 Delete used variables to free memory[249][250]

| | A |
|---|---|
| 1 | >var1=to(100) |
| 2 | =demo.query("select * from EMPLOYEE") |
| 3 | … |
| 4 | > var1=null,A2=null |

Delete variable **var1** and the value of cell **A2**

## 1.9.6 Pass multiple arguments to subroutine[251][252]

| | A | B |
|---|---|---|
| 1 | func | |
| 2 | | =A1 |
| 3 | | =B1 |
| 4 | | return B2+B3 |
| 5 | =func(A1,11,21) | |

Multiple arguments are arranged in order

32

## 1.9.7 Return multiple values by subroutine[253][254]

| | A | B |
|---|---|---|
| 1 | func | |
| 2 | | return [1,2,3,4] |
| 3 | | |
| 4 | =func(A1) | |

[249] url:qcsygdblyjsnc
[250] key:to,query
[251] url:xzcxcddgcs
[252] key:func,return
[253] url:zcxfhdgz
[254] key:func,return

## 1.9.8 Write multi-line comments[255][256]

| | A | B |
|---|---|---|
| 1 | //This is an example about… | |
| 2 | | 1. Remark 1… |
| 3 | | 2. Remark 2… |
| 4 | =1+3 | |

The words from line 1 to line 3 are all comments

## 1.9.9 Write a long statement in multiple cells[257][258]

| | A | B | C | D |
|---|---|---|---|---|
| 1 | 68 | | | |
| 2 | ==if(A1>100:"excellent",A1>80:"good",A1>60:"pass","fail") | | | |

## 1.9.10 Use macro in code[259][260]

| | A | B |
|---|---|---|
| 1 | [1,2,3,4] | |
| 2 | func | return A1.${A2}() |
| 3 | func | return A1.${lower(A3)}() |
| 4 | =func(A2,"sum") | |
| 5 | =func(A3,"Avg") | |

Return **A1.sum()**

Return **A1.avg()**

## 1.9.11 Cross-cellset call[261][262]

| | A |
|---|---|
| 1 | =size.(char(65+rand(26))).concat() |
| 2 | return A1 |

Parmeter *size* is passed in from external

Return result of A1

255 url:cpzsdxf
256 key:comment
257 url:jcyjxjdgdygz
258 key:long statement, multiple cells
259 url:zdmzsyh
260 key:func,return,lower
261 url:crosscellsetcall
262 key:cross-cellset,call

| | A |
|---|---|
| 1 | **=call(“D:/demo.dfx”,5)** |

Call subprogram *demo.dfx* to call pass in parameter 5

## 1.9.12 Parallel processing[263][264]

| | A | B | |
|---|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | | |
| 2 | **=A1.group(GENDER)** | | Group by gender |
| 3 | **fork A2** | **=A3.select(SALARY>10000)** | Get records by specified condition with multiple threads and concatenate results in A3 |

[263] url:parallelprocessing
[264] key:parallel processing

# Chapter2 Operations[265][266]

*Operations* lists code examples about topics in the following chapters, including Set, aggregate and loop operations, Searching and location, Sorting and location, Group operation, and Group and Join over associated tables.

## 2.1 Set, aggregte and loop operations[267][268]

This chapter lists code examples of handling sets, aggregate operations and loop operation, including Judge if it is an interger sequence, Judge if it is an ascending integer sequence, Judge if it is a member or a subset of an integer sequence, Judge if members from two sequences are equal, Perform union, intersection and difference on sorted sequences by merge function, Calculate quadratic sum and variance, Calculate weighted average, Calculate average of an integer sequence after removing the max and the min, Calculate link relative ratio over adjacent rows, Calculate moving average over adjacent sets, Aggregate loop, Union record sequences with different data structures, Set operations, Get topN, Synced segmentation where max/mix value in a segment is used to represent a value, Caculate product of three sets' Catesian products, and Get Fibonacci sequence.

### 2.1.1 Judge if it is an integer sequence[269][270]

| | A |
|---|---|
| 1 | **[1,2,3,4]** |
| 2 | **=A1.pselect(~!=int(~))==null** |

Check if any member is not integer

### 2.1.2 Judge if it is an ascending integer sequence[271][272]

| | A |
|---|---|
| 1 | **[1,2,3,4,5]** |
| 2 | **=A1.pselect(~!=int(~) \|\| ~<=~[-1] )==null** |

[265] url:ysp
[266] key:operation
[267] url:jhjhyxh
[268] key:set operations, aggregation, loop opertions
[269] url:pdsfssl
[270] key:pselect
[271] url:pdsfdzsl
[272] key:pselect

## 2.1.3 Judge if it is a member or a subset of an integer sequence[273][274]

| | A | |
|---|---|---|
| 1 | **[1,2,3,4,5,6,7]** | |
| 2 | **=A1.pos(2)!=null** | **true** for member |
| 3 | **=A1.pos@c([2,3])!=null** | **true** for continuous subset |
| 4 | **=A1.pos@c([3,2])!=null** | **false** |
| 5 | **=A1.pos@c([2,5])!=null** | **false** |
| 6 | **=A1.pos@i([2,3])!=null** | **true** for subset |
| 7 | **=A1.pos@i([3,2])!=null** | **false** |
| 8 | **=A1.pos@i([2,5])!=null** | **true** |

## 2.1.4 Judge if members from two sequences are equal[275][276]

| | A | |
|---|---|---|
| 1 | **[1,2,3]** | |
| 2 | **[3,2,1]** | |
| 3 | **=A1.eq(A2)** | **true** |

[273] url:pdsfcyhzj
[274] key:pos
[275] url:pdlgxldcysfxt
[276] key:eq

## 2.1.5 Perform union, intersection, and difference on sorted sequences by merge function[277][278]

| | A | |
|---|---|---|
| 1 | =demo.query("select CLASS,STUDENTID, SUBJECT, SCORE from SCORES where CLASS=? and SUBJECT=? and STUDENTID<?","Class one", "Math",10) | |
| 2 | =demo.query("select CLASS,STUDENTID, SUBJECT,SCORE from SCORES where CLASS=? and SUBJECT=? and STUDENTID>?","Class two", "Math",5) | |
| 3 | =A1.sort(STUDENTID) | |
| 4 | =A2.sort(STUDENTID) | |
| 5 | =[A3:A4].merge(STUDENTID) | Concatenation |
| 6 | =[A3:A4].merge@u(STUDENTID) | Union |
| 7 | =[A3:A4].merge@i(STUDENTID) | Intersection |
| 8 | =[A3:A4].merge@d(STUDENTID) | Difference |

## 2.1.6 Calculate quadratic sum and variance[279][280]

| | A | |
|---|---|---|
| 1 | [1,2,3,4,5,6,7,8] | |
| 2 | =A1.sum(~*~) | Quadratic sum |
| 3 | =A1.variance() | Variance |
| 4 | =demo.query("select NAME,EVENT,SCORE from GYMSCORE") | |
| 5 | =A4.variance(SCORE) | |

