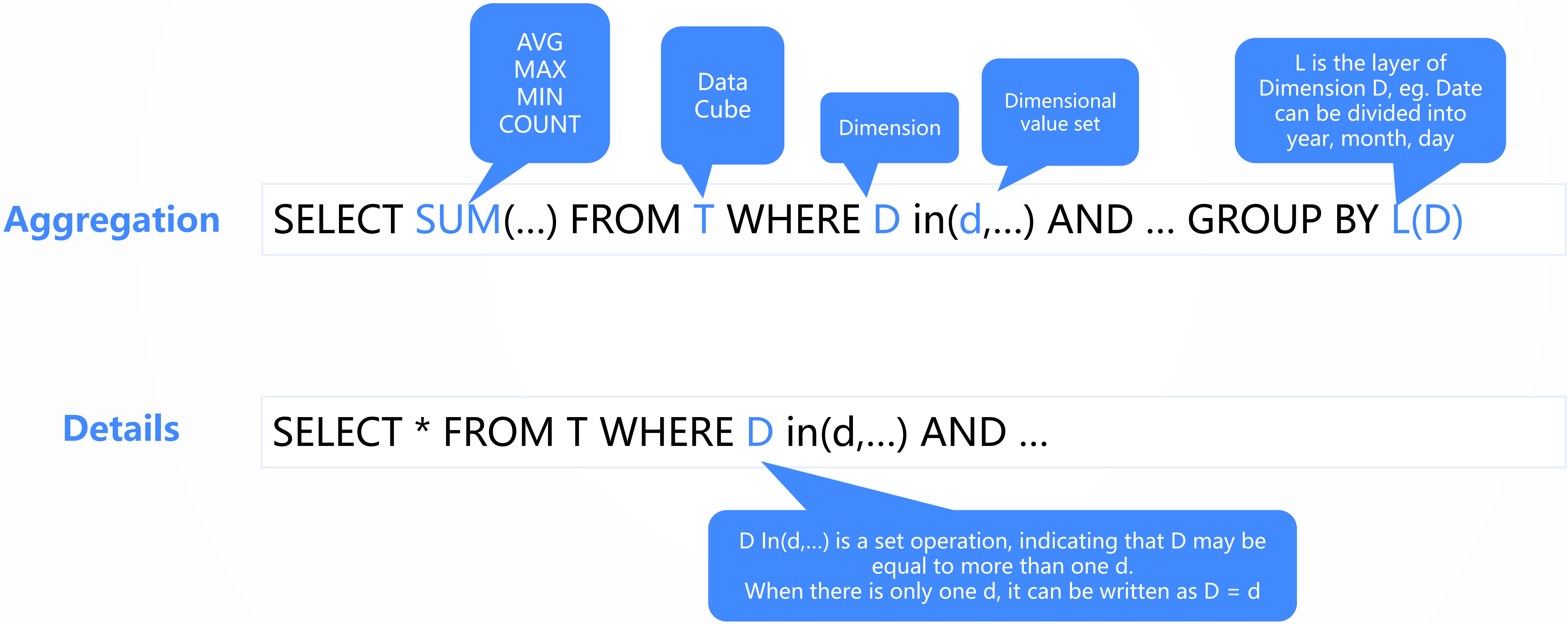
SPL Base

Performance optimization series

Multidimensional analysis

Basic form

Basic operations of multidimensional analysis



Aggregation

SELECT SUM(...) FROM T WHERE D in(d,...) AND ... GROUP BY L(D)

AVG MAX MIN COUNT
Data Cube
Dimension
Dimensional value set
L is the layer of Dimension D, eg. Date can be divided into year, month, day

Details

SELECT * FROM T WHERE D in(d,...) AND ...

D In(d,...) is a set operation, indicating that D may be equal to more than one d.
When there is only one d, it can be written as D = d

