Query of Large Quantity of Random Key Values in Big Data

Search of massive key values

Search of single record：

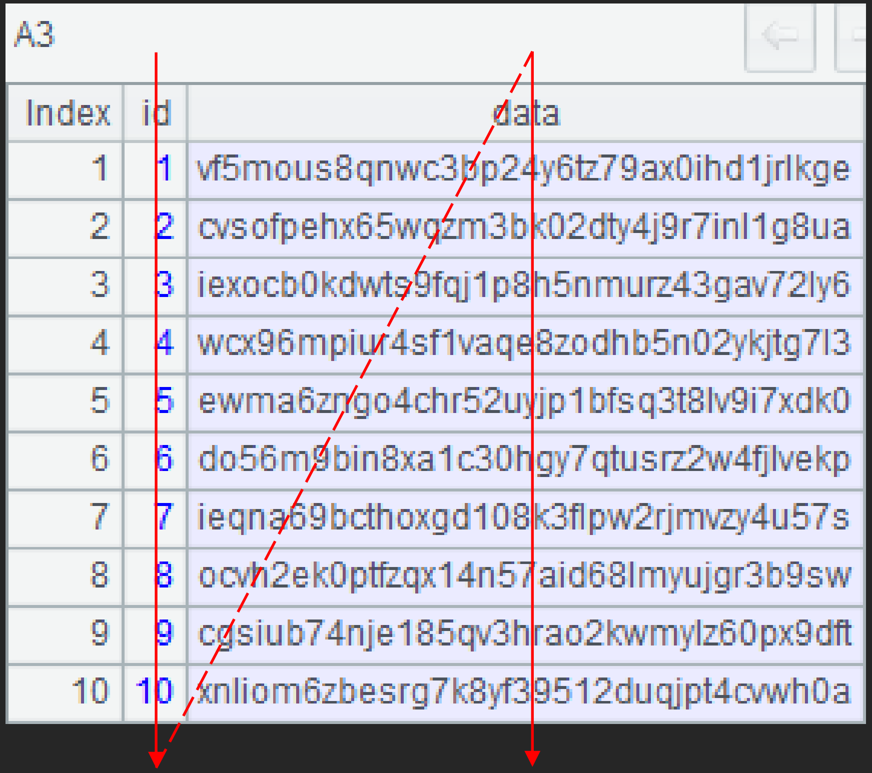
Create index according to key value for data table, and the search complexity is only logN times. For 1 billion rows of data, the comparison is only 30 times, which takes only a few milliseconds on modern computers.

Search of massive key values：

For as many as thousands or even tens of thousands of key values, if we simply use the database index, the time delay will rise to tens of minutes or even hours.

Row storage and column storage

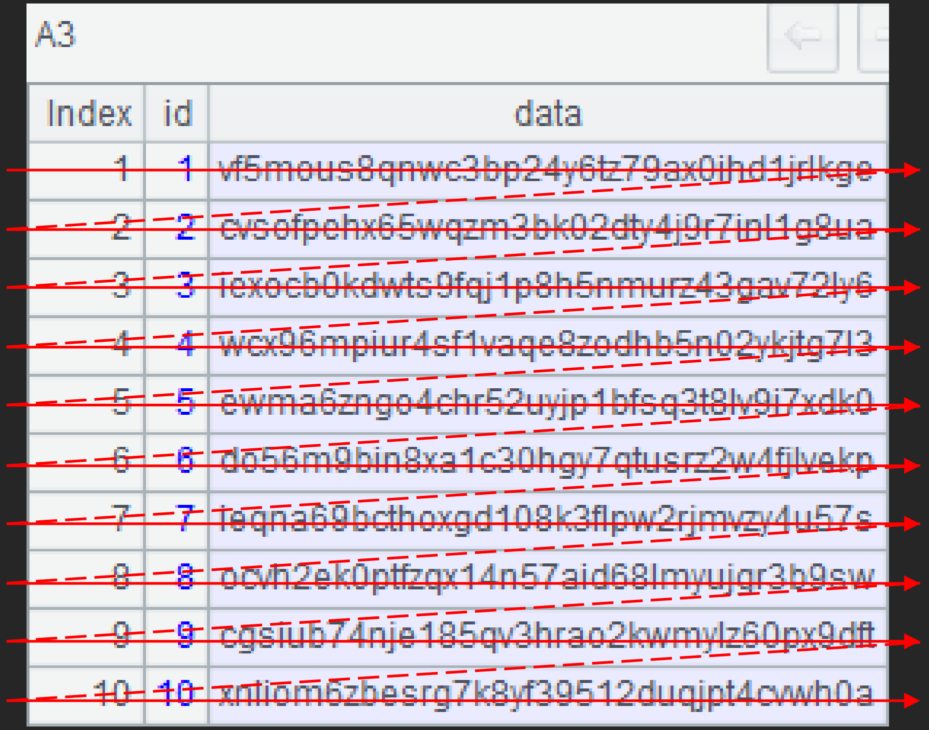
For scenarios where the entire record needs to be taken out, row storage is more appropriate than column storage.



