esProc

Innovated data computing engine

Performance optimization - Traversal

Issued by Raqsoft

Contents

Storage format

Traversal performance differences under different data storage schemes

For the same 60 million data, with different storage formats:

Type	Time (second)
Oracle	328
Ctx (Binary)	26
txt	50

Storage Format	Characteristic	Performance ranking
Binary	Minimum space and fastest resolution	1
Text	Text has the advantage of being generic, but it does not perform well.	2
Database	It is also binary, but generally IO performance is poor, traversal in the database is fast, and data fetching is slow.	3

	A	B	C	D	E
1	>n=4		=now()		
2	>m=60000000				
3	=int(m/n)				
4	fork to(n)				
5		=connect("oracle")	=(A4-1)*A3	=A4*A3	
6		=B5.cursor("select * from orders where rownum>"+string(C5)+" and rownum<="+string(D5))			
7		for B6,10000			
8		=B5.close()			
9	=interval@s(C1,now())				

SPL Based on Database Storage

	A	B
1	=now()	
2	=file("/home/sjr/ctx/ORDERS_orderkey.ctx").create()	
3	fork to(4)	
4		=A2.cursor(;;A3:4)
5		for B4, 10000
6		=B4.close()
7	=interval@s(A1,now())	

SPL Based on Binary Storage

	A	B
1	=now()	
2	=file("/home/sjr/tpch_2_17_0/tbls/orders.tbl")	
3	>n=4	
4	fork to(n)	
5		=A2.cursor(;A4:4)
6		for B5, 10000
7		=B5.close()
8	=interval@s(A1,now())	

SPL Based on Text Storage

Segmentation - Text

Segmentation is for parallel, and need to fulfil four requirements:

1. The amount of data in each segment is basically the same.
2. The number of segments can be specified flexibly and dynamically.
3. Each segment is stored continuously and compactly.
4. Allow data addition

Text files can guarantee these four goals at the same time.
The files are divided equally according to the total byte size. In order to ensure the integrity of the records, remove the beginning of half record and add to the ending half record.

sid	amount
...	...
124	1024
...	...

Records are segmented by total byte segment points

Remove the beginning of half record and add to the ending half record to ensure the integrity of records

	sid	amount
1	sid	amount
2	10	5518
3	10	2081
4	10	5879
5	6	4333
6	2	3495
7	3	3175
8	1	1724
9	7	709
10	5	8841
11	3	1959
12	5	6669
13	1	6566
14	2	2304
15	9	4519
16	5	3638
17	4	4997
18	8	4753
19	5	5063
20	3	6031
21	7	73
22	10	2922

sales.txt

Member
124
126
125
125
125
125
124
126

Divide into eight sections
Remove the beginning and add to the end of each section
Number of records in each section.

Code example

	A
1	=file("sales.txt").cursor@t(;2:8).fetch()

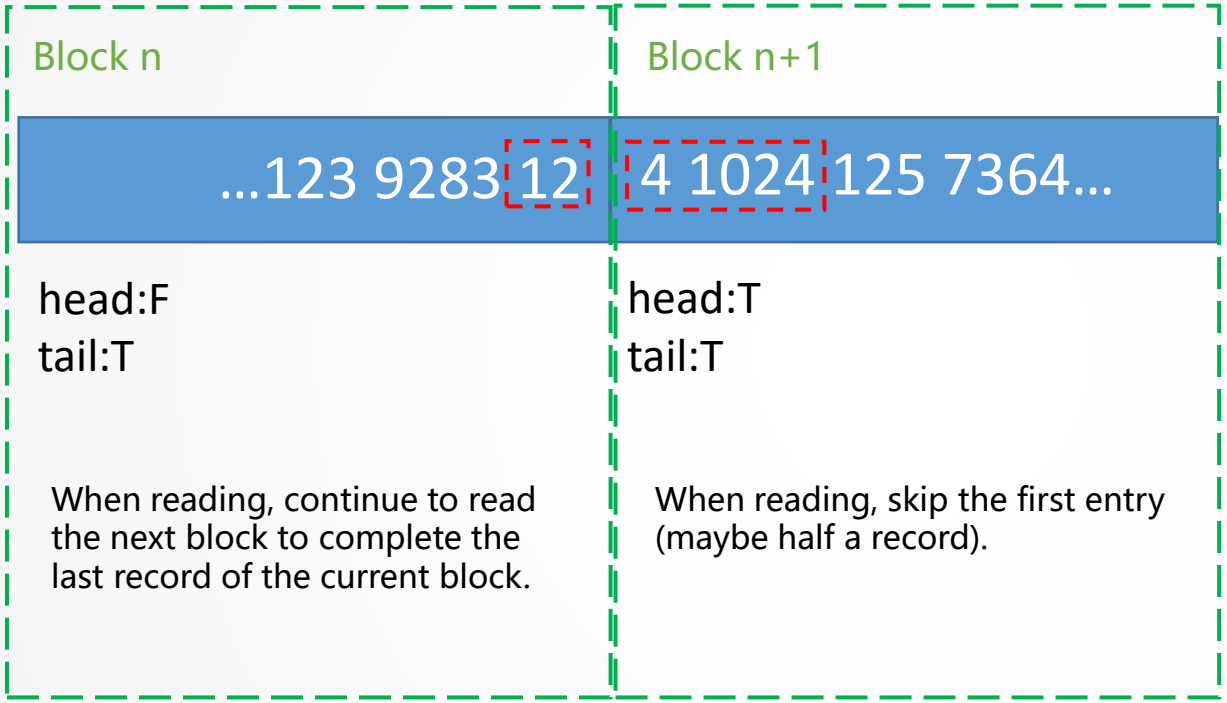
Read the second section in eight sections and compare the contents of the TXT file.

123	10	5176
124	1	8193
125	8	7797
126	2	9167
127	1	8170
128	10	7769
129	6	1100
130	2	8786
131	3	6389
132	4	318
133	8	1407

sid	amount
2	9167
1	8170
10	7769
6	1100
2	8786
3	6389
4	318
8	1407

Segmentation - Binary

The binary file cannot identify when the record ends,
so the block segmentation scheme is needed.