CONTENTS

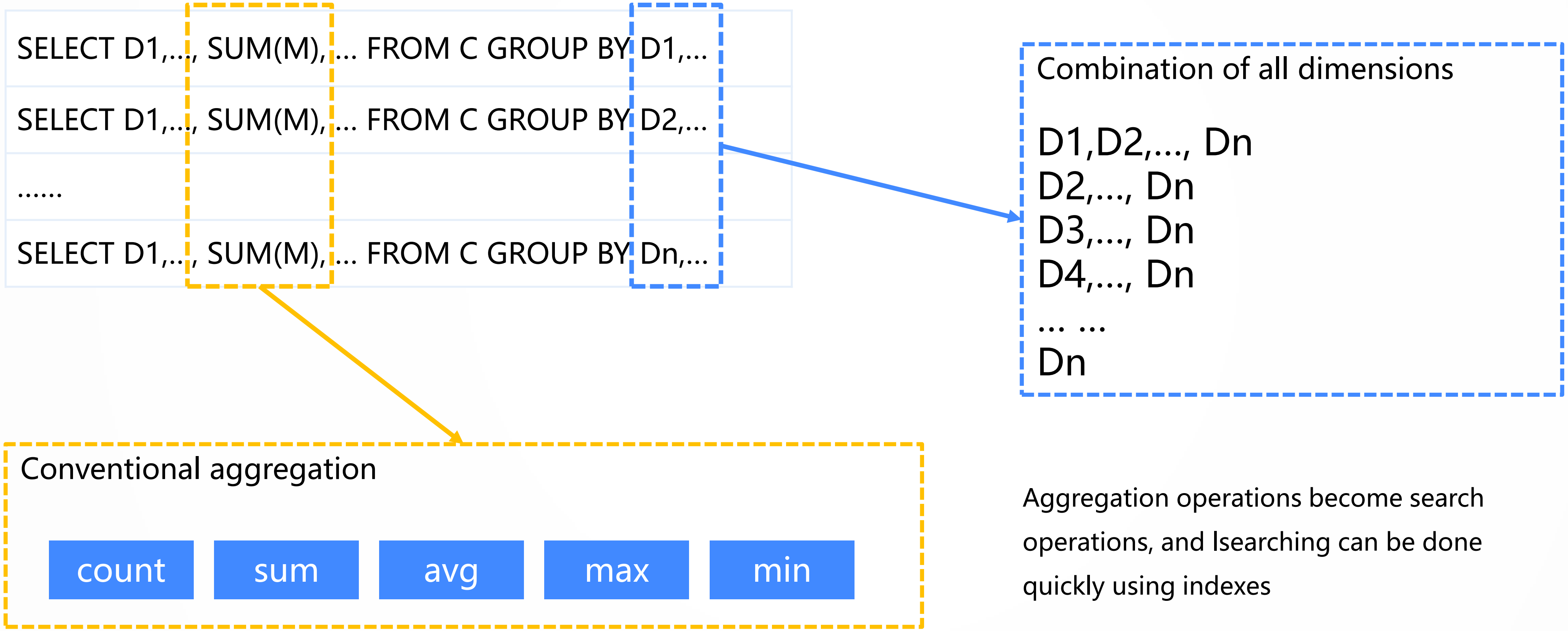
1 Aggregation

2 Slice

3 Cold and hot data routing

Full pre-aggregation

Early approach: Predict the aggregate values of various dimensional combinations, save them in advance, and refer to them directly when accessing.



```
SELECT D1,..., SUM(M), ... FROM C GROUP BY D1,...
SELECT D1,..., SUM(M), ... FROM C GROUP BY D2,...
......
SELECT D1,..., SUM(M), ... FROM C GROUP BY Dn,...
```

Combination of all dimensions

D1,D2,..., Dn
D2,..., Dn
D3,..., Dn
D4,..., Dn
... ...
Dn

Conventional aggregation

count sum avg max min

Aggregation operations become search operations, and Isearching can be done quickly using indexes

Space estimation

There is too much space for all dimensions to be combined.

A CUBE has n independent dimensions, and there are 2^n combinations of dimensions.

Suppose an intermediate cube occupy 1K space (in fact, a cube can not occupy only 1K), when n = 50:

Needed space=1K×2^{50}=1048576 T

Assume that no more than 20 dimensions (aggregation + slicing) are used at a time:

Needed space =1K×(C(50,1)+C(50,2)+…+C(50,20)) , still an enormous figure

conclusion:

When n <= 10 and the middle cube is not too large, the full amount of pre-aggregation can be done.

Functional Blind Zones of Pre-aggregation

The following cases are often used, where pre-aggregation is helpless.

- Unconventional aggregation

  Unconventional aggregation such as unique count, median and variance is often used, difficult to predict, and can not be calculated from other aggregation values.

- Aggregation of combinations

  There are too many combinations to predict, such as:

  TOP3 of average monthly sales = top(3,monthly sum(sales amount) / count())

  MAX of median monthly sales = max(monthly sort(sales amount)(count() + 1/2)))

- Conditional query

  Count the total amount of orders with value greater than 1000

  Count the total amount of orders with sales quantity greater than 50

- Time period statistics

  Count the total sales during the period 2018-6-10 to 2018-6-20

- D in(d,…) conditions

  There are too many combinations of (d,…) to be totally pre-aggregated

Partial pre-aggregation

ORDER_ID	DATE	DPT	SALE	…
……				

Pre-aggregate conventional aggregation for partial dimensions

Calculate aggregation for some commonly used dimensions and combinations and save them in advance, and the results can be returned directly when querying, or aggregate based on these saved intermediate results.

For example, Pre-aggregate according to date and department, and count the total sales.

	A	B
1	=user_file.create()	
2	=A1.cuboid(cu_1,DATE,DPT;sum(SALE))	/Create cube

When querying, if the sales are aggregated according to the date and department, then the results can be returned directly.

	A
1	=user_file.create()
2	=A1.cgroups(DATE,DPT;sum(SALE))

If the sales are aggregated by date, they can be aggregated again on the pre-aggregated results.

	A
1	=user_file.create()
2	=A1.cgroups(DATE;sum(SALE))

When the condition is set in, for example, aggregate the total sales of department 1, 3, 5, i.e. department in (1, 3, 5), can also be achieved by re-aggregation.