A3

Index	id	data
1	1	vf5mous8qnwc3bp24y6tz79ax0ihd1jrlkge
2	2	cvsofpehx65wqzm3bk02dty4j9r7inl1g8ua
3	3	iexocb0kdwts9fqj1p8h5nmurz43gav72ly6
4	4	wcx96mpiur4sf1vaqe8zodhb5n02ykjtg7l3
5	5	ewma6zngo4chr52uyjp1bfsq3t8lv9i7xdk0
6	6	do56m9bin8xa1c30hgy7qtusrz2w4fjlvekp
7	7	ieqna69bcthoxgd108k3flpw2rjmvzy4u57s
8	8	ocvh2ek0ptfzqx14n57aid68lmyujgr3b9sw
9	9	cgsiub74nje185qv3hrao2kwmylz60px9dft
10	10	xnliom6zbesrg7k8yf39512duqjpt4cvwh0a

Column storage



A3

Index	id	data
1	1	vf5mous8qnwc3bp24y6tz79ax0ihd1jrlkge
2	2	cvsofpehx65wqzm3bk02dty4j9r7inl1g8ua
3	3	iexocb0kdwts9fqj1p8h5nmurz43gav72ly6
4	4	wcx96mpiur4sf1vaqe8zodhb5n02ykjtg7l3
5	5	ewma6zngo4chr52uyjp1bfsq3t8lv9i7xdk0
6	6	do56m9bin8xa1c30hgy7qtusrz2w4fjlvekp
7	7	ieqna69bcthoxgd108k3flpw2rjmvzy4u57s
8	8	ocvh2ek0ptfzqx14n57aid68lmyujgr3b9sw
9	9	cgsiub74nje185qv3hrao2kwmylz60px9dft
10	10	xnliom6zbesrg7k8yf39512duqjpt4cvwh0a

Row storage

Summary of contents

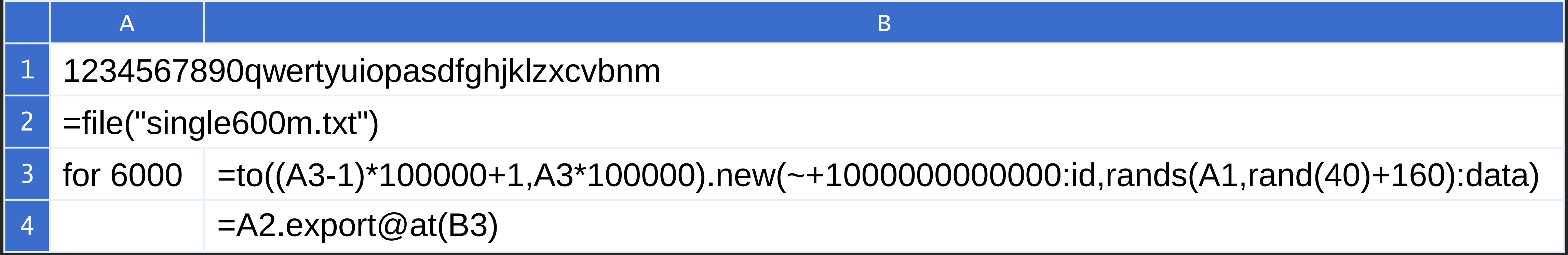
1) Single field key
2) Multi-field key
3) Multithread Query
4) Index redundancy mechanism
5) Data Addition Processing

Single field key - data generation

Let's take the following data structure as an example

Field name	Type	Is the primary key?	Explanation
id	int	Yes	Begin with 1000000000001 to increase
data	string		Data to be acquired

According to the above data structure, 600 million pieces of data can be created in the text file, which can be written as follows:



	A	B
1	1234567890qwertyuiopasdfghjklzxcvbnm	
2	=file("single600m.txt")	
3	for 6000	=to((A3-1)*100000+1,A3*100000).new(~+1000000000000:id,rands(A1,rand(40)+160):data)
4		=A2.export@at(B3)

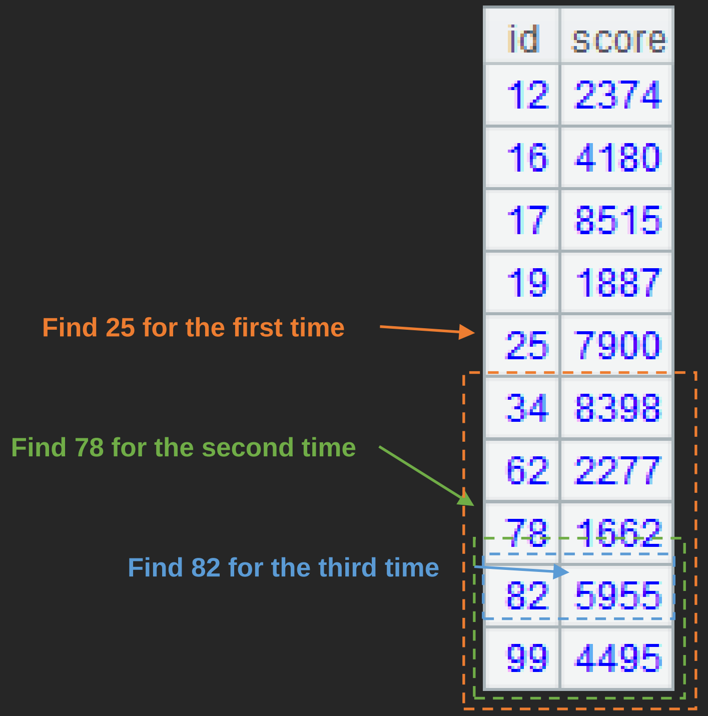
Single field key – Exercise problem

Search of massive random key values

From 600 million records, find records corresponding to 10,000 randomly distributed key set