The head and tail tags indicate whether the first record in this block is a continuation of the previous block record and whether the last record is complete. T stands for incompleteness, F for completeness.

	sid	amount
2	10	5518
3	10	2081
4	10	5879
5	6	4333
6	2	3495
7	3	3175
8	1	1724
9	7	709
10	5	8841
11	3	1959
12	5	6669
13	1	6566
14	2	2304
15	9	4519
16	5	3638
17	4	4997
18	8	4753
19	5	5063
20	3	6031
21	7	73
22	10	2922

Examples of corresponding text structures

Segmentation——Double Increment Segmentation Technique

Set a fixed-length index area, suppose it can store 10 record addresses.

The index space is full for the first time

Position No. 1	Position No. 2	Position No. 3	Position No. 4	Position No. 5	Position No. 6	Position No. 7	Position No. 8	Position No. 9	Position No. 10
1	2	3	4	5	6	7	8	9	10

keep the address at position number 1, write address at position number 3 to position number 2 and address at position number 5 to position number 3, and address at position number 2i-1 to position number i

The index space is full for the second time

Position No. 1	Position No. 2	Position No. 3	Position No. 4	Position No. 5	Position No. 6	Position No. 7	Position No. 8	Position No. 9	Position No. 10
1、2	3、4	5、6	7、8	9、10	11、12	13、14	15、16	17、18	19、20

The index space is full for the third time

Position No. 1	Position No. 2	Position No. 3	Position No. 4	Position No. 5	Position No. 6	Position No. 7	Position No. 8	Position No. 9	Position No. 10
1、2、3、4	5、6、7、8	9、10、11、12	13、14、15、16	17、18、19、20	21、22、23、24	25、26、27、28、	29、30、31、32	33、34、35、36	37、38、39、40

As records continue to append, after the nth double increment, each position in the index space contains 2^(n-1) records.

Column storage

Because each column of data in the columnar storage is continuous, the traversal performance is better than row-wise storage in the case of fewer columns taken out.

	A
1	=file_btx.cursor@b(productid,quantity,area)
2	=now()
3	for A1,1000000
4	=interval@s(A2,now())
5	=file_ctx.create().cursor(productid,quantity,area)
6	=now()
7	for A5,1000000
8	=interval@s(A6,now())

For fifty million row storage file, traversing product IDs, quantities and areas takes 51 seconds

For fifty million column storage file, traversing product IDs, quantities and areas takes 22 seconds

productid	quantity	area
...	...	...
14028	52	1221
15938	64	1112
9364	96	1467
2646	24	1230
...	...	...

On the other hand, because the data is stored continuously by column, random access to hard disk will occur when multi-column computing is needed. When more columns are involved, it may not have advantages over row storage, and multi-threading will further aggravate this problem. Special attention should be paid to this problem when using column storage on mechanical hard disks, which may not always improve performance.

Column storage——Double Increment Segmentation

The double increment segmentation strategy is performed by records, so it’s suitable for segmenting data stored column-wise.

	Position No. 1	Position No. 2	Position No. 3	Position No. 4	Position No. 5	Position No. 6	Position No. 7	Position No. 8	Position No. 9	Position No. 10
Order ID	1、2、3、4	5、6、7、8	9、10、11、12	13、14、15、16	17、18、19、20	21、22、23、24	25、26、27、28、	29、30、31、32	33、34、35、36	37、38、39、40
User ID	1、2、3、4	5、6、7、8	9、10、11、12	13、14、15、16	17、18、19、20	21、22、23、24	25、26、27、28、	29、30、31、32	33、34、35、36	37、38、39、40
Product ID	1、2、3、4	5、6、7、8	9、10、11、12	13、14、15、16	17、18、19、20	21、22、23、24	25、26、27、28、	29、30、31、32	33、34、35、36	37、38、39、40
…	1、2、3、4	5、6、7、8	9、10、11、12	13、14、15、16	17、18、19、20	21、22、23、24	25、26、27、28、	29、30、31、32	33、34、35、36	37、38、39、40

By indexing newly-appended data in each column using the double increment segmentation mechanism, mismatching won’t happen and the corresponding segmentation points in all columns will always form a correct record.

Ordered compression

Column storage is easier for merge compression

USA	M	james
USA	M	jack
USA	F	alice
China	M	yao
China	F	lee

Sort by area first

USA	M	james
USA	M	jack
China	M	yao
USA	F	alice
China	F	lee

Sort by gender first

USA（3）, China（2）; M（2）, F（1）, M（1）, F（1）; Total characters 12

USA（2）, China（1）, USA（1）, China（1）; M（3）, F（2）; Total characters 18

Only count the number of characters, not the number of times in brackets

The number of characters in the area is longer than that of the gender, and putting the column with longer characters in the front will make the amount of storage smaller.

Code example for reading in and sorting

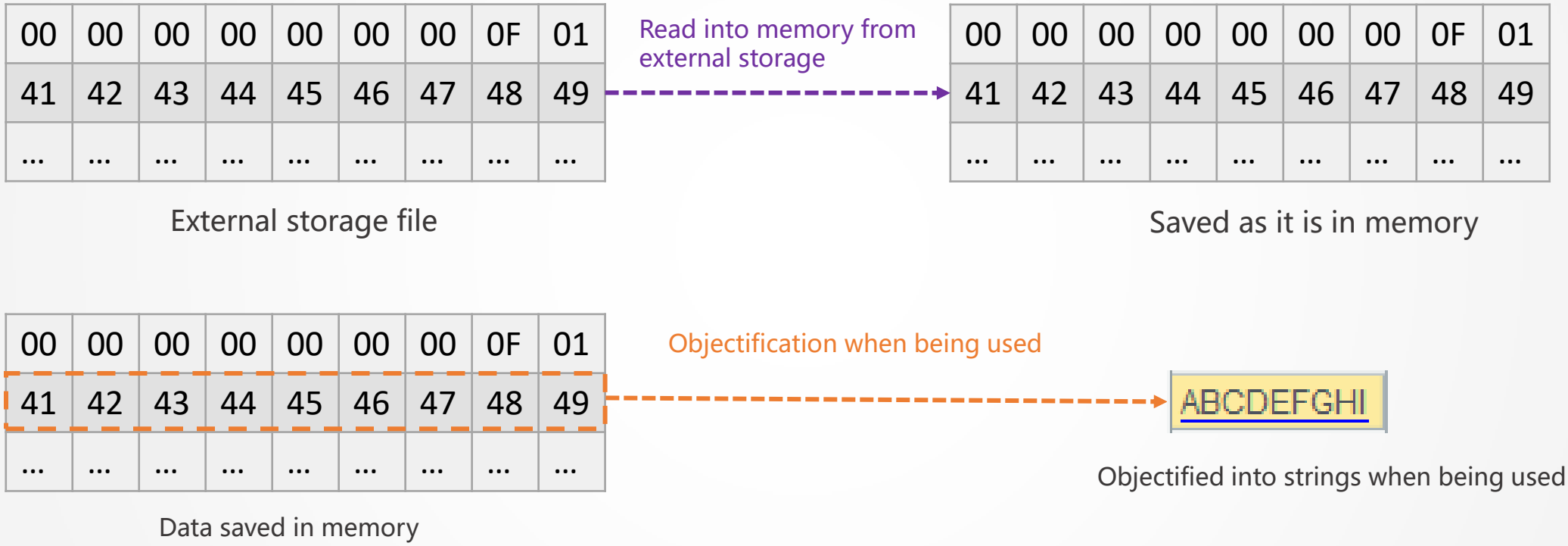
	A	B
1	=file("employee.ctx").create().cursor().sortx(level,height,weight,city)	
2	=file("employee_sort.ctx").create(level,height,weight,city,id,name,sex,birthday,salary,company)	
3	for A1,100000	=A3.new(level,height,weight,city,id,name,sex,birthday,salary,company).cursor()
4		=A2.append(B3)