	A
1	=user_file.create()
2	=A1.cgroups([1,3,5].contain(DPT);sum(SALE))

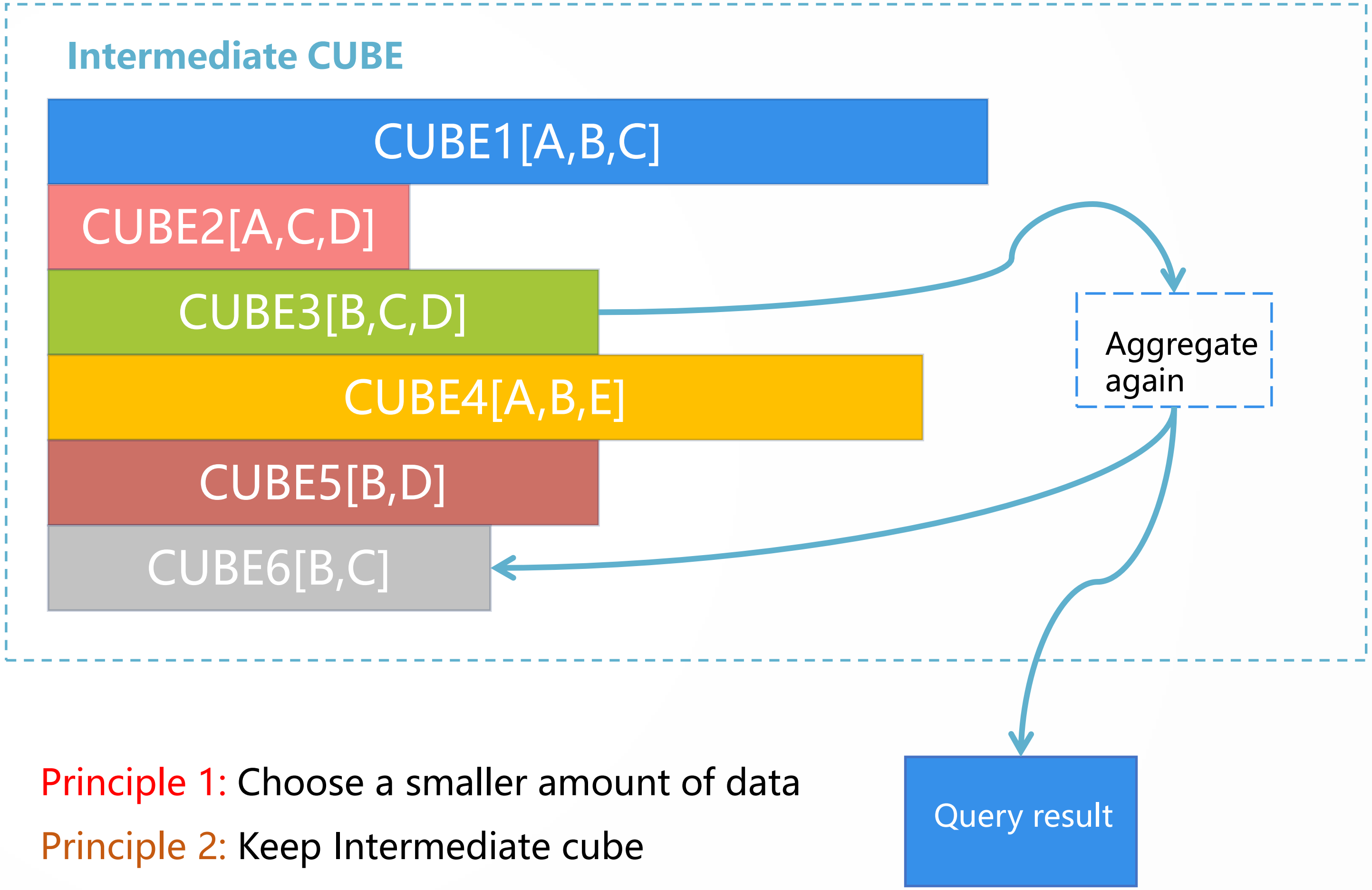
Partial pre-aggregation

Aggregate intermediate cube again

The re-aggregated results can also be preserved as an intermediate cube for future use.

When re-aggregating, if there are more than one intermediate result set available, choose the one with the smallest amount of data.

For example, to aggregate dimensions [B, C], two pre-aggregated cubes can be used: the aggregated dimensions of CUBE1 are [A, B, C], and the aggregated dimensions of CUBE3 are [B, C, D]. Both can aggregate [B, C]. CUBE3 should be chosen at this time.



Principle 1: Choose a smaller amount of data

Principle 2: Keep Intermediate cube

Aggregation-Time period pre-aggregation

Aggregation of Time Dimension

DATE	SALE	...
......		
2018-06-19	1700	
2018-06-21	1600	
2018-06-29	1600	
2018-06-30	600	
......		
2018-11-01	2300	
2018-11-08	2300	
2018-11-11	1600	
......		

	A	B
1	=user_file.create()	
2	=A1.cuboid(month(DATE);sum(SALE))	/Create cube

Total sales from June 19 to November 11=

[6-19,6-30]、[11-1,11-11]、[July, October]

The sum of the total sales of the above three periods. Among them, [July, October] data have been aggregated monthly, and the amount of calculation can be neglected.

[6-19,6-30] and [11-1,11-11] have 23-day data to be aggregated, compared with the original 145-day data, the amount of calculation is only 15%.