## 2.1.7 Calculate weighted average[281][282]

| | A |
|---|---|
| 1 | [9,9.1,8.5,9.8,9.4] |
| 2 | [0.9,0.8,1.0,0.95,1.0] |
| 3 | =(A1**A2).sum()/A2.sum() |

277 url:dyxxlygbfkszbjc
278 key:query,sort,merge
279 url:jspfhfc
280 key:sum,variance
281 url:jsjqpj
282 key:sum

## 2.1.8 Calculate average of an integer sequence after removing the max and the min[283][284]

| | A |
|---|---|
| 1 | [99,98,95,93,87,89,90,96,94] |
| 2 | =(A1.sum()-A1.max()-A1.min())/(A1.len()-2) |
| 3 | =(A1\A1.min()\A1.max()).avg() |

## 2.1.9 Calculate link relative ratio over adjacent rows[285][286]

| | A |
|---|---|
| 1 | [1,2,3,4,5,6] |
| 2 | =A1.(~/~[-1]-1) |
| 3 | =demo.query("select DATE,sum(CLOSING) AMOUNT from STOCKRECORDS GROUP BY DATE") |
| 4 | =A3.derive(AMOUNT/AMOUNT[-1]-1: Period-over-period) |

## 2.1.10 Calculate moving average over adjacent sets[287][288]

| | A |
|---|---|
| 1 | [1,2,3,4,5,6] |
| 2 | =A1.(~[-1,1].avg()) |
| 3 | =demo.query("select STOCKID, DATE,CLOSING from STOCKRECORDS where STOCKID=?","000062") |
| 4 | =A3.(CLOSING[-3,3].avg()) |

283 url:qdzdzhzxzdpj
284 key:sum,max,min,len,avg
285 url:qxlhjsbsq
286 key:query,derive
287 url:qxljhjsydpj
288 key:query,avg

## 2.1.11 Aggregate loop[289][290]

| | A | |
|---|---|---|
| 1 | [1,2,3,4,5,6,7] | |
| 2 | =A1.iterate(~*~~;1) | Return the product |

## 2.1.12 Union record sequences with different data structures[291][292]

| | A |
|---|---|
| 1 | =demo.query("select * from STUDENTS") |
| 2 | =demo.query("select * from EMPLOYEE").derive(interval@y(BIRTHDAY,now()):AGE) |
| 3 | =A1\|A2 |
| 4 | =A3.select(GENDER:"F").avg(AGE) |

## 2.1.13 Set operations[293][294]

| | A | B | |
|---|---|---|---|
| 1 | [1,1,2,3,5,8] | | |
| 2 | [1,2,3,4,5,6] | | |
| 3 | =[A1,A2].isect() | =A1^A2 | Intersection |
| 4 | =[A1,A2].diff() | =A1\A2 | Difference |
| 5 | =[A1,A2].union() | =A1&A2 | Union |
| 6 | =[A1,A2].conj() | =A1\|A2 | Concatenation |
| 7 | =[A1,A2].xunion() | | Retain unique members of every sequence |
| 8 | =demo.query("select * from EMPLOYEE") | | |
| 9 | =A8.select(GENDER:"F") | | |

289 url:hzxh
290 key:iterate
291 url:btsjjgdplzbj
292 key:query,derive,select,avg
293 url:setoperations
294 key:set operations

| | | | |
|---|---|---|---|
| 10 | =A8.select(DEPT:"Sales ") | | |
| 11 | =A8.select(age(BIRTHDAY)>=40) | | Perform set operations on table sequences |
| 12 | =[A9,A10,A11].isect() | =A9^A10^A11 | |
| 13 | =[A9,A10,A11].diff() | =A9\A10\A11 | |
| 14 | =[A9,A10,A11].union() | =A9&A10&A11 | |
| 15 | =[A9,A10,A11].conj() | =A9\|A10\|A11 | |
| 16 | =[A9,A10,A11].xunion() | | |

## 2.1.14 Get topN[295][296]

| | A | |
|---|---|---|
| 1 | =100.(rand(100)) | |
| 2 | =A1.top(5,~) | Get topN in ascending order |
| 3 | =A1.top(-5,~) | Get topN in descending order |
| 4 | =demo.query("select * from EMPLOYEE ") | |
| 5 | =A4.top(5,SALARY) | Get 5 lowest SALARY values |
| 6 | =A4.top(-5;SALARY) | Get records holding 5 highest SALARY values |

## 2.1.15 Alignment arithmetic operations[297][298]

| | A | |
|---|---|---|
| 1 | [1,2,3] | |
| 2 | [5,6,7] | |
| 3 | =A1++A2 | alignment addition |
| 4 | =A1--A2 | alignment subtraction |
| 5 | =A1**A2 | alignment multiplication |
| 6 | =A1//A2 | alignment division |
| 7 | =A1%%A2 | mod operation |

[295] url:gettopn
[296] key:topN
[297] url:alignarithmeticoperation
[298] key:alignment

## 2.1.16 Synced segmentation where max/mix value is used to represent a value[299][300]

| | A | |
|---|---|---|
| 1 | **=to(100).sort(rand())** | |
| 2 | **=A1.median(:4)\|A1.max()** | Define the number of segments |
| 3 | **=A1.((n=~,A2.pseg@r(n)))** | Find which segment each value belongs to |
| 4 | **=A3.(A2(~+1))** | Represent the value with max value in its segment |
| 5 | **=A1.min()\|A1.median(:4)** | Define the number of segments |
| 6 | **=A1.((n=~,A5.segp(n)))** | Represent the value with min value in its segment |

## 2.1.17 Get product of three sets' Cartesian products[301][302]

| | A | |
|---|---|---|
| 1 | **[1,2,3]** | |
| 2 | **[2,3,4]** | |
| 3 | **[3,4,5]** | |
| 4 | **=A1.((x=~,A2.((y=~,A3.(x*y*~)))))** | Calculate Cartesian products in a triplevel loop |

## 2.1.18 Calculate Fibonacci sequence[303][304]

| | A |
|---|---|
| 1 | **=10.iterate([~~(2),~~(1)+~~(2)],[1,1])** |

[299] url:syncedsegmentation
[300] key:synced,segmentation
[301] url:getproductofCartesianproducts
[302] key:product, Cartesian product
[303] url:Finbonacciseq
[304] key:Fibonaccis sequence

## 2.2 Searching and Location[305][306]

This chapter lists code examples about data searching and location, including Locate a member, Locate a sub-sequence, Locate members matching specified conditions and return their positions, Find members matching specified conditions from back to front, Stop searching after the first/all members matching specified conditions are found, Search from the $K^{th}$ member, Query on multiple fields, Speed up query on sorted sequence/record sequence via binary search, Aggregates on query results, Filter a table sequence, Perform query on a table sequence/record sequence by primary key value, Get the record with max/min value and its position, and calculate link relative ratio for selected members.