Comparison of capacities for column storage file with compression after sorting.

employee.ctx	3:24 PM	385.9 MB
employee_sort.ctx	3:45 PM	335.5 MB

In-memory compression

JAVA is sensitive to memory usage and performance is severely affected by insufficient memory.



Data is not objectified when it is read in from external storage, but saved to memory as it is. It is objectified only when being used. This will lose performance, but will reduce memory usage.
The combination of column storage and ordered compression can also reduce memory usage, but will increase the complexity of record generation, so trade-offs are needed.

Minmax Index

When data is stored in blocks, the maximum and minimum values of each block can be recorded.

skipped

Order Date	...	min:2017-03-04 max:2017-03-04	min:2017-03-04 max:2017-03-05	min:2017-03-05 max:2017-03-05	min:2017-03-05 max:2017-03-05	...
Order amount	...	min:43275 max:47628	min:12038 max:52394	min:25670 max:28365	min:28456 max:31283	...

skipped

Using the minmax information on the data block, if there is no intersection between the condition interval and the minmax interval, the block data will be skipped directly.

For example, if the filter range of contract amount is between 29 000 and 30 000, the data blocks of green dotted line is skipped.

Order Date	Order amount	Order Number	...
2017-03-05	28456	7364875	...
2017-03-05	29137	7364876	...
...	...	...	...
2017-03-05	30294	7517645	...
...	...	...	...

Contents

Compute immediately

When using cursors, some calculations are performed immediately.

For example: aggregation functions like cs.groups.

Find the total price of all orders

	A
1	=order_cursor.groups(;sum(price):amount_q)

amount_q
3725114669

Find the price of orders according to the area of the orders.

	A
1	=order_cursor.groups(area;count(~):count_p)

area	count_p
10101	36698
10102	36734
10103	36869
10104	36877
10105	36798
10106	36327
10107	36811
10108	36522

Delayed calculation

Some calculations are not immediately computed after they are defined on the cursor.

For example: cs.select/cs.new/…

Find the sum of the order prices for which the order discount is less than 0.5.

	A
1	=order_cursor.select(discount<0.5)
2	=A1.groups(;sum(price):amount_q)

Value
com.raqsoft.dm.cursor.BFileCursor@2233a68f

amount_q
1843718959

The advantage of delayed computing is that it's as easy to understand since the writing is just like in-memory computing, but it doesn't really generate intermediate result sets (in-memory computing will) to take up space (or caching).

Pre-cursor filtering

Can reduce the generation of Java objects; using minmax index, columns that are not accessed are skipped directly for column storage.

Example: For orders with a discount rate greater than 0.9, find the sum of prices in their respective areas.

	A
1	=now()
2	=order_file.create().cursor(;discount>0.9)
3	=A2.groups(area;sum(price):amount)
4	=interval@s(A1,now())
5	=now()
6	=order_file.create().cursor()
7	=A6.select(discount>0.9).groups(area;sum(price):amount)
8	=interval@s(A5,now())

Pre-cursor filtering
It takes 26 seconds

Select filtering for cursors
It takes 56 seconds.

Filtering conditions

Find out employees whose names contain the word "Zhang" and whose salaries are less than 10,000.

	A
1	=employee_ctx.create().cursor(;salary < 10000 && like(name,“*Zhang*")).fetch()

Pay attention to the order of writing when multiple conditions &&. If the preceding subitem is false, the latter will not be calculated. By putting the condition that most of the result is false in the front, the calculation times of the latter condition item will be reduced.

The number of employees whose salary is less than 10,000 is less than the number of employees whose name contains the word "Zhang". By writing the condition item with a smaller result set in the front, the latter condition item will filter by a smaller former result set. This will also reduce the number of calculations.

Condition 1	Condition 2	&& condition
false	ture	false
false	false	false
ture	false	false
ture	ture	ture

Multi-cursor (Memory)

After data segmentation, multiple cursors can be used to traverse in parallel.

Example: Count Users by City.