	A	B
1	=user_file.create()	
2	=A1.cgroups(;sum(sales); Date>=date(“2018-6-19”) && Date<=date("2018-11-11"))	/Query total sales from June 19 to November 11

MONTH	SALE	...
1	35000	
2	31700	
3	21600	
4	16000	
5	26060	
6	38000	
7	62300	
8	72300	
9	41600	
10	56000	
11	60080	
12	0900	

Aggregation-Time period pre-aggregation

Principle of Time Period Aggregation Using Intermediate CUBE

- The aggregate dimension of the intermediate CUBE is [year, month and day], which can be aggregated into [year, month] or [year].
- Query conditions are time periods, which may not be aggregated again.

  For example:[2018-9-15,2018-10-18] There is no whole year or month in this period of time.

- The intermediate CUBEs that can be used may have multiple levels

  For example, in the period of [2016-11, 2018-2-14], we can use the data of the whole year of 2017, as well as the data of December 2016 and January 2018, which are from two CUBEs.

- To ensure data synchronization, that is, when the original table data is updated, the aggregation results of the intermediate CUBE should also be updated synchronously.

CONTENTS

1 Aggregation

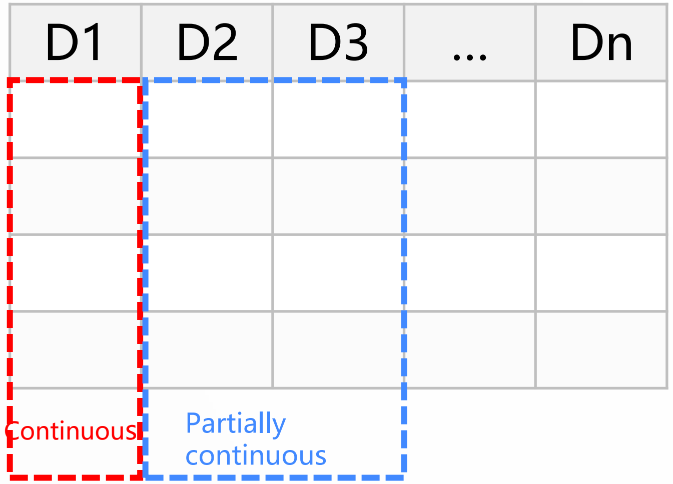
2 Slice

3 Cold and hot data routing

Slice-Redundant Sorting

Redundant sorting of data to ensure the continuity of data storage as far as possible.

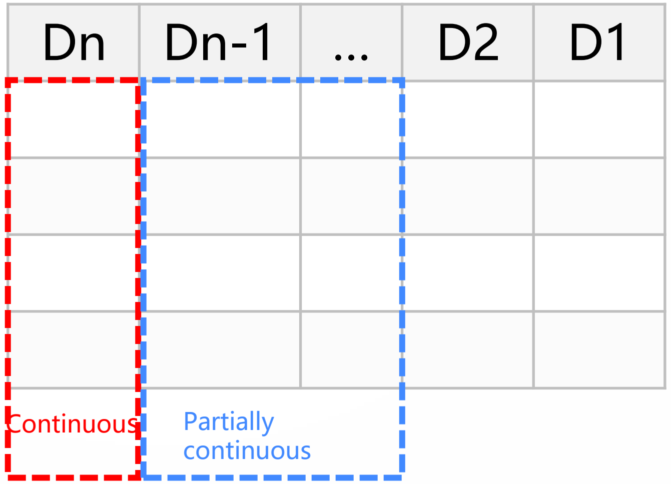
Save the data in two copies, one sorts according to dimension D1,…,Dn, another sorts according to dimension Dn,…,D1. In this way, we can always find a slice dimension in the first half of the two dimension-ranking columns, and the data after slicing is basically continuous. The performance improvement is still obvious.