### 2.2.1 Locate a member[307][308]

| | A | |
|---|---|---|
| 1 | [3,2,1,5,1] | |
| 2 | =A1.pos(1) | 3 |
| 3 | =A1.pos@a(1) | [3,5] |

### 2.2.2 Locate a sub-sequence[309][310]

| | A | B | |
|---|---|---|---|
| 1 | [1,2,3,4,5] | [4,5] | |
| 2 | =A1.pos@c(B1) | | 4 |

[305] url:jsydw
[306] key:searching, location
[307] url:zccydwz
[308] key:pos
[309] url:zczxldwz
[310] key:pos

## 2.2.3 Locate members matching specified conditions and return their positions[311][312]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME, STATE, GENDER, BIRTHDAY,HIREDATE,DEPT, SALARY from EMPLOYEE")** | |
| 2 | **=A1.select(GENDER=="M")** | Locate all members matching the specified condition |
| 3 | **=A1.pselect(GENDER=="M")** | Stop searching after the first matching member is found, and return the member position |

## 2.2.4 Find members matching specified conditions from back to front[313][314]

| | A |
|---|---|
| 1 | **=demo.query("select EID, NAME,STATE, GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE")** |
| 2 | **=A1.select@z(GENDER=="M")** |
| 3 | **=A1.pselect@z(GENDER=="M")** |

311 url:zcmztjdcyjszwz
312 key:query,select,pselect
313 url:chxqz
314 key:query,select,pselect

## 2.2.5 Stop searching after the first/all members matching specified conditions are found[315][316]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT, SCORE from GYMSCORE")** | |
| 2 | **=A1.pselect(EVENT:"UnevenBars")** | Stop searching after the first matching member is found |
| 3 | **=A1(A2).SCORE** | Scores on a specified event |
| 4 | **=demo.query("select EID,NAME,STATE, GENDER,BIRTHDAY, HIREDATE,DEPT, SALARY from EMPLOYEE")** | |
| 5 | **=A4.select(GENDER=="M")** | Find all matching members |
| 6 | **=A4.select@1(GENDER=="M")** | Find the first matching member |
| 7 | **=A4.pselect@a(GENDER=="M")** | Locate all matching members |

## 2.2.6 Search from the K$^{th}$ member[317][318]

| | A |
|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER,BIRTHDAY, HIREDATE,DEPT, SALARY from EMPLOYEE")** |
| 2 | **=A1.pselect(GENDER=="M",8)** |

315 url:zddygjtzzcsyd
316 key:query,select,pselect
317 url:cdkgcyksz
318 key:query,pselect

## 2.2.7 Query on multiple fields[319][320]

| | A |
|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER,BIRTHDAY, HIREDATE, DEPT, SALARY from EMPLOYEE")** |
| 2 | **=A1.select(GENDER:"M",DEPT:"R&D")** |
| 3 | **=A1.pselect(GENDER:"M",DEPT:"R&D")** |
| 4 | **=A1.pselect@a(GENDER:"M",DEPT:"R&D")** |

## 2.2.8 Speed up query on sorted sequence/record sequence via binary search[321][322]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE order by GENDER,DEPT")** |
| 2 | **=A1.select@b(GENDER:"M",DEPT:"R&D")** |
| 3 | **=A1.pselect@b(GENDER:"M",DEPT:"R&D")** |

## 2.2.9 Aggregates on query results[323][324]

| | A |
|---|---|
| 1 | **=demo.query("select NAME,UNITPRICE,QUANTITY from RECEIPT")** |
| 2 | **=A1.select(NAME:"Apple").sum(UNITPRICE*QUANTITY)** |

[319] url:zddgzdjs
[320] key:query,select,pselec
[321] url:dyjyxdxlplsyefftgjssd
[322] key:query,select,pselect
[323] url:zdjshdjghz
[324] key:query,select,sum

## 2.2.10 Filter a table sequence[325][326]

| | A |
|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER,BIRTHDAY, HIREDATE, DEPT, SALARY from EMPLOYEE")** |
| 2 | **=A1.select(SALARY<6300)** |
| 3 | **=A1.select(SALARY>5000)** |

## 2.2.11 Perform query on a table sequence/record sequence by primary key value[327][328]

| | A |
|---|---|
| 1 | **=demo.query("select * from SCORES")** |
| 2 | **=A1.keys(CLASS,STUDENTID)** |
| 3 | **=A1.pfind(["Class one",2])** |
| 4 | **=A1(A3)** |
| 5 | **=A1.find(["Class one",2])** |

## 2.2.12 Get the record with max/min value and its position[329][330]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE ")** | |
| 2 | **=A1.sort(HIREDATE)** | Sort |
| 3 | **=A2.pmax(BIRTHDAY)** | The position of the youngest employee |
| 4 | **=A2(to(A3-1))** | The records of employees employed earlier |
| 5 | **=A4.minp(BIRTHDAY).NAME** | The name of the oldest employee |

[325] url:dxbzgl
[326] key:query,select
[327] url:syzjzzplxbzjsjl
[328] key:query,keys,pfind,find
[329] url:zczdzzxzszjlhwz
[330] key:query,sort,pmax,to,minp

## 2.2.13 Calculate link relative ratio for selected members[331][332]

| | A | B | |
|---|---|---|---|
| 1 | **=demo.query("select * from STOCKRECORDS")** | | |
| 2 | **=A1.pselect@a(CLOSING>10)** | | Locate the **DATE**s on which the closing prices are greater than 10 |
| 3 | **=A1.calc(A2, CLOSING/CLOSING[-1]-1)** | | Calculate the corresponding increase |
| 4 | **==A2.new(** | **A1(~).DATE:Date,** | Output the result set |
| 5 | | **A1(~).CLOSING:ClosingPrice,** | |
| 6 | | **A3(#):Increase)** | |

## 2.2.14 Get positions of topN[333][334]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE ")** | |
| 2 | **=A1.ptop(5,BIRTHDAY)** | Get sequence numbers of 5 eldest people |

## 2.3 Sorting and Location[335][336]

This chapter lists code example about data sorting and location, including Get members in odd positions, Calculate ranking, Members ranking top 10, the 3rd and the 2nd from the bottom of the sequence, and median, Calculate link relative ration for top 3, Members ranking at top 20% and middle 50%, Select 10 members randomly, Calculate max continuous interval, Sort a table sequence, Sort by specified order, and Create binary search index for a record sequence.