	A
1	=user.cursor@m(;4)
2	=A1.groups(city;count(~):num)

city	num
Aiken	66505
Albany	66213
Albuquerque	65928
Anchorage	65849
Annapolis	66127
Asheville	65790
Atlanta	66182
Atlantic City	66165
Augusta	66071
Austin	66335
Baltimore	65756

The use of multiple cursors for grouping and aggregation is the same as ordinary cursor.

id	city
1	Atlanta
2	Charleston
3	Tuscaloosa
4	Lawrence
5	OklahomaCity
6	Honolulu
7	Cleveland
8	Pennsylvania
9	Virginia Beach
10	Tulsa
11	Greenville

Segment 1

id	city
2490369	Detroit
2490370	Jersey City
2490371	Columbus
2490372	Rochester
2490373	Las Vegas
2490374	Birmingham
2490375	Bismark
2490376	Albuquerque
2490377	Dover
2490378	Jersey City
2490379	Charlotte

Segment 2

4 parallel cursors
Partial content of each segment of data

id	city
4997121	Norman
4997122	Birmingham
4997123	Pennsylvania
4997124	Long Island
4997125	Saint Louis
4997126	Billings
4997127	Washington
4997128	Sacramento
4997129	Bloomington
4997130	Norfolk
4997131	Tampa

Segment 3

id	city
7503873	Lansing
7503874	Huntington
7503875	Huntsville
7503876	Reno
7503877	Denver
7503878	Springfield
7503879	Charlotte
7503880	Lafayette
7503881	Topeka
7503882	Tucson
7503883	Meridian

Segment 4

Multi-cursor (External Storage)

After data segmentation, multiple cursors can be used to traverse in parallel.

Example: Count male users with height greater than 170cm by cities(user_ctx is a group table file object).

	A
1	=user_ctx.create().cursor@m(;gender== “M” && height>170;4)
2	=A1.groups(city;count(~):num)

city	num
Aiken	66505
Albany	66213
Albuquerque	65928
Anchorage	65849
Annapolis	66127
Asheville	65790
Atlanta	66182
Atlantic City	66165
Augusta	66071
Austin	66335
Baltimore	65756

The use of multiple cursors for grouping and aggregation is the same as ordinary cursor.

id	city
1	Atlanta
2	Charleston
3	Tuscaloosa
4	Lawrence
5	OklahomaCity
6	Honolulu
7	Cleveland
8	Pennsylvania
9	Virginia Beach
10	Tulsa
11	Greenville

Segment 1

id	city
2490369	Detroit
2490370	Jersey City
2490371	Columbus
2490372	Rochester
2490373	Las Vegas
2490374	Birmingham
2490375	Bismark
2490376	Albuquerque
2490377	Dover
2490378	Jersey City
2490379	Charlotte

Segment 2

4 parallel cursors
Partial content of each segment of data

id	city
4997121	Norman
4997122	Birmingham
4997123	Pennsylvania
4997124	Long Island
4997125	Saint Louis
4997126	Billings
4997127	Washington
4997128	Sacramento
4997129	Bloomington
4997130	Norfolk
4997131	Tampa

Segment 3

id	city
7503873	Lansing
7503874	Huntington
7503875	Huntsville
7503876	Reno
7503877	Denver
7503878	Springfield
7503879	Charlotte
7503880	Lafayette
7503881	Topeka
7503882	Tucson
7503883	Meridian

Segment 4

Contents

Small grouping

When memory can accomodate the result set after grouping.

Example: an online mall has tens of millions of users and hundreds of millions of orders.

id	gender	type
3679666	F	6071
5538579	F	46
3479704	F	3351
4599465	F	6024
2141998	F	3061
148397	M	3480
709163	M	3222
8409353	M	785
5568863	M	9355
804994	F	7679
3599727	F	3335

Partial data of orders

Count the number of orders by order category (10,000 types).

	A
1	=order_cursor.groups(type;count(~):count)

type	count
1	10119
2	10108
3	10197
4	10065
5	10086
6	10118
7	10068
8	10118
9	10001
10	9927
11	10059

Ten thousand results
Memory can accomodate

Large sorting

When the data to be sorted cannot be accommodated by memory.

Example: Sorting user IDs in billions of order information

	A
1	=order_ctx.create().cursor().sortx(userid).fetch(10)

① Read part of the order record and write it as a temporary file

tmpdata2238386729201306812
tmpdata3908783783193554803
tmpdata5387736568463644291
tmpdata5284908551445175383
tmpdata5207250637950334006
tmpdata7241345240835400243
tmpdata425586817048643446
tmpdata5260285369551512440
tmpdata2235245067521657899

② View temporary file

	A
1	=file("tmp/tmpdata2235245067521657899").cursor@b()

③ Each temporary file has an ordered user ID

Index	orderid	userid
1	1195090	147
2	1177245	281
3	1411649	305
4	531773	537
5	1457604	859
6	821707	1571
7	687850	1717
8	2268545	1739
9	2276485	1992
10	295927	2091
11	1326086	2141
12	1184737	2378
13	2272474	2388
14	2098612	3433
15	781530	3597
16	1875256	3853
17	706940	4635
18	558733	4803
19	2200906	4823

④ Finally, the temporary files are aggregated

Large grouping

When the grouped result set is too large to fit in memory.

Large grouping: grouping orders by user ID (tens of millions of user ids)

id	gender	type
3679666	F	6071
5538579	F	46
3479704	F	3351
4599465	F	6024
2141998	F	3061
148397	M	3480
709163	M	3222
8409353	M	785
5568863	M	9355
804994	F	7679
3599727	F	3335

Partial data of orders

	A
1	=order_cursor.groupx(id;count(~):count)
2	=A1. fetch (1000)
3	>A1.close()

tmp
- tmpdata4458383290554497206
- tmpdata6838364572503028301
- tmpdata85319640366422427
- tmpdata4668858076644814483