D1	D2	D3	…	Dn

Continuous

Partially continuous



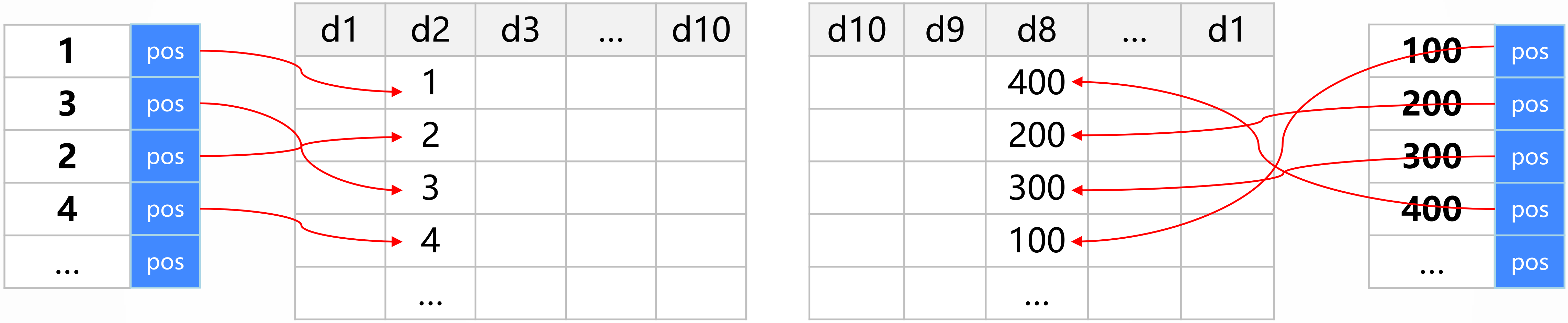
Dn	Dn-1	…	D2	D1

Continuous

Partially continuous

Slice-Redundant Sorting-Index

Accessing Redundant Ordered Data by Index



1	pos
3	pos
2	pos
4	pos
…	pos

d1	d2	d3	…	d10
	1			
	2			
	3			
	4			
	…			

d10	d9	d8	…	d1
		400		
		200		
		300		
		100		
		…		

100	pos
200	pos
300	pos
400	pos
…	pos

Create index

	A	B
1	=user_file.create()	/Open the file
2	=A1.index(id_2,d2)	/Create sorting index for d2
3	=A1.index(id_8,d8)	/Create sorting index for d8

Search for result

	A	B
1	=user_file.create()	/Open the file
2	=A1.icursor(;d2>0,id_2)	/Traverse d2 records
3	=A1.icursor(;d8>0,id_9)	/Traverse d8 records

Slice-Redundant Sorting

Choose the position of dimension in slice according to business data

D1	D2	D3	...	Dn
1	10	100		
1	10	108		
1	20	106		
2	15	100		
2	15	102		
2	16	90		
2	16	101		

Continuous

Partially continuous

Partial continuity refers to the composition of several continuous parts. For example, D1 is completely continuous, D2 consists of two consecutive parts and D3 has four consecutive parts.

It is not difficult to find that the continuity of dimension depends on the number of previous dimension values. If the number of Dn value is m, then Dn+1 may be divided into m continuous parts.

Therefore, we should try to choose the dimension with fewer values to put in front.

Slice-Dimension value filtering-Sequencing

Numbering the dimension values and then filtering bit by bit.

```
SELECT * FROM table1 WHERE AREA IN (Beijing, Hebei, Shandong, Guangdong, Fujian, Sichuan, Jiangsu,
Zhejiang, Shanghai);
```

The area set is converted to a truth table X, and the area in the set corresponds to true, while the area not in the set corresponds to false.

UID	AREA
10001	Shanghai
80022	Chongqing
20021	Zhejiang
00078	Guangdong
50001	Jiangsu
……	……

Conversion of area to natural number

AREA
12
19
14
22
13
……

1	2	3	4	5	6	7	…	12	13	14	15	16	…
T	F	F	F	F	F	F	…	T	T	T	F	F	…

Before conversion, to determine whether Shanghai is in the set, an average of 9/2 = 4.5 comparisons are needed.

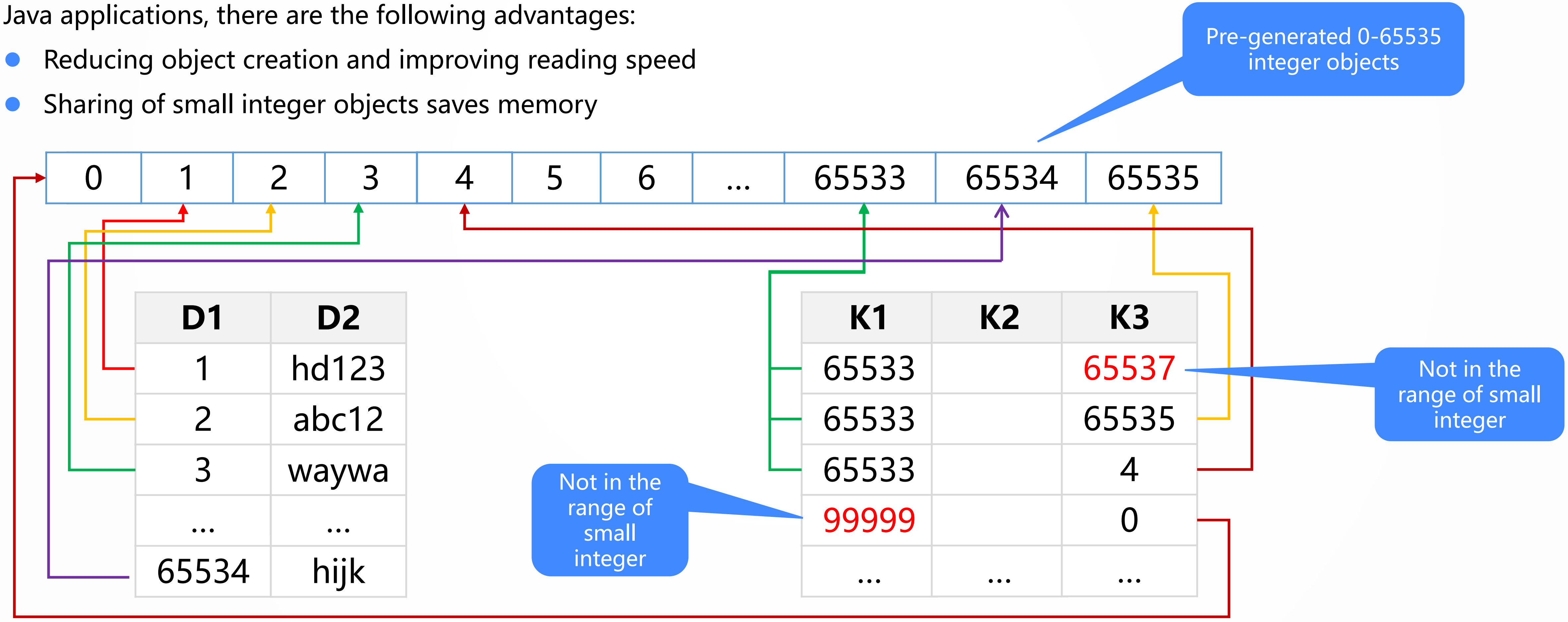
After conversion, according to Shanghai's serial number 12, you can look up the table once and get X(12)=true.