331 url:zdxccyjsbsq
332 key:query,pselect,calc,new
333 url:getposoftopN
334 key:position,topN
335 url:pxydw
336 key:sorting, location

## 2.3.1 Get members in odd positions[337][338]

| | A | |
|---|---|---|
| 1 | **[1,2,3,4,5,6,7,8,9,10]** | |
| 3 | **=A1.step(2,1)** | **[1,3,5,7,9]** |

## 2.3.2 Calculate ranking[339][340]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT, SCORE from GYMSCORE")** | |
| 2 | **=A1.ranks@z(SCORE)** | Calculate the ranking of all scores |
| 3 | **=A1.rank@z(16, SCORE)** | Ranking of **16** points |
| 4 | **=[99,98,97,96,93,87,99,95].rank@z(98)** | Ranking of **98** in the integer sequence |
| 5 | **=[99,98,97,96,93,87,99,95].rank@sz(98)** | Find ranking for 98; won't ignore duplicate members |
| 6 | **=demo.query("select * from SCORES ")** | |
| 7 | **=A6.pivot(CLASS,STUDENTID;SUBJECT,SCORE)** | Column-to-row transposition |
| 8 | **=A7.sort(CLASS,-English)** | Sort by class and English in descending order |
| 9 | **=A8.derive(rank(English;CLASS):RANK)** | |

[337] url:qcjswzdcy
[338] key:step
[339] url:jspm
[340] key:query,ranks,rank

## 2.3.3 Members ranking top 10, the 3rd, the 2nd from the bottom of the sequence, and median[341][342]

| | A | |
|---|---|---|
| 1 | =demo.query("select NAME, EVENT,SCORE from GYMSCORE") | |
| 2 | =A1.sort(-SCORE) | |
| 3 | =A2(to(10)) | Members with scores ranking top **10** |
| 4 | =A2.m([3,-2]) | The member with the score ranking the third, and the one with the score ranking the second-to-last |
| 5 | =round(A2.len()/2) | |
| 6 | =A2(A5) | Median score |

## 2.3.4 Calculate link relative ratio for top 3[343][344]

| | A | |
|---|---|---|
| 1 | =demo.query("select * from STOCKRECORDS where STOCKID=?","000062") | |
| 2 | =A1.sort(DATE) | Sort by **DATE** |
| 3 | =A2.psort(-CLOSING) | Sort by **CLOSING** |
| 4 | =A3(to(3)) | The sequence numbers of records for the three dates with highest closing prices |
| 5 | =A4.(A2.calc(A4.~, CLOSING-CLOSING[-1])) | Calculate the increases growth rates of the three days |

[341] url:q10md3mdsd2mzws
[342] key:query,sort,to,m,round
[343] url:zdq3mjsbsq
[344] key:query,sort,psort,to,calc

## 2.3.5 Members ranking at top 20% and middle 50%[345][346]

| | A | |
|---|---|---|
| 1 | =demo.query("select NAME, EVENT,SCORE from GYMSCORE") | |
| 2 | =A1.sort(-SCORE) | |
| 3 | =A2.len() | Total number of people |
| 4 | =round(A3*0.2) | The number of members ranking at the top 20% |
| 5 | =A2(to(A4)) | Members ranking at the top 20% |
| 6 | =round(A3*0.25) | Positions of members ranking at the first 25% of the middle |
| 7 | =round(A3*0.75) | Positions of members ranking at the last 25% of the middle |
| 8 | =A2(to(A6,A7)) | Records of members ranking at the middle 50% |
| 9 | =A8(1) | The highest score of middle ranking members |
| 10 | =A8.m(-1) | The lowest score of middle ranking members |

## 2.3.6 Select 10 members randomly[347][348]

| | A |
|---|---|
| 1 | =demo.query("select NAME,EVENT,SCORE from GYMSCORE") |
| 2 | =A1.sort(rand())(to(10)) |

## 2.3.7 Calculate max continuous interval[349][350]

| | A | |
|---|---|---|
| 1 | =demo.query("select * from STOCKRECORDS where STOCKID=?","000062") | |
| 2 | =A1.sort(DATE) | |
| 3 | =A2.max(a=if(CLOSING/CLOSING[-1]>=1.05, a+1,0)) | The max continuous interval (days) when the increase is greater than 5% |

345 url:q20dcyz50dcy
346 key:query,sort,len,round,to,m
347 url:sjq10gcy
348 key:query,sort,rand,to
349 url:jszdlxqj
350 key:query,sort,max

## 2.3.8 Sort a table sequence[351][352]

| | A |
|---|---|
| 1 | **=demo.query("select * from SCORES").sort(-SCORE)** |
| 2 | **=demo.query("select * from SCORES").psort(-SCORE)** |

## 2.3.9 Sort by specified order[353][354]

| | A |
|---|---|
| 1 | **[CA,IL,KY,CO,NY]** |
| 2 | **=demo.query("select NAME,ABBR,CAPITAL,POPULATION from STATES")** |
| 3 | **=A2.align(A1,ABBR)** |

## 2.3.10 Create binary search index for a record sequence[355][356]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME, EVENT,SCORE from GYMSCORE")** | |
| 2 | **=A1.sort(-SCORE)** | Create a binary search index |
| 3 | **=A2.select@b(SCORE:14.175)** | Return matching records after the binary search is completed |
| 4 | **=A1.psort(-SCORE)** | Create an index |
| 5 | **=A1(A4).pselect@b(SCORE:14.175)** | |
| 6 | **=A4(A5)** | Return the sequence number of the record matching the specified condition, after the binary search is completed |

[351] url:dxbzpx
[352] key:query,sort,psort
[353] url:asxgddcxpx
[354] key:query,align
[355] url:wpljlefczsy
[356] key:query,sort,select,psort,pselect

# 2.4 Group operation[357][358]

This chapter lists code examples about common group operations, including Get the distinct value of a field, Delete duplicate members, Delete duplicate adjacent members, Concatenate grouping results into a table sequence, Set five members in each group, Compute aggregate value afer grouping, Ge subset of the grouping result, Refilter or re-sort grouping result, Refilter or re-sort subsets of grouping result, Regroup subsets of grouping result, Perform intragroup cross-row calculation, Get a specified member from each grouped subset, Find the most appeared member, Get topN from each subgroup, Find continuous array, Group by neighboring same key values, Create a new group for a different condition, and Group & aggregate iteratively.