Single Field Key-Index Principle

Example of dichotomy : Find User Information with ID 82



Find 25 for the first time

Find 78 for the second time

Find 82 for the third time

id	score
12	2374
16	4180
17	8515
19	1887
25	7900
34	8398
62	2277
78	1662
82	5955
99	4495

Ordered users table

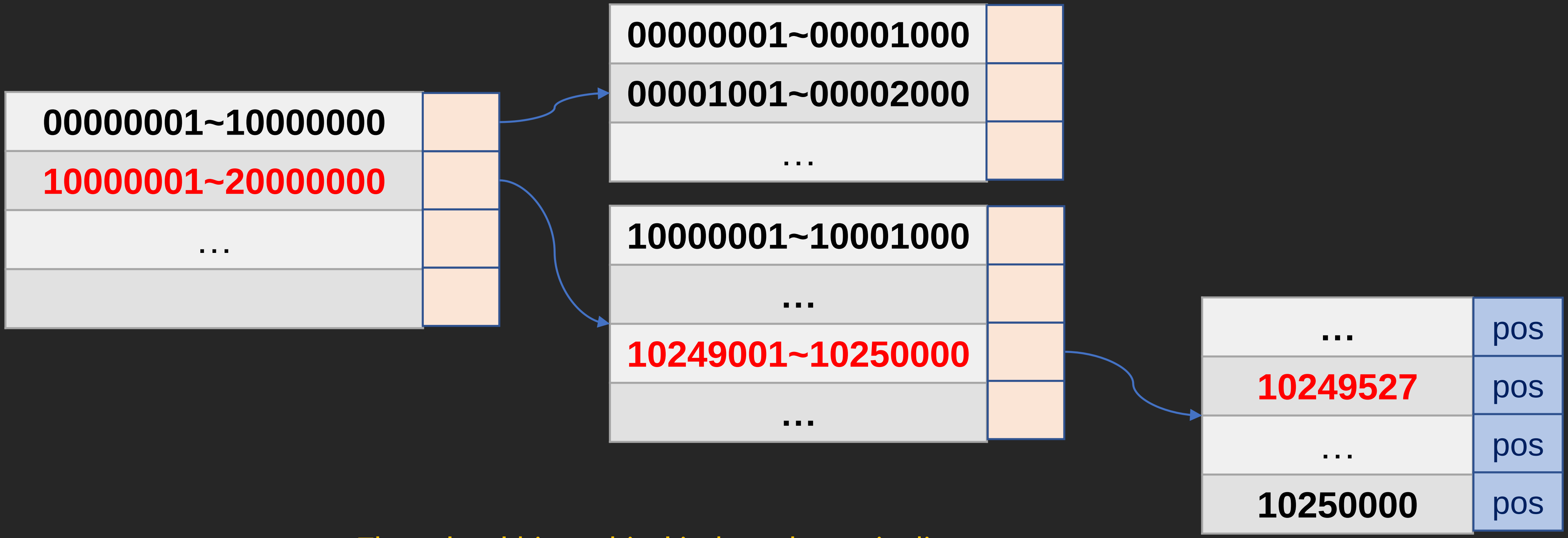
In this case, sequential lookup (traversal) requires nine comparisons, while dichotomy only uses three comparisons.

The time complexity of sequential lookup is O (n).

The time complexity of dichotomy lookup is O (log_2).

Single field key - hierarchical index

Large index can not fit in memory, and there is no need to fit in memory, hierarchical index can be used.



Three-level hierarchical index schematic diagram

Single field key - Key value type

For non-integer key values, they should first be converted to integers.