Slice-Dimension value representation

Pre-generation of small integer objects

Dimension values are mostly small integers, which can be generated in advance. For Java applications, there are the following advantages:

- Reducing object creation and improving reading speed
- Sharing of small integer objects saves memory



Pre-generated 0-65535 integer objects

0	1	2	3	4	5	6	…	65533	65534	65535

D1	D2
1	hd123
2	abc12
3	waywa
…	…
65534	hijk

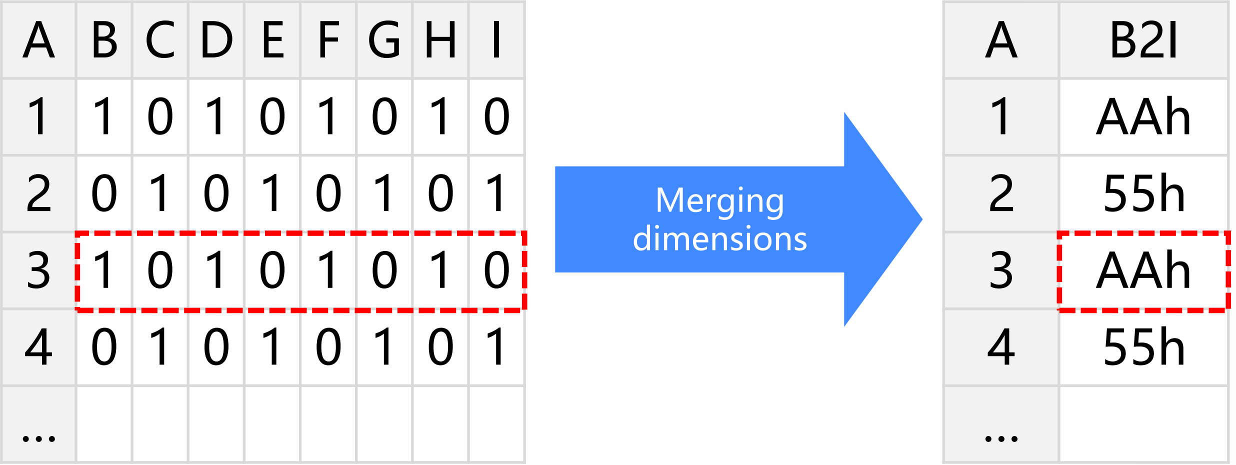
K1	K2	K3
65533		65537
65533		65535
65533		4
99999		0
…	…	…

Not in the range of small integer

Not in the range of small integer

Slice-Label Filtering

Use bit operation when there are many binary dimensions.



A	B	C	D	E	F	G	H	I
1	1	0	1	0	1	0	1	0
2	0	1	0	1	0	1	0	1
3	1	0	1	0	1	0	1	0
4	0	1	0	1	0	1	0	1
…								

Merging dimensions

A	B2I
1	AAh
2	55h
3	AAh
4	55h
…	

	A	B
1	=file1.cursor()	/Open the file
2	=A1.select(B==1&&C==0&&D==1&&E==0 &&F==1&&G==0&&H==1&&I==0)	/Conditional filtering
3	=A1.select(B==1&&D==1&&I==0)	/Conditional filtering

	A	B
1	=file1.cursor()	/Open the file
2	=A1.select(B2I==0xAA)	/Filter according to bit condition
3	=A1.select(and(B2I,0xC1)==0xC0)	/Filter according to bit condition