## 2.4.1 Get the distinct value of a field[359][360]

| | A |
|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** |
| 2 | **=A1.id(EVENT)** |

## 2.4.2 Delete duplicate members[361][362]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,TYPE, PRODUCTION from LIQUORS")** | |
| 2 | **=A1.id@u(TYPE)** | Delete repeated members without changing the member order |
| 3 | **=A1.group@1u(TYPE)** | |
| 4 | **=[1,2,2,3,3,4,5,6,2,3].id@u()** | **[1,2,3,4,5,6]** |

[357] url:cgfz
[358] key:group operation
[359] url:zczddwyz
[360] key:query,id
[361] url:sccfcy
[362] key:query,id,group

## 2.4.3 Delete duplicate adjacent members[363][364]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,TYPE, PRODUCTION from LIQUORS")** | |
| 2 | **=A1.id@o(TYPE)** | Delete repeated adjacent members without sorting |
| 3 | **=A1.group@1o(TYPE)** | |
| 4 | **=[1,2,5,5,3,4,5,6,2,3].id@o()** | **[1,2,5,3,4,5,6,2,3]** |

## 2.4.4 Concatenate grouping results into a table sequence[365][366]

| | A |
|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** |
| 2 | **=A1.group@s(EVENT)** |
| 3 | **=A1.group(EVENT)** |
| 4 | **=A2.conj()** |

## 2.4.5 Set five members in each group[367][368]

| | A |
|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** |
| 2 | **=A1.group(int((#-1)/5))** |

[363] url:qcxldcfcy
[364] key:query,id,group
[365] url:jfzdjgzhbcxb
[366] key:query,group,conj
[367] url:m5gcyfwyz
[368] key:query,group,int

## 2.4.6 Compute aggregate value after grouping[369][370]

| | A |
|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** |
| 2 | **=A1.groups(EVENT:GymEvent; sum(SCORE):TotalScore)** |
| 3 | **=A1.group(EVENT)** |
| 4 | **=A3.new(EVENT: GymEvent,~.sum(SCORE): TotalScore)** |

## 2.4.7 Get subset of the grouping result[371][372]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME,STATE, GENDER, BIRTHDAY,HIREDATE, DEPT,SALARY from EMPLOYEE")** | |
| 2 | **=A1.group(DEPT)** | |
| 3 | **=A2.maxp(~.avg(age(BIRTHDAY)))** | The records of employees in a department whose average age is the highest |

[369] url:jsfzhhzz
[370] key:query,groups,group,new,sum
[371] url:qcfzjgdzj
[372] key:query,group,maxp,avg

## 2.4.8 Refilter or re-sort grouping result[373][374]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT,SCORE from GYMSCORE")** | |
| 2 | **=A1.group(EVENT)** | |
| 3 | **=A2.select(~.avg(SCORE)>14.3)** | |
| 4 | **=A3.sort(-(~.avg(SCORE)))** | |
| 5 | **=A4(to(2))** | Top **2** events with the max average score |
| 6 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE")** | |
| 7 | **=A6.groups(DEPT;count(age(BIRTHDAY)>40):Number)** | |
| 8 | **=A7.select(Number>=20).(DEPT)** | Department with more than 20 employees over their 40s |

## 2.4.9 Refilter or re-sort subsets of grouping result[375][376]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,EVENT, SCORE from GYMSCORE")** | |
| 2 | **=A1.group(EVENT)** | |
| 3 | **>A2.(~=~.sort(-SCORE))** | |
| 4 | **>A2.(~=~(to(2)))** | |
| 5 | **=A2.(~.(NAME)).isect()** | Athletes with all event scores at the top **2** |

[373] url:dfzjgzglhpxq
[374] key:query,group,sort,select,count
[375] url:dfzjgdzjzglhpx
[376] key:query,group,sort,to,isect

## 2.4.10 Regroup subsets of grouping result[377][378]

| | A | |
|---|---|---|
| 1 | =demo.query("select EID,NAME,STATE,GENDER,BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE") | |
| 2 | =A1.group(DEPT) | |
| 3 | >A2.(~=~.group(month(BIRTHDAY),day(BIRTHDAY))) | Regroup the subsets |
| 4 | =A2.maxp(~.count()) | |
| 5 | =A4(1).DEPT | |

## 2.4.11 Perform intragroup cross-row calculation[379][380]

| | A | B | C | |
|---|---|---|---|---|
| 1 | =demo.query("select * from STOCKRECORDS") | | | |
| 2 | =A1.group(STOCKID).(~.sort(DATE)) | | | |
| 3 | for A2 | =0 | | |
| 4 | | if A3.pselect(B3=if( CLOSING/CLOSING[-1] >=1.05,B3+1,0):4)>0 | | Limit up for three days |
| 5 | | | =C5\|A3.STOCKID | Stores the result |

[377] url:dfzjgdzjzzfz
[378] key:query,group,maxp,month,day
[379] url:zdzncyzkhjs
[380] key:query,group,pselect

## 2.4.12 Get a specified member from each grouped subset[381][382]

| | A | |
|---|---|---|
| 1 | **=demo.query("select NAME,TYPE,PRODUCTION from LIQUORS")** | |
| 2 | **=A1.group(TYPE).new(TYPE,~.m(-1):Last)** | Group and aggregate directly |
| 3 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE ")** | |
| 4 | **=A3.group(DEPT)** | Group first |
| 5 | **=A4.(~.minp(SALARY))** | Then aggregate |

## 2.4.13 Find the most appeared member[383][384]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE ")** | |
| 2 | **=A1.group(DEPT)** | Group |
| 3 | **=A2.maxp(~.count())** | Find the group with the most employees |
| 4 | **=A3(1).DEPT** | The **DEPT** with the most employees |

## 2.4.14 Get topN from each subgroup[385][386]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE ")** | |
| 2 | **=A1.groups(DEPT;top(3;BIRTHDAY):TOP3_EMP)** | Get 3 eldest employees in each department |

[381] url:qcmgfzzjzdmgcy
[382] key:query,group,new,minp
[383] url:zccxcszddcy
[384] key:query,group,maxp,count
[385] url:gettopNfromsubgroup
[386] key:topN,subgroup

## 2.4.15 Find continuous array[387][388]

| | A |
|---|---|
| 1 | [1,3,4,5,8,9,15,16,20] |
| 2 | =A1.group(~-#).select(~.len()>1) |

## 2.4.16 Group by neighboring same key values[389][390]

| | A |
|---|---|
| 1 | =demo.query("select* from SCORES") |
| 2 | =A1.groups@o(CLASS,STUDENTID;sum(SCORE):SCORE) |

## 2.4.17 Create new group for a different condition [391][392]

| | A | |
|---|---|---|
| 1 | =demo.query("select* from STOCKRECORDS") | Order by date |
| 2 | =A1.select(STOCKID:"000062") | Select stock 000062 |
| 3 | =A2.group@i(CLOSING<CLOSING[-1]) | Create a new group if condition is true |
| 4 | =A3.max(~.len())-1 | Find number of continuous rising days for the stock |

## 2.4.18 Group & aggregate iteratively [393][394]

| | A | |
|---|---|---|
| 1 | =file("E:/txt/orders_i.csv").import@t() | |
| 2 | =A1.group(sellerid;(~.iterate((x=#,~~+amount),0,~~>500000),x):breach50) | Find how many months when each salesperson achieves sales of over 50,000 |

[387] url:findcontinuousarray
[388] key:continuous,array
[389] url:groupbyneigboringsamevalues
[390] key:group,neigboring,same value
[391] url:newgroupfordifferentcondition
[392] key:new,group,different,condition
[393] url:groupaggregateiteratively
[394] key:group,aggregate,iteratively

## 2.5 Group & Join over associative tables[395][396]

This chapter lists code examples about group & join operations over associative tables, including Group by specified category, Group by specified ranges, Conditional grouping with possible overlapped ranges, Join table on equivalence conditions, Join tables based on the first one (left join), Join records even if specified conditions are not matched (full join), Align tables on condition that specified fields in them are equal, Perform a join under non-equal conditions, Perform unconditional join (full cross join), Convert foreign key references into record fields, Convert members of a subtable into table sequence fields, and Form a wide table.