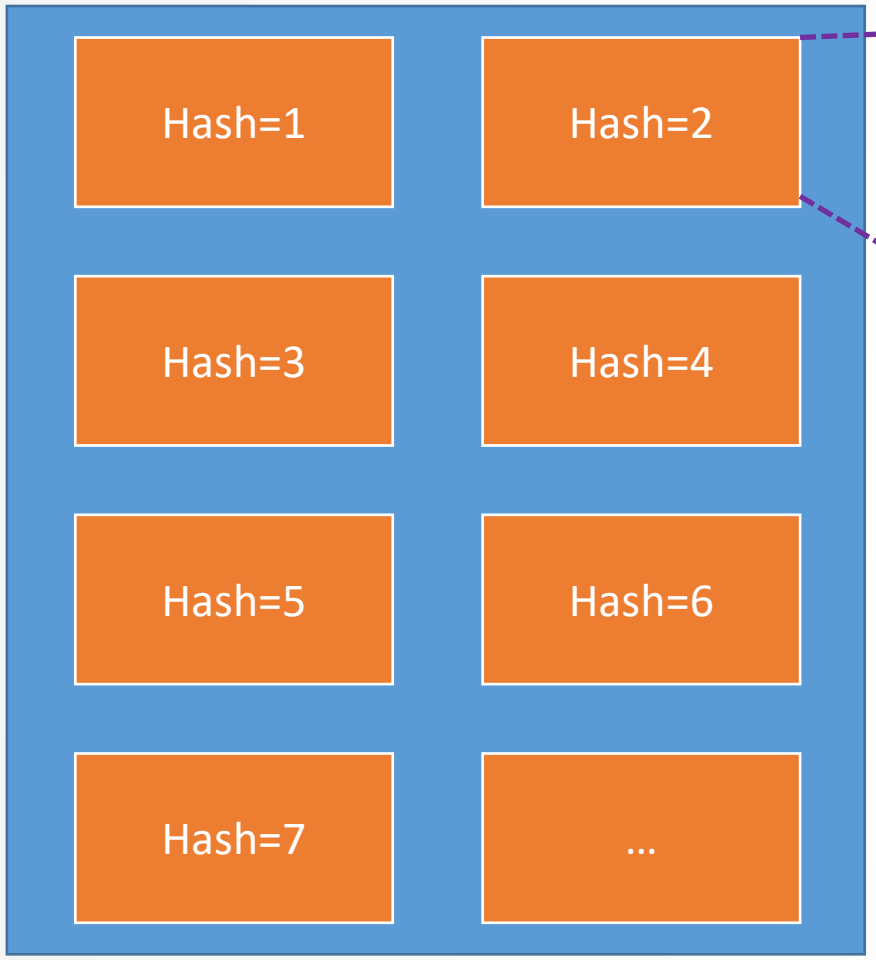
External storage cache file

id	count
1	12
2	8
3	7
4	8
5	15
6	13
7	4
8	8
9	14
10	7
11	16

Memory can't accommodate full result set. The first 1000 user IDs are fetched here.

After grouping by user id, the result set is too large to be loaded in memory at one time. The temporary external storage cache file is generated by using the method similar to the large sorting, and the grouping aggregation calculation is completed by using the external storage.

Hash grouping



userid	price	...
102	20	...
202	10	...
202	5	...
102	15	...
...	...	...

Hash keys of different userids are the same, and then traverse to find the same userid to aggregate

Get hash keys by hash calculation of grouping fields

	A
1	=order_ctx.create().cursor()
2	=A1.groupx@u(userid;sum(price):amount)
3	=A2.fetch()

Because hash function is not monotonous for the ranking field, the result of grouping is out of order.

Understanding aggregation

Other forms of aggregation operations of business significance - Understanding TopN as Aggregation

TopN of complete set

	A	B
1	=order_ctx_cursor.groups(;top(-3;due))	//Records of the three highest payable orders

TopN after grouping

	A	B
1	=order_ctx_cursor.groups(area;top(-2;due):top2)	//Records of the two highest payable orders per area

area	top2
10101	[[11967703,553329878,64528, ...],[35545768,5...
10102	[[13194936,458055115,207603, ...],[19183387,...
10103	[[41484229,565824001,107989, ...],[21870156,...
10104	[[48504327,19439204,202589, ...],[26110931,5...
10105	[[27906626,385692639,141642, ...],[37720790,...
10106	[[21837967,297887618,268908, ...],[7448959,5...

	customer	email	contractam...	due	invoiceam...	service
2	Wendy Bach	efk0n@col...	59974	59836	59414	2010-06
9	Charlene ...	1xr7eo@co...	59863	59790	59790	2009-05

Ordered grouping

Ordered grouping can be used when the grouping field itself is ordered.

Example: For order data (date ordered), count the number of orders per month in 2018.

	A	
1	=order_ctx_cursor.group(month(orderdate):month;count(~):count).fetch()	/When the month of the order date changes, it is treated as a new group.

orderid	userid	orderdate	...
826378	283674	2018-01-01	...
...	...	...	...
19387343	63742	2018-02-01	...
...	...	...	...
83625134	109527	2018-03-01	...
...	..	...	...

January Data

February Data

March Data

month	count
1	10567435
2	13522376
3	13583181
4	11247788
5	13318732
6	12294849
7	12451471
8	14102229
9	10039713
10	10531743
11	13792462
12	13201660

Count distinct

When the data is ordered, use merging and grouping to quickly de-duplicate (count distinct) .

	user	logintime
1	1	2019-06-12 0...
	1	2019-06-12 0...
	1	2019-06-12 0...
	1	2019-06-12 0...
	1	2019-06-12 0...
	1	2019-06-12 0...
	1	2019-06-12 0...
	1	2019-06-12 0...
2	2	2019-06-12 0...
	2	2019-06-12 0...
	2	2019-06-12 0...
	2	2019-06-12 0...
	2	2019-06-12 0...
	2	2019-06-12 0...
n	...	

Example: View the number of users logged in from a system log.

	A
1	=LogTable_cursor.groups@o(user).len()

user
1
2
3
4
5
6

Compare only with adjacent numbers

Value
129374

A total of 129374 users have appeared in this log.

Partially ordered grouping

When a field is ordered, it needs to be grouped by the following field.

Example: Order files ordered by date need to be grouped by product field to count sales.

orderdate	productid	due
2018-01-01	16	36250
2018-01-01	39	39719
2018-01-01	51	25696
2018-01-01	65	27279
2018-01-01	24	27834
2018-01-01	15	34660
2018-01-01	42	32415
2018-01-01	87	38065
2018-01-01	22	21338
2018-01-01	34	21130

The original data is ordered by date

	A
1	=order_ctx_cursor.group@q(orderdate;productid; sum(due):amount).fetch()

In calculating big data, when a field is ordered and only the following field needs to be grouped, the @q option can be used for in-memory grouping.

Such a processing method can avoid temporary external storage file reading and writing for large grouping, and improve efficiency.

orderdate	productid	amount
2018-01-01	94	924067
2018-01-01	95	811462
2018-01-01	96	1066541
2018-01-01	97	928465
2018-01-01	98	659242
2018-01-01	99	745950
2018-01-01	100	732033
2018-01-02	1	896970
2018-01-02	2	790339
2018-01-02	3	847484
2018-01-02	4	618941

Count sales after grouping date and product fields

Partially ordered sorting

When a field is ordered, only the following field needs to be sorted.