License number	Data column
京A12345	…
沪B56789	…
…	…

Convert to numbers

License number	Data column
100112345	
110256789	

ID number	Data column
110105197608028736	…
310104199809209731	…
…	…

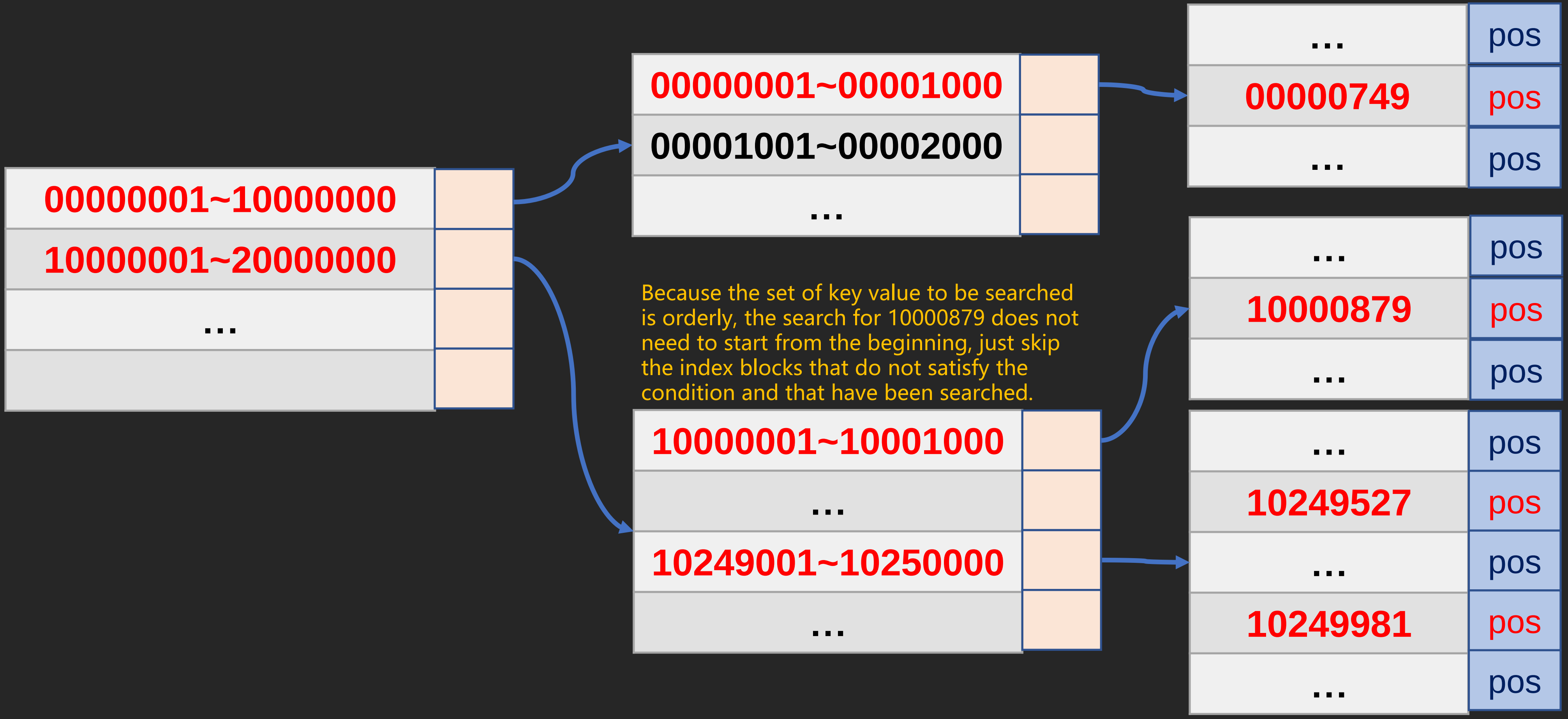
Multi-layer serial number

ID number	Data column
11 01 05 19 76 08 02 87 36	…
31 01 04 19 98 09 20 97 31	…
…	…

Single field key - key value sorting

Key value set to be searched is orderly, so we can avoid turning back when searching.

Example： According to the ordered set of key value, search（749、…、10000879、…、10249527、10249981、…)



Because the set of key value to be searched is orderly, the search for 10000879 does not need to start from the beginning, just skip the index blocks that do not satisfy the condition and that have been searched.

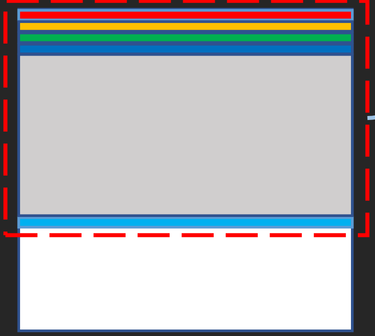
Because the set of key value to be searched is ordered, These two key values 10249527 and 10249981 fall into the same index block, and there is no need to go back and compare.

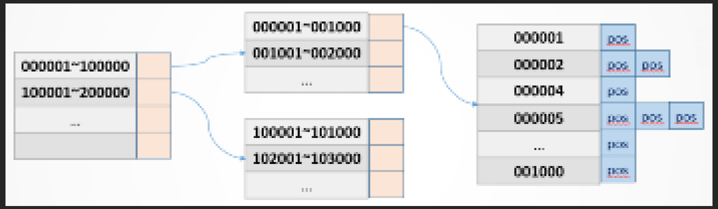
Single field key - index cache

Preload Index Cache to Improve Query Efficiency

Random key-value queries directly using indexes:
First query， time :80 seconds
Second query， time:78 seconds
Third query， time:77 seconds
Fourth query， time:76 seconds
……
Nth query, time:25 seconds

Cache Memory

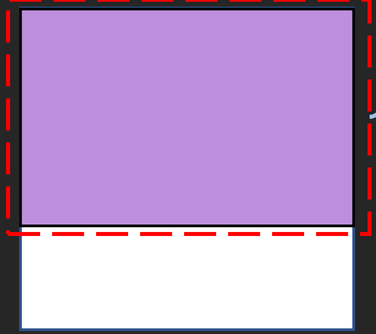




Sorting index

Preload index cache:
Each random key query takes about 25 seconds

Cache Memory



Each time an index is used for key value queries, the operating system generates a cache.

After N times of using index queries, the efficiency will reach the limit.

Index caches can be pre-loaded so that each query is in the most efficient state.

Single field key – Create group table and index

Generate a group table file that is stored row-wise using the text file that has been created.

	A
1	=file("single600m.txt").cursor@t()
2	=file("single600m.ctx").create@r(#id,data).append(A1)

Note: Key value id should be ordered in the table, because disk jitter can be relatively reduced when the set of key value is concentrated.