### 2.5.1 Group by specified category[397][398]

| | A |
|---|---|
| 1 | **[America,Jamaica,France,Scotland,England]** |
| 2 | **=demo.query("select NAME,TYPE, PRODUCTION from LIQUORS")** |
| 3 | **=A2.align@a(A1,PRODUCTION)** |

Group by **PRODUCTION**

### 2.5.2 Group data by specified ranges[399][400]

| | A | B |
|---|---|---|
| 1 | **?<25** | **Below 25** |
| 2 | **?>=25 && ?<=30** | **25 to 30** |
| 3 | **?>30 && ?<=40** | **30 to 40** |
| 4 | **?>40 && ?<=50** | **40 to 50** |
| 5 | **?>50** | **over 50** |
| 6 | **=create(Section,AgeGroup).record([A1:B5])** | |
| 7 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT, SALARY from EMPLOYEE").derive(age(BIRTHDAY):AGE)** | |
| 8 | **=A7.enum@r(A6.(Section),AGE)** | |
| 9 | **=A8.new(A6(#).AgeGroup:AgeGroup, ~.count():Number, ~.avg(AGE):AverageAge)** | |

Group by **AGE**

395 url:glfzylj
396 key:associative grouping operation, joins
397 url:azddflfz
398 key:query,align
399 url:azddtjfwfz
400 key:create,record,query,enum,new,derive

## 2.5.3 Conditional grouping with possible overlapped ranges[401][402]

| | A |
|---|---|
| 1 | **[?<5000,?>=7000,?>10000]** |
| 2 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT, SALARY from EMPLOYEE")** |
| 3 | **=A2.enum@r(A1,SALARY)** |

## 2.5.4 Join tables on equivalence conditions[403][404]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATES")** |
| 2 | **=demo.query("select * from EMPLOYEE")** |
| 3 | **=join(A1:State,NAME;A2:Employee,STATE)** |

## 2.5.5 Join tables based on the first one (left join)[405][406]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATES")** |
| 2 | **=demo.query("select * from EMPLOYEE")** |
| 3 | **=join@1(A1:State,NAME;A2:Employee,STATE)** |

401 url:fwkncddtjfz
402 key:query,enum
403 url:sydztjjxlj
404 key:query,join
405 url:adygbwjzzljzlj
406 key:query,join

## 2.5.6 Join records even if specified conditions are not matched (full join)[407][408]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATES")** |
| 2 | **=demo.query("select * from EMPLOYEE")** |
| 3 | **=join@f(A1:State,NAME;A2:Employee,STATE)** |

## 2.5.7 Align tables on condition that specified fields in them are equal[409][410]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=demo.query("select * from ATTENDANCE")** |
| 3 | **=demo.query("select * from PERFORMANCE")** |
| 4 | **=join@1(A1:Employee,EID;A2:Attencance,EMPLOYEEID; A3:Performance,EMPLOYEEID)** |

## 2.5.8 Perform a join under non-equal conditions[411][412]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATES")** |
| 2 | **=demo.query("select * from CITIES")** |
| 3 | **=demo.query("select * from GYMSCORE")** |
| 4 | **=xjoin(A1:State,left(NAME,1)=="A";A2:City,POPULATION> 1000000;A3:Score,EVENT=="Floor")** |

[407] url:ljbnppdjlqlj
[408] key:query,join
[409] url:jdgbamzdzxdtjdq
[410] key:query,join
[411] url:fdztjdyblj
[412] key:query,xjoin

## 2.5.9 Perform unconditional join (full cross join)[413][414]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATES")** |
| 2 | **=demo.query("select * from STUDENTS")** |
| 3 | **=xjoin(A1:State;A2:Student)** |

## 2.5.10 Convert foreign key references into record fields[415][416]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from CITIES").keys(CID)** | |
| 2 | **=demo.query("select * from STATES where STATEID<?",51).keys(STATEID)** | |
| 3 | **=A1.switch(STATEID,A2)** | Create a reference between the main table and a subtable |
| 4 | **=A1.group(STATEID.REGIONID)** | Directly access the main table via reference fields |
| 5 | **=A2.run(CAPITAL=A1.select@1(NAME==CAPITAL))** | |
| 6 | **=A1.new(NAME,STATEID.CAPITAL.NAME:StateCapital)** | |
| 7 | **=A1.select(STATEID.CAPITAL.POPULATION>1000000)** | |

413 url:wtjdljjwqjc
414 key:query,xjoin
415 url:jwjyyzcjlxzd
416 key:query,keys,switch,group,run,select

## 2.5.11 Convert members of a subtable into table sequence fields[417][418]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER, BIRTHDAY,HIREDATE,DEPT,SALARY from EMPLOYEE")** | |
| 2 | **=demo.query("select * from FAMILY where RELATION=?","child")** | |
| 3 | **=A1.select(GENDER=="F" && A2.id(EID).pos(EID)>0)** | |
| 4 | **=A3.run(EID=A2.select(EID==A3.EID))** | Create a reference between the main table and a subtable |
| 5 | **>A3.(EID=EID.sort(-AGE))** | Sort a record sequence field again |
| 6 | **=A3.new(NAME,EID(1).GENDER:GenderOfFirstChild, age(BIRTHDAY)-EID(1).AGE:ReproductiveAge)** | Directly aggregate a record sequence field |

## 2.5.12 Form a wide table[419][420]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from STATES")** | |
| 2 | **=demo.query("select * from EMPLOYEE")** | |
| 3 | **=A1.fname()\"NAME"** | Disable A1's key |
| 4 | **=A2.join(STATE,A1:NAME,${A3.concat@c()})** | Join two tables by keys and add a new field |

[417] url:jzbcyzcplxzd
[418] key:query,select,pos,id,run,new
[419] url:formwidetable
[420] key:wide table

# Chapter3 SQL computations[421][422]

*SQL computations* lists SQL code examples about topics in the following chapters, including General Computations, Data Search, Group and Join Operations, and Data Maintenance and Structure.

## 3.1 General Computations[423][424]

This chapter lists code examples about functions of general computations, including IS NULL/NVL/COALESCE, CAST/CONVERT, CASE/DECODE, AND/OR/NOT/<>, LIKE, COUNT/SUM/AVG/MAX/MIN, and IN/EXISTS.