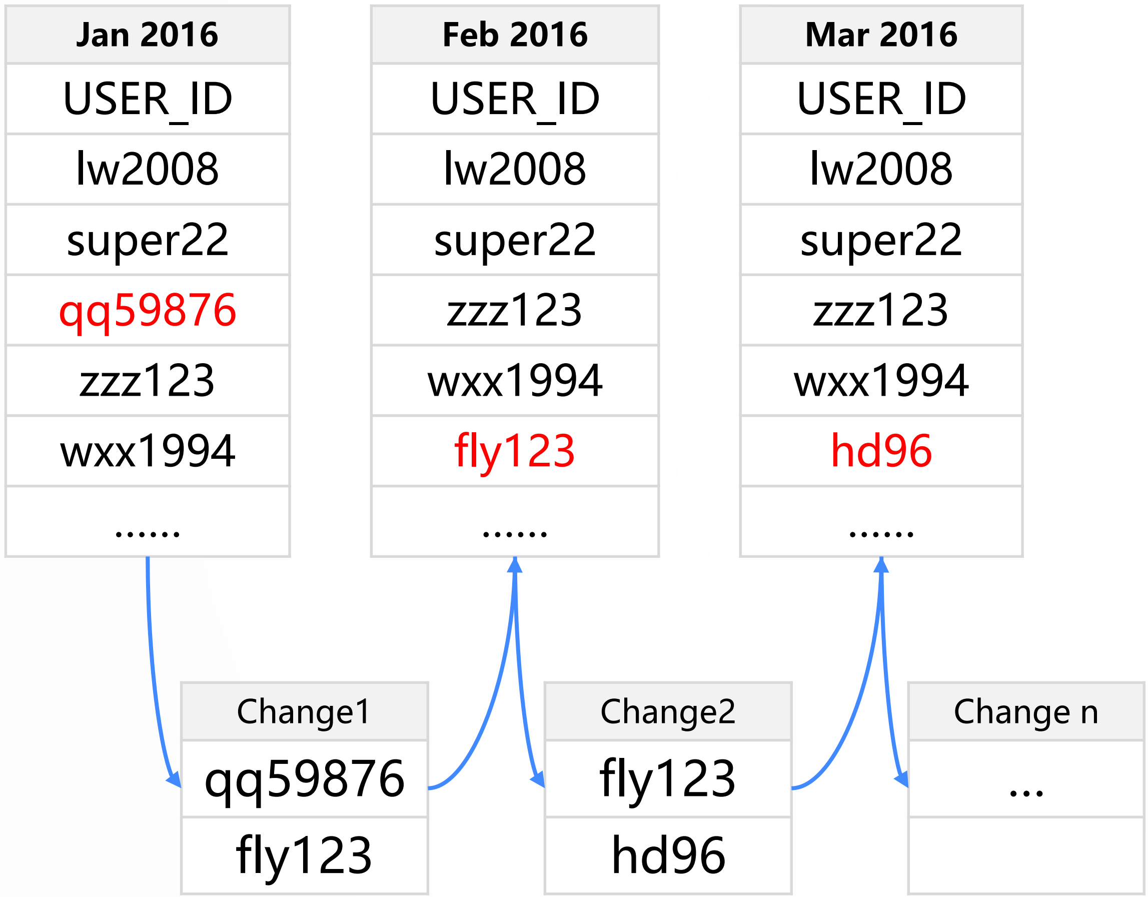
Slice-Set intersection

Regarding the set as a label

SELECT * FROM T WHERE K in(SELECT K FROM S1 INTERSECT SELECT K FROM S2)	S1 and S2 are subsets, and the computation is very slow
SELECT * FROM T WHERE S1 in S AND S2 in S	S is the set that current record belongs to
SELECT * FROM T WHERE S1 in (k1,k2,…) AND S2 in (k1,k2,…)	Represent S with (k1,k2,…)
SELECT * FROM T WHERE (S1=true & S2=true)	S1 and S2 are considered as bool fields
SELECT * FROM T WHERE (S1&S2)=1	Converting S1 and S2 into two-dimensional value fields

Slice-Change Label storage

Label data that does not change much over time only saves the changed part and reduces memory



Jan 2016
USER_ID
lw2008
super22
qq59876
zzz123
wxx1994
……

Feb 2016
USER_ID
lw2008
super22
zzz123
wxx1994
fly123
……

Mar 2016
USER_ID
lw2008
super22
zzz123
wxx1994
hd96
……

Change1
qq59876
fly123

Change2
fly123
hd96

Change n
…

	A	B
1	=data_201601.import().(USER_ID)	/Load January 2016 data
2	=data_change.cursor().fetch(5)	/Five-month change data
3	=[A1].insert(0,A2)	
4	=A3.xunion()	/Calculate June data

It is not necessary to load the monthly historical data, but only the first one. The latter is calculated on the basis of change.

February data = January data XOR change 1

March data = January data XOR change 1 XOR change 2

The nth month data = January data XOR change 1 XOR... XOR change (n-1)