Create a sort index of ID keys for the group table file.

	A
1	=file("single600m.ctx").create().index(id_idx;id)

Single field key - group table query

Random generation of 10,000 ids, use index to query.

	A	B
1	=file("single600m.ctx").create()	/Open the group table
2	=A1.index@3(id_idx)	/Load three level index cache
3	=10000.(1000000000000+(rand(600000000)+1)).sort()	/Randomly choose 10,000 ordered key values
4	=now()	/Current time
5	=A1.icursor(;A3.contain(id),id_idx).fetch()	/Batch key value search using index
6	=interval@ms(A4,now())	/Time used for searching

There are two points to note when querying:

1. Key value sorting: The set of key value to be searched is ordered.

2. Index cache: Preload index cache before the search.

Single field key - Oracle imports data and creates index

Import the created text file data into Oracle.

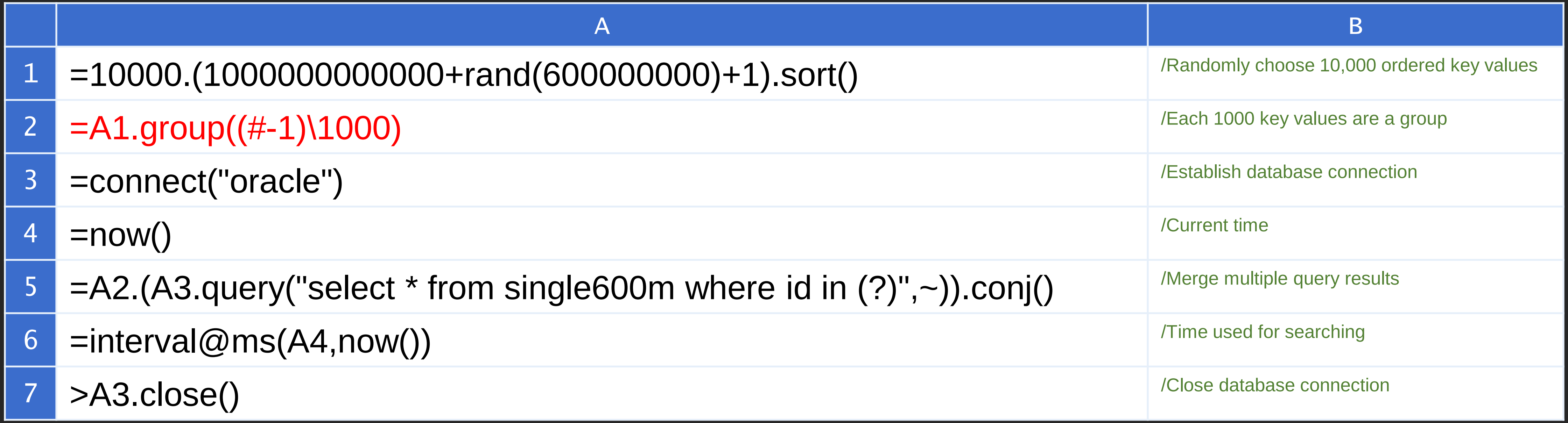
Oracle create table：create table single600m (id number(13),data varchar2(200));

Import text file content into table using Oracle's SqlLoader (omitted)

Oracle create index：create unique index idx_id_600m on single600m(id);

Single field key – Oracle query

Random generation of 10,000 ids, use index to query.



	A	B
1	=10000.(1000000000000+rand(600000000)+1).sort()	/Randomly choose 10,000 ordered key values
2	=A1.group((#-1)\1000)	/Each 1000 key values are a group
3	=connect("oracle")	/Establish database connection
4	=now()	/Current time
5	=A2.(A3.query("select * from single600m where id in (?)",~)).conj()	/Merge multiple query results
6	=interval@ms(A4,now())	/Time used for searching
7	>A3.close()	/Close database connection

Grouping into several groups of 1000 entries because the maximum number of in in the database supports 1000 entries.

