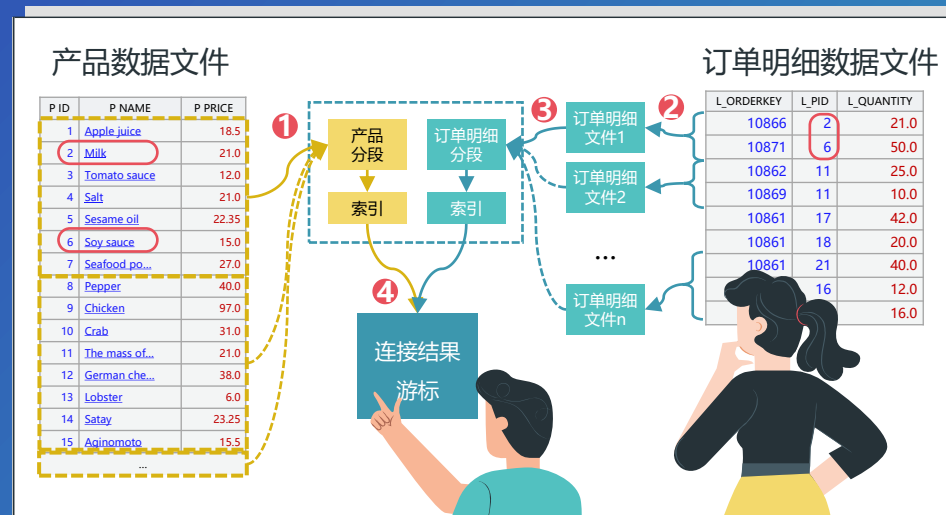


性能优化



课程目录

第一章 查找	11	哈希索引（外存）	44
1.1 单键值查找	14	1.2 多键值查找	46
二分法（内存）	16	键值排序	47
序号定位（内存）	18	索引查找行存文件	48
哈希索引（内存）	21	索引查找列存文件	49
位置索引（内存）	25	带值索引	52
多层序号定位（内存）	28	索引缓存	55
二分法（外存）	37	1.3 结果集查找	57
排序索引（外存）	40	索引返回多条	58
内外存查找技术类比	43	物理有序存储	60

数据更新	63	存储格式	90
1.4 多条件查找	69	并行和分段	93
区间查找.....	70	文本分段	96
多字段联合索引	73	二进制分段	99
多字段分别索引	76	区块分段	100
多条件次序	79	倍增分段	101
全文检索-排序索引.....	81	列式存储.....	104
全文检索-全文索引.....	83	列存倍增分段	106
第二章 遍历	86	有序压缩.....	107
2.1 存储方案	89	内存压缩.....	110

minmax索引	112	大排序.....	130
2.2 常规遍历	114	哈希大分组	131
延迟计算	115	聚合理解	134
游标前过滤	117	有序分组	137
过滤条件	119	有序去重	140
多路游标（内存）	121	半序分组	142
多路游标（外存）	123	半序排序	144
2.3 分组排序.....	125	序号小分组	146
小分组.....	126	序号大分组	149
大分组.....	127	序号大排序	152

利用索引排序.....	155
分组维度冗余.....	157
2.4 高级遍历.....	159
遍历复用.....	160
数据拆分.....	164
有序游标-字段变化.....	166
有序游标-条件变化	168
组内迭代	170
程序游标	174
手工并行.....	175

第三章 连接	177
3.1 连接运算理解	180
连接运算.....	181
传统计算方法.....	182
连接剖析.....	185
外键表	186
同维表	187
主子表	188
3.2 外键表	190
全内存外键预关联	191

全内存多外键预关联	195
全内存复制外键属性	197
全内存复制多个外键	199
仅维表全内存-临时指向	201
仅维表全内存-外键序号化	205
仅维表全内存-排号键连接	208
维表过滤-利用已建索引	212
维表过滤-对位序列	213
内连接-维表字段仅用于过滤	216
内连接-游标读取时关联过滤	217

内连接-关联过滤同时属性化	219
内连接-游标读取时关联过滤属性化.....	220
大维表、小事实表	222
维表事实表都很大-单边哈希手段.....	224