Slice – Join filtering

For methods to improve Join performance, please refer to Join related documents

Completely in-memory pre-association

Sequence-number-based dimension table reference

Reuse index in filtering dimension table

Parallel

CONTENTS

1 Aggregation

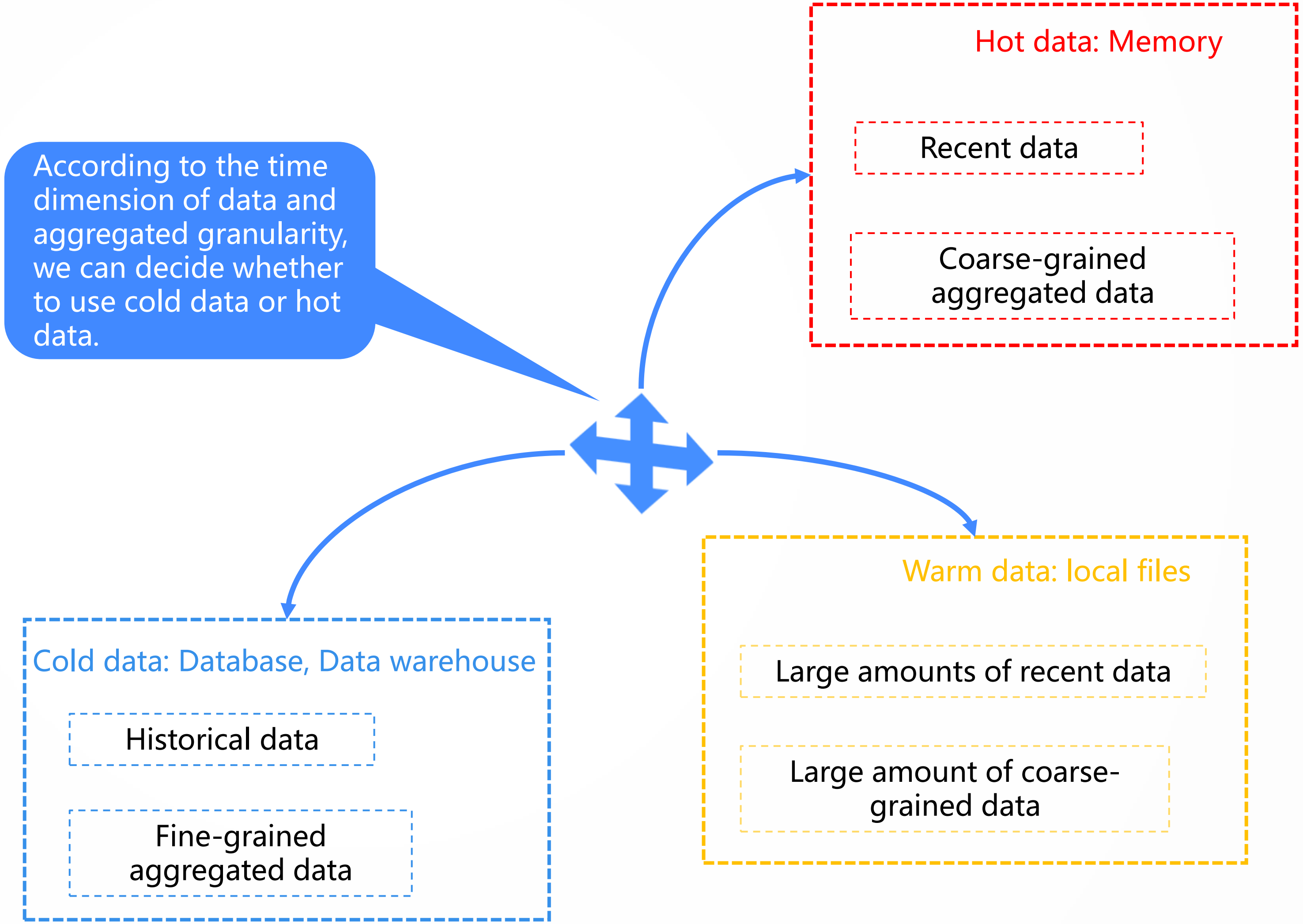
2 Slice

3 Cold and hot data routing

Cold and hot data routing

Data Routing

According to the data temperature, a hierarchical strategy is adopted: hot data with frequent and high concurrent access is pre-positioned, cold data with massive low frequency access is post-positioned, and controlled by data routing.



According to the time dimension of data and aggregated granularity, we can decide whether to use cold data or hot data.

Hot data: Memory

Recent data

Coarse-grained aggregated data

Cold data: Database, Data warehouse

Historical data

Fine-grained aggregated data

Warm data: local files

Large amounts of recent data

Large amount of coarse-grained data

Cold and hot data routing

According to the time dimension of the data, short-term data is hot and long-term data is cold.

For data with time dimensions, recent data can be loaded into memory.

	A	B
1	=env(hot_data, file(“data_2019.ctx”).create().cursor().fetch())	/Hot data in 2019 is totally loaded into memory

When querying, if the data belongs to 2019, access hot data.

	A	B	
1	=file(“data_history.ctx”).create()		/Cold data before 2019
2	if(year(key_date)==2019)	=hot_data.select(DATE==key_date)	/Hot data, in-memory query
3	else	=A1.cursor(; DATE==key_date).fetch()	/cold data, access file

Key_date is the date parameter entered.

Cold and hot data routing

For pre-aggregated data granularity, coarse-grained data is hot, and fine-grained data is cold.

For the pre-aggregation of time dimension, coarse-grained pre-aggregation can be loaded into memory.

	A	B
1	=env(data_year,file(“data_year.ctx”).create().cursor().fetch())	/Data pre-aggregated by year is loaded into memory

When querying, the hot data is accessed in case of yearly aggregation.

	A	B	
1	=file(“data_day.ctx”).create()		/Cold data aggregated by date
2	if(ifdate(key_date))	=A1.cursor(; DATE==key_date).fetch()	/cold data, access file
3	else	=hot_data.select(DATE==key_date)	/Hot data, in-memory query

Hot data is used if key_date is not date data.

THANKS

Issued by Raqsoft