Multi field key

Let's take the following data structure as an example

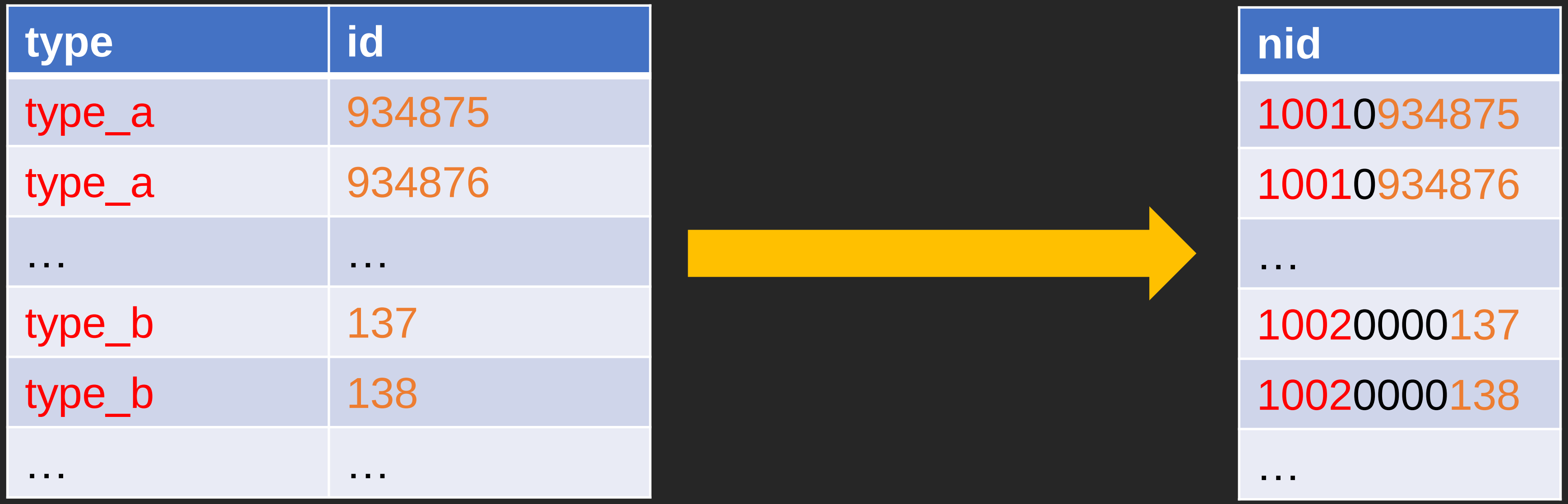
Field name	Type	Is the primary key?	Explanation
type	string		enumerable
id	int		The ID of each enumeration type increases from 1
data	string		Data to be acquired

Type and ID fields are used as joint primary keys to determine a record.

Multi field key -- Merge primary keys

It involves the storage and comparison of sets, and is slower than single field key.

In order to achieve high performance, a more common method is to combine multi-field key into single-field key.



type	id
type_a	934875
type_a	934876
…	…
type_b	137
type_b	138
…	…

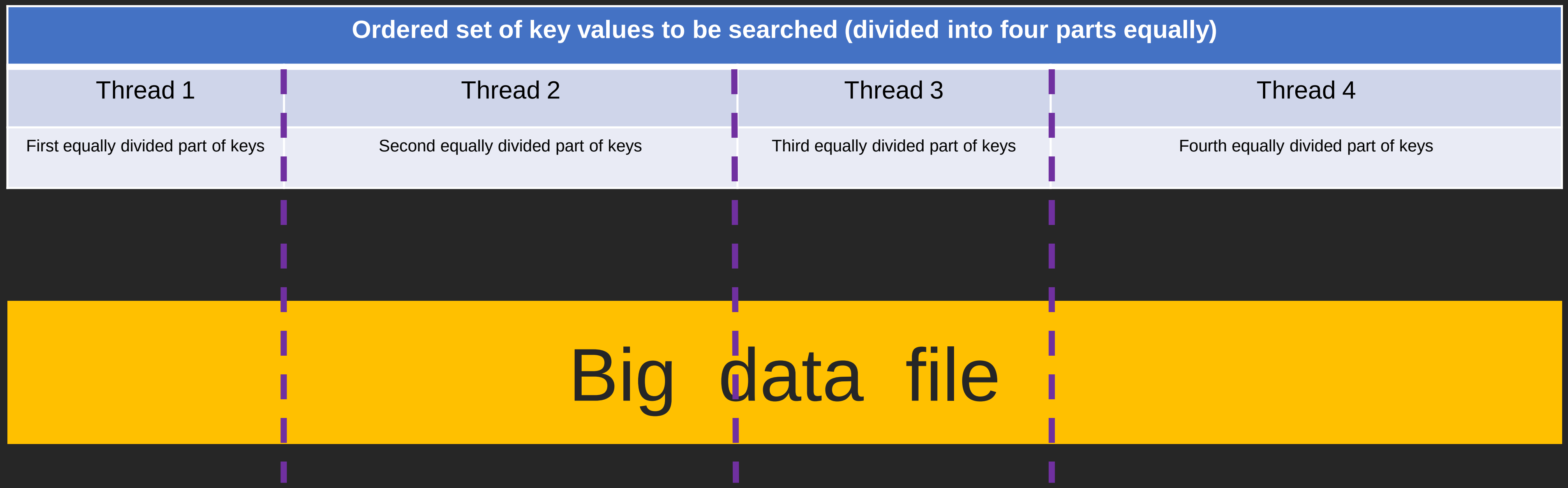
nid
10010934875
10010934876
…
10020000137
10020000138
…

For the NID after merging primary keys, it can be processed according to the method of single field key. Pay attention that the NID needs to be ordered.

Multithread Query

Multi-threaded Parallel Approach to Further Improve Performance

Example: data file, 4 threads, ordered keys are divided into 4 segments in sequence



Ordered set of key values to be searched (divided into four parts equally)			
Thread 1	Thread 2	Thread 3	Thread 4
First equally divided part of keys	Second equally divided part of keys	Third equally divided part of keys	Fourth equally divided part of keys

Multithread Query – Group table

Multi-threaded Parallel Approach to Further Improve Performance

	A	B
1	=file("single600m.ctx").create()	
2	=A1.index@3(id_idx)	
3	=10000.(1000000000000+(rand(600000000)+1)).sort()	
4	=A3.group((#-1)\1000)	
5	=now()	
6	fork A4	=A1.icursor(;A6.contain(id),id_idx)
7		=B6.fetch()
8	=A6.conj()	
9	=interval@ms(A5,now())	

Attention should be paid to:

1. After sorting the random key set in A3, A4 is divided by the row number of A3, so as to ensure the key set of each thread is concentrated.
2. The action of data fetching must be completed in each thread, so that the real parallel can be achieved.