### 3.1.1 IS NULL/NVL/COALESCE[425][426]

| | A | |
|---|---|---|
| 1 | **=demo.query("select EID,NAME,STATE,GENDER,BIRTHDAY, HIREDATE,DEPT, SALARY from EMPLOYEE")** | |
| 2 | **=A1.select(DEPT!=null)** | Not null |
| 3 | **=A1.select(DEPT ==null)** | Null |
| 4 | **=demo.query("select NAME,UNITPRICE,QUANTITY from RECEIPT")** | |
| 5 | **=A4.(NAME).ifn()** | The first non-null member |

### 3.1.2 CAST/CONVERT[427][428]

| | A | |
|---|---|---|
| 1 | **=date("1983-09-12")** | Convert string to date |
| 2 | **=string(A1,"yyyyMMdd")** | Convert date to string |
| 3 | **=int("5")** | Convert string to integer |
| 4 | **=string(5)** | Convert integer to string |
| 5 | **=decimal(A3)** | Convert integer to big decimal |
| 6 | **=ifnumber(A5)** | Judge if **A5** is a number |
| 7 | **=float("234")** | Convert string to float |

421 url:sqlp
422 key:SQL
423 url:jshs
424 key:general computation
425 url:isnullnvlcoalesce
426 key:query,select,ifn
427 url:castconvert
428 key:date,string,int,string,decimal,ifnumber,float

## 3.1.3 CASE/DECODE[429][430]

| | A |
|---|---|
| 1 | 1 |
| 2 | =case(A1,1,"ClassOne",2,"ClassTwo","ClassThree") |
| 3 | =if(A1==1, "ClassOne", A1==2,"ClassTwo","ClassThree") |

## 3.1.4 AND/OR/NOT, <>[431][432]

| | A | |
|---|---|---|
| 1 | =demo.query("select * from SALES") | |
| 2 | =A1.select(CLIENT=="PWQ" \|\| CLIENT=="QUICK") | OR |
| 3 | =A1.select(AMOUNT>5000 && AMOUNT<10000) | AND |
| 4 | =A1.select(!(CLIENT=="TRADH")) | NOT |
| 5 | =A1.select(CLIENT!="TRADH") | <> |

## 3.1.5 LIKE[433][434]

| | A |
|---|---|
| 1 | =demo.query("select * from SALES") |
| 2 | =A1.select(like(CLIENT,"*AY*")) |

[429] url:casedecode
[430] key:case,if
[431] url:andornot
[432] key:and,or,not,select
[433] url:like
[434] key:query,select,like

## 3.1.6 COUNT/SUM/AVG/MAX/MIN[435][436]

| | A |
|---|---|
| 1 | **=demo.query("select * from SALES")** |
| 2 | **=A1.sum(AMOUNT)** |
| 3 | **=A1.count()** |
| 4 | **=A1.avg(AMOUNT)** |
| 5 | **=A1.max(AMOUNT)** |
| 6 | **=A1.min(AMOUNT)** |

## 3.1.7 IN/EXISTS[437][438]

| | A | |
|---|---|---|
| 1 | **[1,3,5,7,9]** | |
| 2 | **=demo.query("select * from EMPLOYEE")** | |
| 3 | **=A2.select(A1.pos(EID)>0)** | IN |
| 4 | **[English,Math]** | |
| 5 | **=demo.query("select * from SCORES where CLASS='Class one'")** | |
| 6 | **=A5.select(SCORE>75).group(STUDENTID)** | |
| 7 | **=A6.select(~.(SUBJECT).pos(A4)!=null)** | EXISTS |
| 8 | **=A7.(STUDENTID)** | |

## 3.2 Data Search[439][440]

This chapter lists code examples about general data search, including SELECT * FROM …, WHERE …, SELECT … FROM, AS, SELECT…, ORDER BY/ASC/DESC, DISTINCT, FISRT/LAST/TOP/BOTTOM, UNION/UNION ALL/INTERSECT/MINUS, SELECT … FROM (SELECT … ), SELECT (SELECT … FROM) FROM, and CURSOR/FETCH.

435 url:countsumavgmaxmin
436 key:sum,count,avg,max,min
437 url:inexists
438 key:query,select,pos
439 url:ybsjjs
440 key:data search

### 3.2.1 SELECT * FROM …[441][442]

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

### 3.2.2 WHERE …[443][444]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=A1.select(SALARY>5000)** |

### 3.2.3 SELECT … FROM[445][446]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=A1.new(EID,NAME)** |

### 3.2.4 AS[447][448]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=A1.new(EID:EmployeeNo,NAME+" "+SURNAME:EmployeeName)** |

441 url:selectxfrom
442 key:select,query
443 url:where
444 key:where,query,select
445 url:selectfrom
446 key:select,query,new
447 url:as
448 key:as,query,new

## 3.2.5 SELECT …[449][450]

| | A | |
|---|---|---|
| 1 | **=[[1,"Beverage"]].new(~(1):TypeNumber,~(2):Name)** | Create a table sequence containing only one record |
| 2 | **=create(TypeNumber,TypeName).record( [1,"Beverage",2,"Grain"])** | Create an empty table sequence, and fill data in it |

## 3.2.6 ORDER BY/ASC/DESC[451][452]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=A1.sort(BIRTHDAY,-SALARY)** |

## 3.2.7 DISTINCT[453][454]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from SALES")** | |
| 2 | **=A1.id(CLIENT)** | Get the distinct value |
| 3 | **=A1.(CLIENT)** | All values available |
| 4 | **=A1.([CLIENT,SELLERID])** | All available values in the fields |

## 3.2.8 FISRT/LAST/TOP/BOTTOM[455][456]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.m(1).NAME** | FIRST |
| 3 | **=A1.m(-1).NAME** | LAST |
| 4 | **=A1.m(to(3))** | TOP 3 |
| 5 | **=A1.to(-3)** | BOTTOM 3 |

449 url:csb
450 key:new,create,record
451 url:orderbyascdesc
452 key:query,sort
453 url:distinct
454 key:distinct,query,id
455 url:firstlasttopbottom
456 key:query,m,to,first,last,top,bottom

## 3.2.9 UNION/UNION ALL/INTERSECT/MINUS[457][458]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.select(DEPT=="Sales" \|\| DEPT=="R&D")** | |
| 3 | **=A1.select(SALARY>5000)** | |
| 4 | **=A2\|A3** | UNION ALL |
| 5 | **=A2&A3** | UNION |
| 6 | **=A2^A3** | INTERSECTION |
| 7 | **=A2\A3** | DIFFERENCE |

## 3.2.10 SELECT … FROM (SELECT … )[459][460]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.select(DEPT=="Sales")** | Query |
| 3 | **=A2.count()** | Aggregate the result set |