Example: Order file ordered by date, needs to be sorted by product field.

	A
1	=order_ctx_cursor.group@qs(orderdate;productid).fetch(10000)

Compared with partially ordered grouping, only sorting is done here, no grouping and aggregation.

orderdate	productid	due
2018-01-01	16	36250
2018-01-01	39	39719
2018-01-01	51	25696
2018-01-01	65	27279
2018-01-01	24	27834
2018-01-01	15	34660
2018-01-01	42	32415
2018-01-01	87	38065
2018-01-01	22	21338
2018-01-01	34	21130

The original data is ordered by date

After partially ordered sorting

orderdate	productid	due
2018-01-03	100	28036
2018-01-03	100	24317
2018-01-03	100	33572
2018-01-03	100	29315
2018-01-03	100	36037
2018-01-03	100	26493
2018-01-04	1	33688
2018-01-04	1	24155
2018-01-04	1	20183
2018-01-04	1	37617

Order date is 2018-01-03,
The result sorted by product ID

Order date is 2018-01-04,
The result sorted by product ID

Grouping and sorting by order number

Serial number grouping and sorting can be used when some data fields can be seen as ordinal numbers.

orderdate	productid	due
2018-09-18	52	24273
2018-05-23	95	20293
2018-01-26	11	39598
2018-04-27	3	23821
2018-08-26	52	24253
2018-03-05	12	37760
2018-01-30	61	34050
2018-01-10	76	32106
2018-06-22	12	21878
2018-04-01	37	35438

Partial data of orders

Grouping example: Grouping the order date by month to count the number of orders.

	A
1	=order_ctx_cursor.groupx@n(month(orderdate):month; count(~):count)

After month(orderdate), it can be seen as serial number 1~12, which can be used to group and aggregate.

month	count
1	84384
2	76262
3	84738
4	81737
5	85150
6	81787

Partial result after grouping

Sorting example: Sorting order dates by month

	A
1	=order_ctx_cursor.groupx@ns(month(orderdate))

After month(orderdate), it can be seen as serial number 1~12, which can be used to sort.

orderdate	productid	due
2018-01-01	14	22203
2018-01-11	1	38822
2018-01-30	2	36406
2018-01-06	40	39912
2018-01-06	86	34709
2018-01-05	23	24207
2018-01-27	59	29561
2018-01-12	30	35678

Partial result after sorting

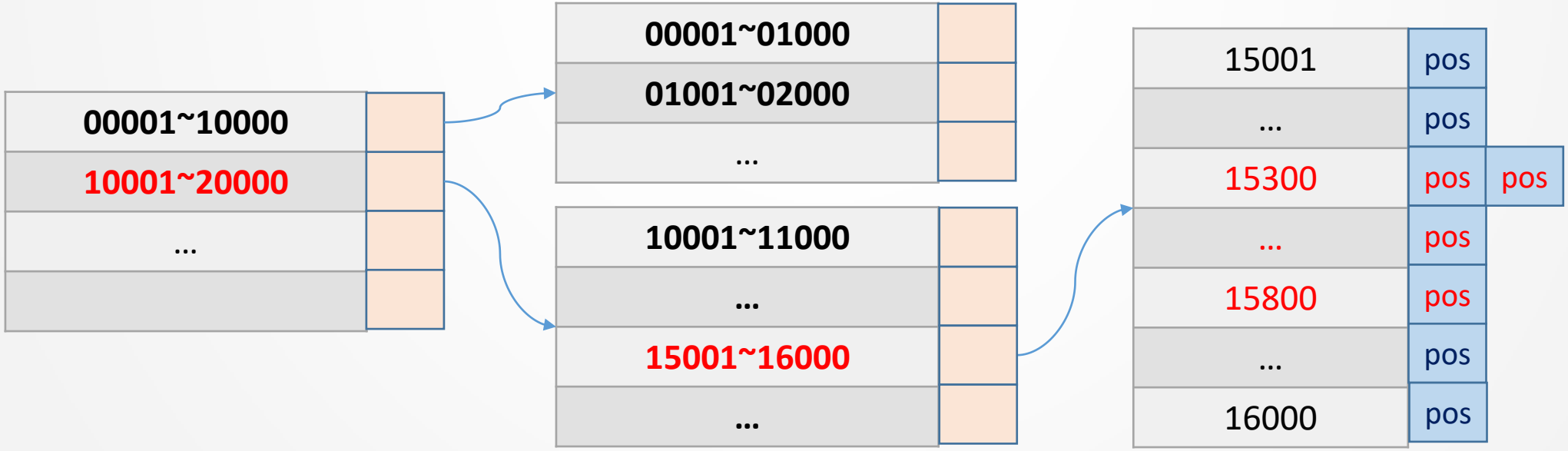
Sorting by Index

Indexes are sometimes used for sorting field that has been built with sorted indexes.

Example: Sort by consumption amount, and count the number of orders for certain consumption amount.

	A
1	=order_file.create()
2	=A1.icursor(;consume>15300 && consume<15800,consume_idx).groups@o(consume;count(~):count)

This example uses sorted index to return ordered cursor, and uses ordered grouping for fast statistics. But if it is disordered in physics, the performance may not be better when the quantity is large.



Segmental Sorting and Grouping

When sorting field can be grouped according to a rule and each group is small enough to fit in memory.