Multithread Query - Oracle

Multi-threaded Parallel Approach to Further Improve Performance

	A	B
1	=10000.(1000000000000+rand(600000000)+1).sort()	
2	=A1.group((#-1)\1000)	
3	=now()	
4	fork A2	=connect("oracle")
5		=B4.query("select * from single600m where id in (?)",A4)
6		>B4.close()
7	=A4.conj()	
8	=interval@ms(A3,now())	

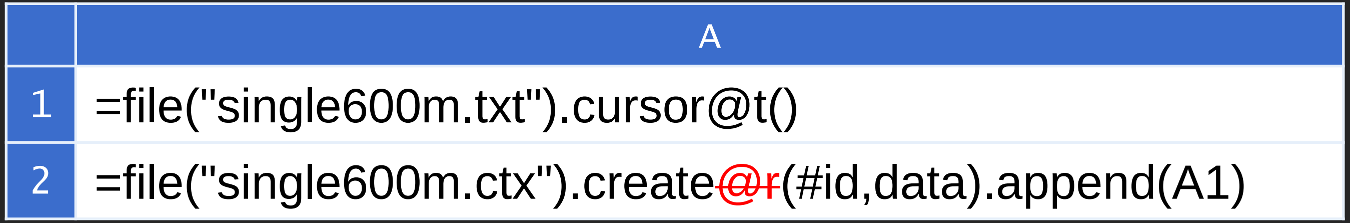
Grouping into several groups of 1000 entries because the maximum number of in in the database supports 1000 entries.

Index redundancy mechanism

Column storage is often used when data needs to be traversed, but it is not suitable for searching.

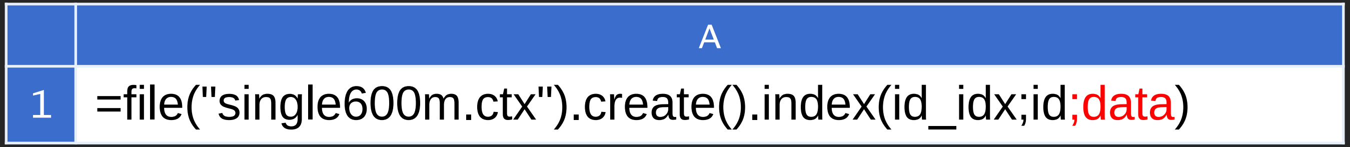
Index redundancy mechanism can be used to improve the random search performance of column-stored data.

Create column-stored group table



	A
1	=file("single600m.txt").cursor@t()
2	=file("single600m.ctx").create~~@r~~(#id,data).append(A1)

Create valued index



	A
1	=file("single600m.ctx").create().index(id_idx;id;data)

Use valued index file to search eliminates the need to read the original column-stored file. Although the efficiency is better than the row-stored non-redundant sort index, it needs to pay the cost that the disk space occupied is larger than the original column-stored file.

Comparison of testing results (1)

Extracting 100,000 batch random keys from 600 million pieces of data

Testing environment

Processor	Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz two core
Memory	64G
Hard disk	SAS 1TB
Operating system	centos6.8(64 bit)

Testing result

Time consumed（ms）					
Single thread			Multi-thread（10 threads）		
Oracle	Row-stored group table	Index redundancy	Oracle	Row-stored group table	Index redundancy
117322	20745	19873	39549	10975	9561

Comparison of testing results (2)

Extracting 100,000 batch random keys from 600 million pieces of data

Extracting 100,000 batch random keys from 1.2 billion pieces of data

Testing environment

Processor	Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz two cores
Memory	64G
Hard disk	SSD 1TB
Operating system	centos6.8(64 bit)