3.3 主子与同维表.....227

有序归并.....	228
并行有序归并.....	231
用主表过滤子表.....	234
用子表过滤主表.....	237
主子表一体化存储.....	240

有序和数据更新.....	243
3.4 子查询转换.....	245
子查询转连接.....	246
事实表包含维表部分主键.....	247
外键表的IN、 EXISTS.....	251
主子表的IN、 EXISTS.....	254
主子表， 关联字段过滤成逻辑主键.....	258
WHERE子查询转换为连接	261
集合运算-差集.....	264
集合运算-交集.....	268

同表关联, EXISTS非等值条件.....	271
子查询转换-小结.....	274

第四章 多维分析.....275

4.1 聚合.....	278
全量预汇总.....	279
部分预汇总.....	281
时间段预汇总.....	285
4.2 切片.....	289
布尔维序列解决切块.....	290
冗余排序.....	293

冗余排序-索引.....	294
二值维度（是否型）	296
预处理为小整数.....	298
随时间变化小的标签数据.....	299
4.3 数据路由.....	300
数据路由.....	301

综合练习一 高并发有关联查询	305
综合练习二 大明细表自关联.....	331
综合练习三 实时指标计算.....	354
附录 SPL和组表.....	381

适合人群

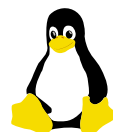
- ✓ 在大数据查询统计业务中遇到性能问题，需要优化提速的同学。
- ✓ 了解编程基础知识的同学，例如：变量、分支、循环等。
- ✓ 了解结构化计算基本概念的同学，例如：字段、记录、遍历、连接等。

课程准备

环境



Windows 64位 或



Linux 64位PC

数据

每章都有课堂练习需要的数据，数据量有点大，提前准备好

工具

到乾学院下载，并安装润乾集算器*

思想

先听原理讲解，再思考提速方法，还要动手练习

思想准备
也很重要哦

*本课程讲解性能优化算法，代码示例将用集算器SPL实现。采用其它语言要么实现不了、要么过于复杂，不适合用于教学。

因此，学员需要先简单了解集算器SPL语法，在充分理解算法后可以换用其它高级语言自行实现。

第一章 查找

1.1 单键值查找

1.2 键值查找

1.3 结果集查找

1.4 多条件查找

课前准备1-硬件和主目录

硬件

内存：8G以上 硬盘：空余30G以上 CPU：主频1G以上

主目录

在硬盘适当位置解压缩"seek.zip"文件夹，并将集算器主目录
设置成"seek"文件夹

设置方法：集算器-菜单-工具-选项-主目录

课前准备2-生成数据文件

要求

上课前执行并读懂生成数据文件的脚本，回答三个问题：

A, 数据文件数据量是多少？有哪些字段？

B, 有哪些字段，生成后占用硬盘有多大？

C, 有什么特征：有序、列存、行存、分段等等。

执行脚本

- 1、"主目录\dfx\orders.dfx", 生成订单集文件"主目录\data\btx\orders.btx"。
- 2、"主目录\dfx\residents.dfx", 生成居民集文件"主目录\data\btx\residents.btx"。
- 3、"主目录\dfx\fulltext.dfx", 生成全文检索组文件"主目录\data\ctx\fulltext.ctx"。
- 4、"主目录\dfx\keyvalues.dfx", 生成键值组文件"主目录\data\ctx\col.ctx"。

第一章 查找

1.1 单键值查找

课堂练习p1.1-数据准备

打开p1.1.dfx，生成一百万数据量的客户序表customer，并随机取出1000个id、name

	A	B
1	abcdefghijklmnopqrstuvwxyz	
2	=to(1000000).new(~:id,rands(A1,5)/~:name)	=env(customer,A2)
3	=to(1000).(rand(1000000)+1)	=env(ids,A3)
4	=A2(A3).(name)	=env(names,A4)