Example: For order file, sort by user id.

orderid	userid	orderdate
100000067473	17985431	2018-09-03
100000067474	16063511	2018-09-03
100000067475	16492110	2018-09-03
100000067476	11034981	2018-09-03
100000067477	18328142	2018-09-03
100000067478	17000786	2018-09-03
100000067479	11605948	2018-09-03
100000067480	19937216	2018-09-03
100000067481	15634998	2018-09-03
100000067482	18729157	2018-09-03
100000067483	16680981	2018-09-03
100000067484	12862752	2018-09-03

Partial content of order_file

	A
1	=order_file.cursor@b()
2	=A1.sortx@n(userid;userid\100000)
3	=user_file.export@b(A2)

After A2 is executed, it is grouped by userid\100000, and N temporary files are created.

```
[2019-06-25 17:28:37]
INFO: Create temporary file: E:\esProc\tmp\tmpdata1165466636928713499

[2019-06-25 17:28:37]
INFO: Create temporary file: E:\esProc\tmp\tmpdata4073295682889221679

[2019-06-25 17:28:37]
INFO: Create temporary file: E:\esProc\tmp\tmpdata1462223521012627634

[2019-06-25 17:28:37]
INFO: Create temporary file: E:\esProc\tmp\tmpdata8416877084849575604
```

Backstage log at A2 execution

Understanding segmental sorting:
If the userid is the same, the userid\100000 is the same, and the two are in the same order.
Compared with sortx, option @n can be used to omit the last aggregation.
Similarly, groupx@n can be used for large grouping.

orderid	userid	orderdate
100373833987	10000001	2018-12-11
101006951386	10000001	2018-12-14
100628626955	10000001	2018-12-18
100146933415	10000001	2018-12-31
100555368239	10000002	2018-01-02
100327959700	10000002	2018-01-02
100082799868	10000002	2018-01-03

user_file data after sorting

Redundant Grouping Dimensions

Omit "redundant items" in grouping dimension.

Example: According to order data and user data, the total order amount of each user is calculated by user id and user name.

	A
1	=users.keys(userid)
2	=order_ctx.create().cursor(userid,due).switch(userid,A1)
3	=A2.groups(userid.userid;userid.name,sum(due):amount)

The corresponding SQL is:

```
SELECT u.userid, u.name, SUM(o.due) AS aoumnt
FROM users u
        LEFT JOIN orders o ON u.userid = o.userid
GROUP BY u.userid, u.name
```

From the actual situation, it is obvious that the grouping dimension U.name in SQL is redundant, but according to the grammatical requirements of SQL, it can not be omitted even it's redundant.

userid	name
1	Brady Brewster
2	Fern Eliot
3	Juan Lowell
4	Jonathan Hoover
5	Judy Alick

A1 is user data

userid	name
79548	Andrew Crofts

userid	due
79548	17458
215588	45352
208430	24904
785633	36733
649860	26436

A2 is order data, Joins user data by userid

userid	name	amount
1	Brady Brewster	313130
2	Fern Eliot	478274
3	Juan Lowell	414755
4	Jonathan Hoover	237866
5	Judy Alick	362272

A3 groups by userid and calculates the sum of orders. The name follows userid here.

Contents

Reuse of traversal result

Calculate the sum of the amount grouped by city and the sum of the amount grouped by date.

Cursor

Cursor is pressed into channel

Channel

The same cursor can be pressed into several different channels. Multiple results can be computed by traversing cursor once.

city	date	amount
北京	2019-05-15	81
深圳	2019-05-14	18
上海	2019-05-16	2
广州	2019-05-15	37
北京	2019-05-13	49
上海	2019-05-15	55
广州	2019-05-14	32
上海	2019-05-16	50
北京	2019-05-14	55
广州	2019-05-16	60
上海	2019-05-15	55
深圳	2019-05-13	7
深圳	2019-05-16	59
深圳	2019-05-15	19
深圳	2019-05-13	20
深圳	2019-05-15	27
上海	2019-05-16	55
北京	2019-05-14	26

Data file

	A
1	=file.cursor@b()
2	=channel(A1).groups(city;sum(amount):camount)
3	=A1.groups(date;sum(amount):damount)
4	=A2.result()

date	damount
2019-05-13	1140
2019-05-14	1130
2019-05-15	1261
2019-05-16	1646

city	camount
上海	1292
北京	1458
广州	1187
深圳	1240

The sum of amounts grouped by date is defined in the cursor

The sum of amounts grouped by cities is defined in the channel.

Data Split

When each group of data after grouping needs to be separated independently, data splitting can be used to complete it.

zone	usertag	score
407	grnuup	407
298	daddrg	679
812	upbunb	672
730	dwdany	587
11	bpiedi	680
184	egiwag	455
685	pqpnnq	420
486	byaunr	689
890	npiipi	512
971	npncpd	551

Partial content of user_file

Example: Data with zone numbers greater than 10 and less than or equal to 10 are split into two separate files.

	A
1	=user_file.cursor@b()
2	=file("zone_gt10.btx")
3	=A1.select(zone<=10;A2)
4	=file(“zone_lt10.btx”).export@b(A3)

The action of exporting files here can also be changed to push to the channel.

zone_select_lt10.btx

zone_select_gt10.btx

Result files after conditional split

Example: Data after grouping for 1000 zones is stored independently in each file.

	A
1	=user_file.cursor@b()
2	=1000.(file("zone_"+string(~)+".btx"))
3	=A1.groupn(zone;A2)
4	=A1.skip()

Name

zone_1.btx
zone_2.btx
zone_3.btx
zone_4.btx
zone_5.btx
zone_6.btx
zone_7.btx
zone_8.btx
zone_9.btx

Part of the zone files exported.

Ordered Cursor

User behavior analysis: single-user operation is complex, but cross-user operation is almost nonexistent.

Example: Get the order information of customers who purchased more than 10 coffee and more than 10 milk in 2018 and save it as latte 2018.btx.

orderid	userid	category	quantity	orderdate
63044682	1	wine	3	2018-02-16
63057527	1	Coffee	6	2018-04-04
63059556	1	Tea	2	2018-04-12
63067460	1	Cola	10	2018-05-11
63075249	1	Cola	5	2018-06-08
63061783	2	Cola	5	2018-04-20
63062280	2	wine	4	2018-04-22
63062678	2	Milk	10	2018-04-23
63064165	2	wine	4	2018-04-29
63064634	2	Coffee	7	2018-04-30
63067961	2	Coffee	6	2018-05-12

Historical Data of an Online Supermarket in 2018 (Ordered by userid))
order2018.btx

	A	B	C
1	=file(“order2018.btx").cursor@b()		
2	for A1;userid	=A2.groups(category;sum(quantity):amount).select(category==" Coffee" || category=="Milk")	
3		=B2.(~.amount>10)	
4		if B3.len()==2 && B3(1) && B3(2)	=file("latte2018.btx").export@ab(A2)

orderid	userid	category	quantity	orderdate
63044682	1	wine	3	2018-02-16
63057527	1	Coffee	6	2018-04-04
63059556	1	Tea	2	2018-04-12
63067460	1	Cola	10	2018-05-11
63075249	1	Cola	5	2018-06-08

Read the same user's purchase information in A2 each time

Index	category	amount
1	Coffee	6

Current user only buys coffee, and the number is not more than 10
B4 is false

Ordered Cursor——Changing conditions

Log analysis: starting with a special string, to find such a string is the beginning of a group.