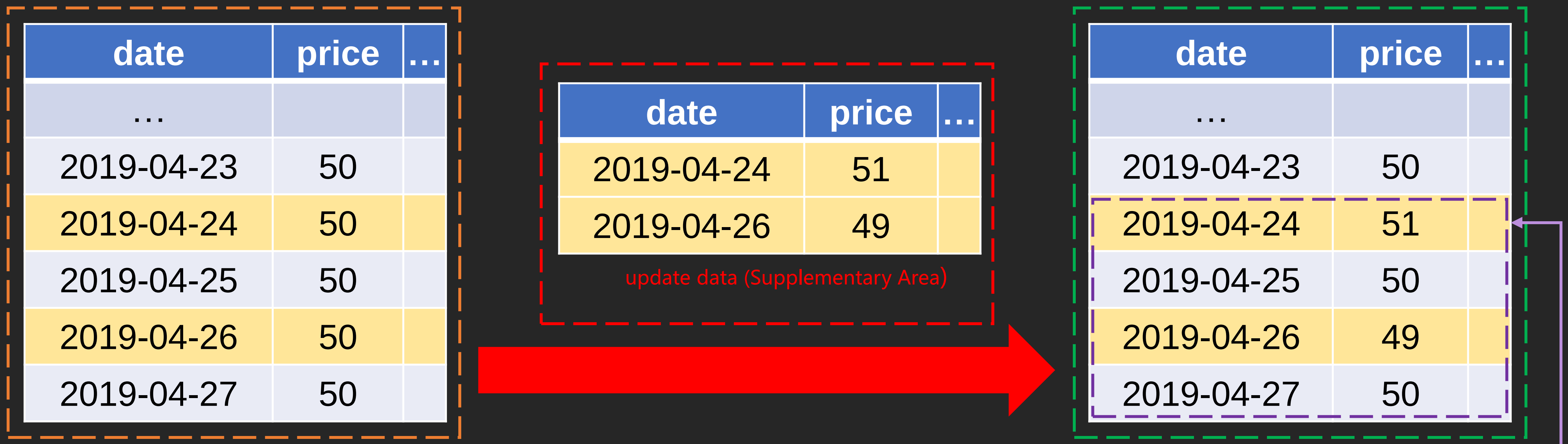
Testing result

Time consumed（ms）			
Single thread		Multi-thread（10 threads）	
Oracle	Row-stored group table	Oracle	Row-stored group table
56671	23990	35184	13264
Oracle	Row-stored group table	Oracle	Row-stored group table
151089	24421	95987	14623

Characteristic summary: The index performance of esProc is basically related only to the amount of data extracted, has little relation with the total amount of data, but the index performance of traditional database is very much related with the total amount of data.

Data update

Update of modified file data



date	price	…
…		
2019-04-23	50	
2019-04-24	50	
2019-04-25	50	
2019-04-26	50	
2019-04-27	50	

date	price	…
2019-04-24	51	
2019-04-26	49	

update data (Supplementary Area)

date	price	…
…		
2019-04-23	50	
2019-04-24	51	
2019-04-25	50	
2019-04-26	49	
2019-04-27	50	

Rapid reorganization means that only the part after the first complement data appears is reorganized. Previous data need not be rewritten.

When recent cumulative incremental data change

	A	B
1	=add_file.create().update(update_data)	/Update file data
2	=add_file.reset@q()	/Rapid Reorganization of Supplementary Area Data

Data addition

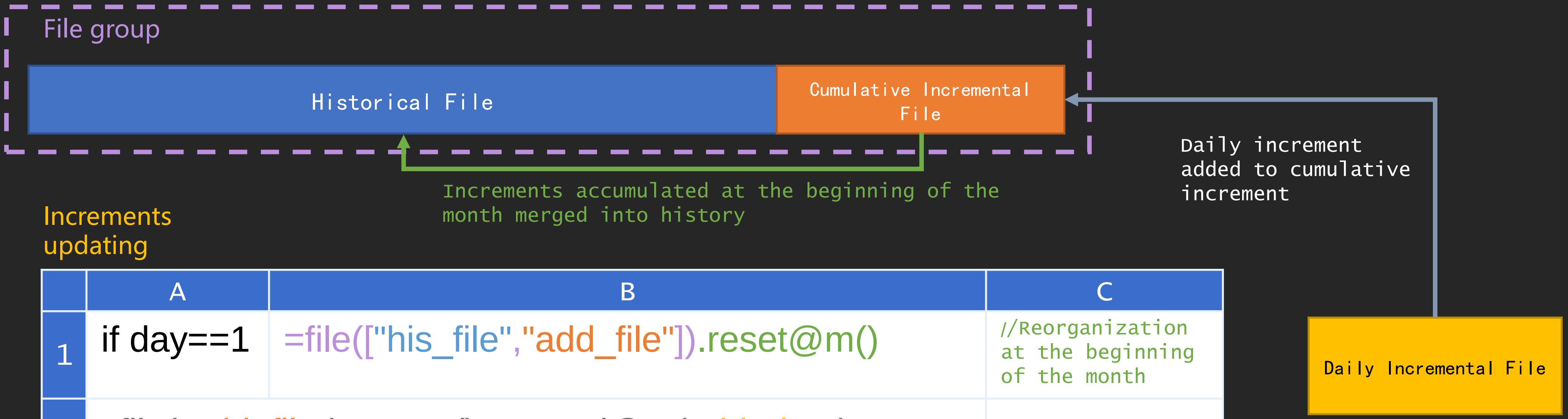
When keys are ordered, add new data directly.
When the keys are not ordered, new and old files need to get in order first, and then merge and sort.

	A	B
1	=file("single600m.ctx")	
2	=A1.create().cursor()	/Create group table cursor
3	=file("singleadd.txt")	
4	=A3.cursor@t()	/Added txt cursor
5	=file("single.ctx_temp").create(#id,data)	/Create new group table
6	=A5.append([A2,A4].mergex(id))	/After merging and sorting, the results are saved into the new group table

Note: The group table and TXT in A1 and A3 need to be ordered by ID.

Data addition

Addition of Daily Data Files


File group
Historical File
Cumulative Incremental File
Increments accumulated at the beginning of the month merged into history
Daily increment added to cumulative increment
Daily Incremental File

Increments updating

	A	B	C
1	if day==1	=file(["his_file","add_file"]).reset@m()	//Reorganization at the beginning of the month
2	=file("add_file").create().append@m(add_data)		/Daily supplement

File group query

	A	B
1	=file(["his_file","add_file"])	/File group
2	=A1.create().icursor(;id=="3197608180")	/Query

To be continued in the next chapter

- Search of Large Quantity of Random Key Values in Cluster

Coming soon