Index	id	name
1	1	gksnb1
2	2	mnydu2
3	3	haczk3
4	4	xmqlw4
5	5	cfzdc5
6	6	bdovi6
7	7	yzbx7
8	8	ygzwc8
9	9	tcxfn9
10	10	qcctw10

customer序表

Index	Member
1	230682
2	17875
3	495748
4	923660
5	791917
6	537433
7	516557
8	52907
9	170939
10	710313

ids序列

Index	Member
1	oyouj230682
2	rvukw17875
3	daipv495748
4	yvrpw923660
5	xwclg791917
6	vcllc537433
7	ygoww516557
8	jphkv52907
9	wuwmo170939
10	ymxds710313

names序列

知识点-二分法（内存）

单键值查找：每次只查找一个键值

用二分法找用户id为82的记录

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

users表

本例中，顺序查找（遍历）需要9次比较，二分法只用了3次比较。

顺序查找的时间复杂度为 $O(n)$;
二分查找的时间复杂度为 $O(\log_2 n)$ 。

代码示例

	A	B
1	=users.select@b(id==82)	/用二分法找id等于82的成员
2	=users.select(id==82)	/用遍历法找id等于82的成员

课堂练习p1.2-二分法（内存）

练习：打开p1.2.dfx，用二分法改写，记录执行时间

	执行时间（毫秒）
遍历	
二分法	

	A	B
1	=now()	
2	for ids	=customer.select(id==A2)
3	=interval@ms(A1,now())	

	A	B
1	=now()	
2	for ids	=customer.select@b(id==A2)
3	=interval@ms(A1,now())	

思考：这样写会出现什么问题？为什么？如何解决？

注意：
后续练习都用interval
函数计算执行时间

	A	B
1	=customer.select@b(name=="Luke")	/找到name等于Luck的客户

补充阅读 关系数据库的理论基础是无序集合，不能保证数据在物理存储时的有序性，无法利用存储上的次序，理论上不能实现二分法查找。
各种数据库在实现时有可能在工程上做些优化，一定程度实现二分法，但不能确保。

知识点-序号定位（内存）

找到编号为9的用户信息

Index	id	score
1	1	84
2	2	2339
3	3	8512
4	4	3461
5	5	7795
6	6	4423
7	7	6337
8	8	5970
9	9	1498
10	10	10000

users表的序号

用户id正好和每行的序号相同，可以直接用序号来定位查找。

序号：表中每行记录对应的顺序号

当数据表中的键值本身就是序号时，直接使用序号索引即可找到对应记录。

用户id是从1开始的连续正整数，和每行的顺序号完全一致，所以对id的查找可以直接用序号定位。

代码示例

	A	B
1	=users(9)	/利用序号直接定位

知识点-序号定位（内存）

找到日期为2019-05-03的价格

Index	day	price
1	2019-04-25	22
2	2019-04-26	20
3	2019-04-27	24
4	2019-04-28	42
5	2019-04-29	41
6	2019-04-30	3
7	2019-05-01	7
8	2019-05-02	11
9	2019-05-03	22
10	2019-05-04	16

prices1表

日期day转为
从2019-04-24
开始的天数

Index	dayid	price
1	1	22
2	2	20
3	3	24
4	4	42
5	5	41
6	6	3
7	7	7
8	8	11
9	9	22
10	10	16

prices2表

查找时可以将日期参数转为序号
后，采用序号定位的方法。

键值转换为序号

Prices表中的日期是连续的，并且每天一条，所以可以转化成序号定位。
从prices1转换为prices2的计算是一次性的，不影响查找的性能。

日期查找

要查找的日期先要转换为序号，然后用序号直接定位。
转换计算和查找本身相比非常快，对性能的影响可以忽略。

代码示例

	A
1	=interval("2019-04-24","2019-05-03")
2	=prices2(A1)

课堂练习p1.3-序号定位（内存）

练习：打开p1.3.dfx，用序号定位法改写，比较执行时间

	执行时间（毫秒）
遍历	
序号定位	

	A	B
1	=now()	
2	for ids	=customer.select(id==A2)
3	=interval@ms(A1,now())	

	A	B
1	=now()	
2	for ids	=customer(A2)
3	=interval@ms(A1,now())	