## 3.2.11 SELECT (SELECT … FROM) FROM[461][462]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=demo.query("select * from FAMILY")** | |
| 3 | **=A1.derive()** | |
| 4 | **=A1.run(EID=A2.select(EID:A1.EID))** | Calculate the subtable reference first |
| 5 | **=A1.new(NAME,EID.count():NumberOfMembers)** | |
| 6 | **=A3.new(NAME,A2.select(EID:A3.EID).count(): NumberOfMembers)** | Use the direct-write method |

457 url:unionunionallintersectminus
458 key:union,query,select
459 url:selectfromselect
460 key:select,count
461 url:selectselectfromfrom
462 key:query,derive,run,new,count

## 3.2.12 CURSOR/FETCH[463][464]

| | A | B | C |
|---|---|---|---|
| 1 | **=demo.cursor("select * from SALES")** | | |
| 2 | **for** | | |
| 3 | | **=A1.fetch(100)** | Fetch **100** records each time |
| 4 | | **if B3==null** | **break** |
| 5 | | **…** | |

# 3.3 Group and Join Operations[465][466]

This chapter lists code examples about group operation and joins, including GROUP BY, HAVING, Perform equi-join on same-level tables, Perform equi-join on main table and subtable, Non-equi-joins, and LEFT JOIN/FULL JOIN.

## 3.3.1 GROUP BY[467][468]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.groups(DEPT; sum(SALARY):SalarySum)** | Group and aggregate together |
| 3 | **=A1.group(DEPT)** | First group |
| 4 | **=A3.new(DEPT,~.count():EmployeeNumber)** | Then aggregate |

[463] url:cursorfetch
[464] key:cursor,fetch,for,break
[465] url:fzygljs
[466] key:group operation, associative operation
[467] url:groupby
[468] key:query,groups,group,new

## 3.3.2 HAVING[469][470]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.groups(DEPT; sum(SALARY):SalarySum)** | Group and aggregate |
| 3 | **=A2.select(SalarySum >200000)** | Filter the aggregate result |
| 4 | **=A1.group(DEPT)** | Group |
| 5 | **=A4.select(~.count()>30)** | Filter grouped subsets |

## 3.3.3 Perform equi-join on same-level tables[471][472]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATENAME")** |
| 2 | **=demo.query("select * from STATEINFO")** |
| 3 | **=join(A1:StateName,STATEID;A2:StateInfo,STATEID)** |

## 3.3.4 Perform equi-join on main table and subtable[473][474]

| | A |
|---|---|
| 1 | **=demo.query("select * from STATES")** |
| 2 | **=demo.query("select * from EMPLOYEE")** |
| 3 | **=join(A1:State,NAME;A2:Employee,STATE)** |

## 3.3.5 Non-equi-joins[475][476]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=demo.query("select * from LIQUORS")** |
| 3 | **=demo.query("select * from RECEIPT")** |
| 4 | **=xjoin(A1:Employee,STATE=="New York";A2:Liquor,STOCK>500;A3:Food,QUANTITY>2)** |

469 url:having
470 key:query,groups,select,group
471 url:tjbdzlj
472 key:query,join
473 url:zzbdzlj
474 key:query,join
475 url:fdzlj
476 key:query,xjoin

## 3.3.6 LEFT JOIN/FULL JOIN[477][478]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=demo.query("select * from STATES")** | |
| 3 | **=demo.query("select * from ATTENDANCE")** | |
| 4 | **=demo.query("select * from PERFORMANCE")** | |
| 5 | **=join@1(A2:State,NAME;A1:Employee,STATE)** | Left join |
| 6 | **=join@f(A2:State,NAME;A1:Employee,STATE)** | Full join |
| 7 | **=join@1(A1:Employee,EID;A3:Attendance, EMPLOYEEID;A4:Performance, EMPLOYEEID)** | Join with the first table in alignment |

## 3.4Data Maintenance and Structure[479][480]

This chapter lists code examples about data maintenance and structure, including INSERT, INSERT FROM SELECT…, DELETE…WHERE…, UPDATE…WHERE…, CREATE/DROP TABLE, ALTER TABLE, KEYS, and CONNECT/DISCONNECT/COMMIT/ROLLBACK.

## 3.4.1 INSERT[481][482]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from SCORES")** | |
| 2 | **=A1.insert(0,"Class one",20,"PE",100)** | Append a new record |
| 3 | **=A1.insert(5,"Class one",21,"PE",100)** | Insert a new record |

## 3.4.2 INSERT FROM SELECT …[483][484]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from SALES")** | |
| 2 | **=A1.derive()** | Duplicate the table sequence |

477 url:leftjoinfulljoin
478 key:query,join
479 url:sjwhyjg
480 key:data maintenance, structure
481 url:inert
482 key:query,insert
483 url:insertfromselect
484 key:query,derive

## 3.4.3 DELETE … WHERE …[485][486]

| | A |
|---|---|
| 1 | **=demo.query("select ORDERID,CLIENT,SELLERID, ORDERDATE,AMOUNT from SALES")** |
| 3 | **=A1.delete(A1.select(AMOUNT<10000))** |

Delete the matching records

## 3.4.4 UPDATE … WHERE …[487][488]

| | A |
|---|---|
| 1 | **=demo.query("select ORDERID,CLIENT,SELLERID, ORDERDATE, AMOUNT from SALES")** |
| 2 | **=A1.select(CLIENT:"HL").run(AMOUNT= int(AMOUNT*1.1))** |

Update data on certain conditions

## 3.4.5 CREATE/DROP TABLE[489][490]

| | A | |
|---|---|---|
| 1 | **=create(ProductNo,ProductName,UnitPrice,Quantity)** | Create a table sequence |
| 2 | **>A1=null** | Delete the table sequence |

## 3.4.6 ALTER TABLE[491][492]

| | A |
|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** |
| 2 | **=A1.rename(EID:ID)** |

Modify field names

485 url:deletewhere
486 key:query,delete,select
487 url:updatewhere
488 key:query,select,run
489 url:createdroptable
490 key:create
491 url:altertable
492 key:query,rename

## 3.4.7 KEYS[493][494]

| | A | |
|---|---|---|
| 1 | **=demo.query("select * from EMPLOYEE")** | |
| 2 | **=A1.keys(EID)** | Set a primary key |

## 3.4.8 CONNECT/DISCONNECT/COMMIT/ROLLBACK

**495496**

| | A | B | |
|---|---|---|---|
| 1 | **=connect@e("demo")** | | Establish a connection |
| 2 | **>A1.execute@k(...)** | | |
| 3 | **=A1.error()** | | Error message arising from the execution of the previous database operation |
| 4 | **if A3==0** | **>A1.commit()** | Commit if no errors |
| 5 | **else** | **>A1.rollback()** | Rollback if any errors occur |
| 6 | **>A1.close()** | | Close the connection |

493 url:keys
494 key:query,keys
495 url:connectdisconnectcommitrollback
496 key:connect,execute,error,commit,rollback,close