Example: a log, with - - - flag - - - as the start of the event, and the next row is user tag. Find the user with the largest number of rows of records in a single event.

Member
---flag---
usertag:rkhmfy
ajqssojpdwhugxmgmg
srtpthodnpemqnjzwcsplnhkkjscfprvsaokps
nbmghigkmewzukqmlmcvqczvjuehyaohlaabgkhotlyher...
bmxtxcovdtijzbigmnzrzvlykekwxpcpxpvosonucebzelpai
ywciadvsjexnxeoj
qmgzrhqvfmtwtzilpnznsmummttbzpd
---flag---
usertag:hommtp
bcgirppghtdvhuib
mulcbpngerzmlxdvlesegkyvxrwtyi
ldrwcxiveiydguupxwmekkaezuhfdxhfoloh
jeljpnjedsnemycvamjtnmatswxocpsmvwvmfqfyjkfumalf...

Partial data of log file

	A	B
1	=file("log.txt").cursor@i()	=[,]
2	for A1;~=="---flag---"	=[A2(2),A2.len()-2]
3		>B1=if(B2(2)>B1(2),B2,B1)

Member
---flag---
usertag:rkhmfy
ajqssojpdwhugxmgmg
srtpthodnpemqnjzwcsplnhkkjscfprvsaokps
nbmghigkmewzukqmlmcvqczvjuehyaohlaabgkhotlyher...
bmxtxcovdtijzbigmnzrzvlykekwxpcpxpvosonucebzelpai
ywciadvsjexnxeoj
qmgzrhqvfmtwtzilpnznsmummttbzpd

Records of the first group of event

Member
usertag:dufcuf
13

After the program is executed, B1 is the maximum number of rows and the corresponding usertag.

Iteration within the group

Iterative aggregation grammar can be computed while traversing, and the target set only needs to be traversed once.

Example: Calculate the cumulative sales of each user as of today for each month (data is ordered by userid and date).

userid	orderdate	due
10000001	2018-01-02	19
10000001	2018-01-23	10
10000001	2018-01-30	16
10000001	2018-02-10	16
10000001	2018-03-21	14
10000001	2018-03-26	17
10000001	2018-04-03	15
10000001	2018-05-21	12

Partial content of order_cursor

	A
1	=order_cursor.derive(iterate(~~+~.due,0;userid,month(orderdate)):total)
2	=A1.fetch@x(;userid)

The parameter meaning in expression iterate(~~+~.due,0;userid,month(orderdate):
"~~+~.due" is the cumulative value for due. Each time the userid and month (order date) change, the accumulated variable is cleared and accumulated again.

userid	orderdate	due	total
10000001	2018-01-02	19	19
10000001	2018-01-23	10	29
10000001	2018-01-30	16	45
10000001	2018-02-10	16	16
10000001	2018-03-21	14	14
10000001	2018-03-26	17	31
10000001	2018-04-03	15	15
10000001	2018-05-21	12	12

The total column in A2 is the monthly cumulative value of the first userid.

Program Cursor

Using program cursor can avoid reading and writing temporary external storage files and improve performance.

Example: According to the order information of customers who purchased more than 10 coffee and more than 10 milk in 2018, find the last two purchase records of that year for that customer.

Method 1: Use latte2018.btx saved in “Ordered cursor” section.

	A
1	=latte2018_cursor.groups(userid;top(-2;orderdate):top2)

userid	top2
2	[[63126426,2,wine, ...],[63117504,2,Milk, ...]]
6	[[63131220,6,Coffee, ...],[63127286,6,Milk, ...]]
9	[[63127325,9,Tea, ...],[63124019,9,Tea, ...]]

orderid	userid	category	quantity	orderdate
63126426	2	wine	9	2018-12-12
63117504	2	Milk	5	2018-11-09

Method2: Use program cursor to avoid the landing of temporary intermediate files.

	A	B	C	D
1	func	=file(“order2018.btx").cursor@b()		
2		for B1;userid	=B2.groups(category;sum(quantity):amount).select(category=="Coffee" || category=="Milk")	
3			=C2.(~.amount>10)	
4			if C3.len()==2 && C3(1) && C3(2)	return C2
5	=cursor@c(A1).groups(userid;top(-2;orderdate):top2)			

Computing content id equivalent to “ordered cursor “ section

Manual parallelism

Manual parallelism is more flexible than easy-to-use function options.

Example: Parallel statistics of all orders of users purchasing more than 3 milk in monthly files.

	A	B	C	D
1	=12.(file("month_"+string(#)+".txt"))			
2	fork to(12)	for A1(A2).cursor@t();userid	=B2.groups(category;sum(quantity):amount).select(category=="Milk")	
3			if C2.len()==1 && C2.amount>3	=file("milk_gt3_"+string(A2)+".txt").export@at(B2)

A2 is divided into 12 threads. Each thread uses an ordered cursor to calculate a user whose milk purchasing quantity is more than 3 in the same month, and export all the order information of that user in the same month to the corresponding month's file (milk_gt3_month.txt).

1	orderid	userid	category	quantity	orderdate
2	63036881	3	Cola	1	2018-01-19
3	63033101	4	Milk	5	2018-01-05
4	63037888	5	Tea 7	2018-01-22	
5	63033796	7	wine	4	2018-01-08
6	63033205	8	Coffee	9	2018-01-06
7	63038443	8	Milk	4	2018-01-24
8	63039666	8	Tea 6	2018-01-29	

month_1.txt

1	orderid	userid	category	quantity	orderdate
2	63033101	4	Milk	5	2018-01-05
3	63033205	8	Coffee	9	2018-01-06
4	63038443	8	Milk	4	2018-01-24
5	63039666	8	Tea 6	2018-01-29	
6	63035725	12	Milk	5	2018-01-15
7	63036028	20	Coffee	7	2018-01-16
8	63038625	20	Milk	5	2018-01-25

milk_gt3_1.txt

Innovation makes progress!