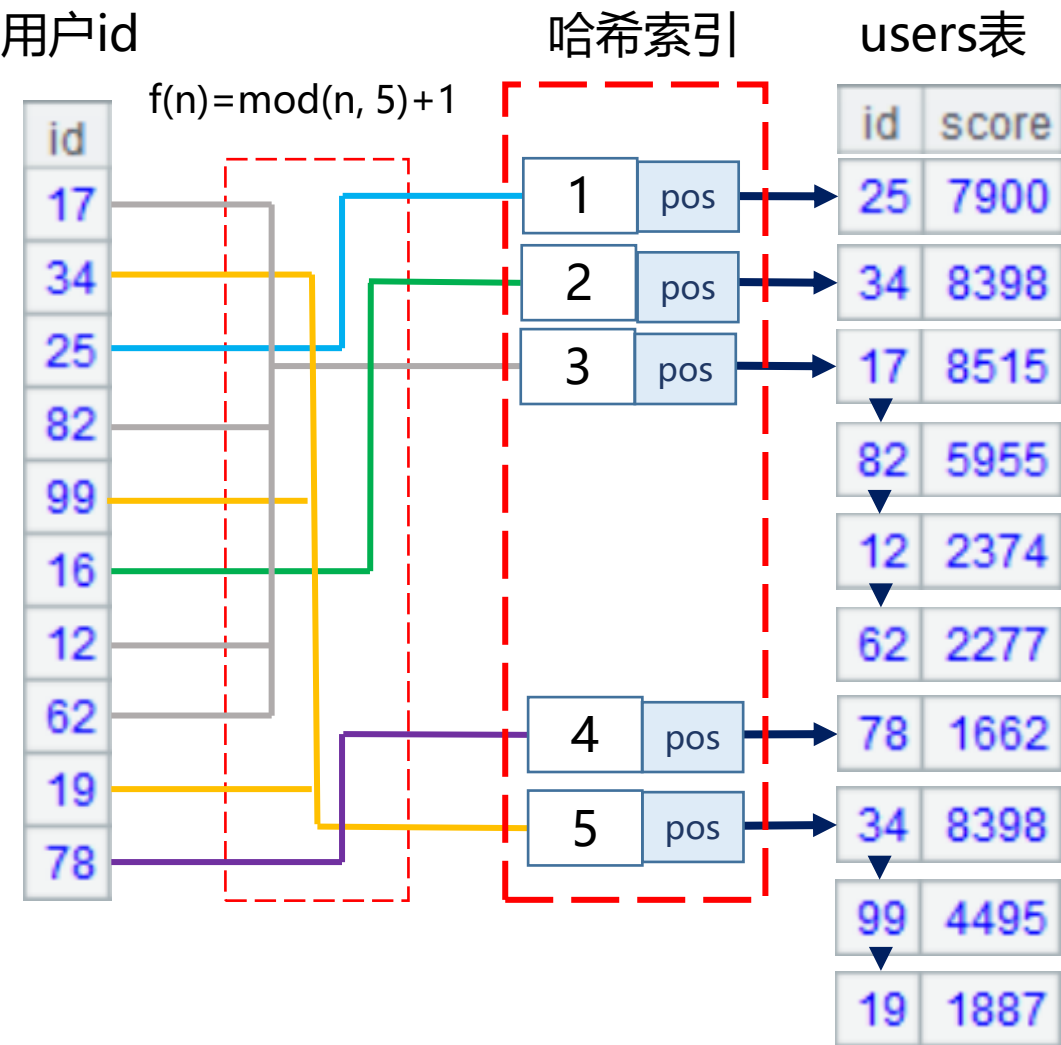
思考：如果客户id是从10000开始的，可以直接用序号定位吗？为什么？

这种情况如何转换为序号定位？

补充阅读 基于无序集合理论的关系数据库，没有提供序号定位的手段。即使可以用序号定位时，也只能用主键查找。

知识点-哈希索引（内存）

为无序键值-用户id生成哈希索引



什么是索引

索引类似于原表的"目录", 是在原表之外, 另外建立的存储结构。

查找时, 先查索引, 在"目录"中找到原表的位置, 再去原表找到对应的记录。

应用索引都要先建立索引, 然后才能利用索引查询。建立索引是个相对独立的过程, 可以放在系统初始化时一次性建成, 定时更新。

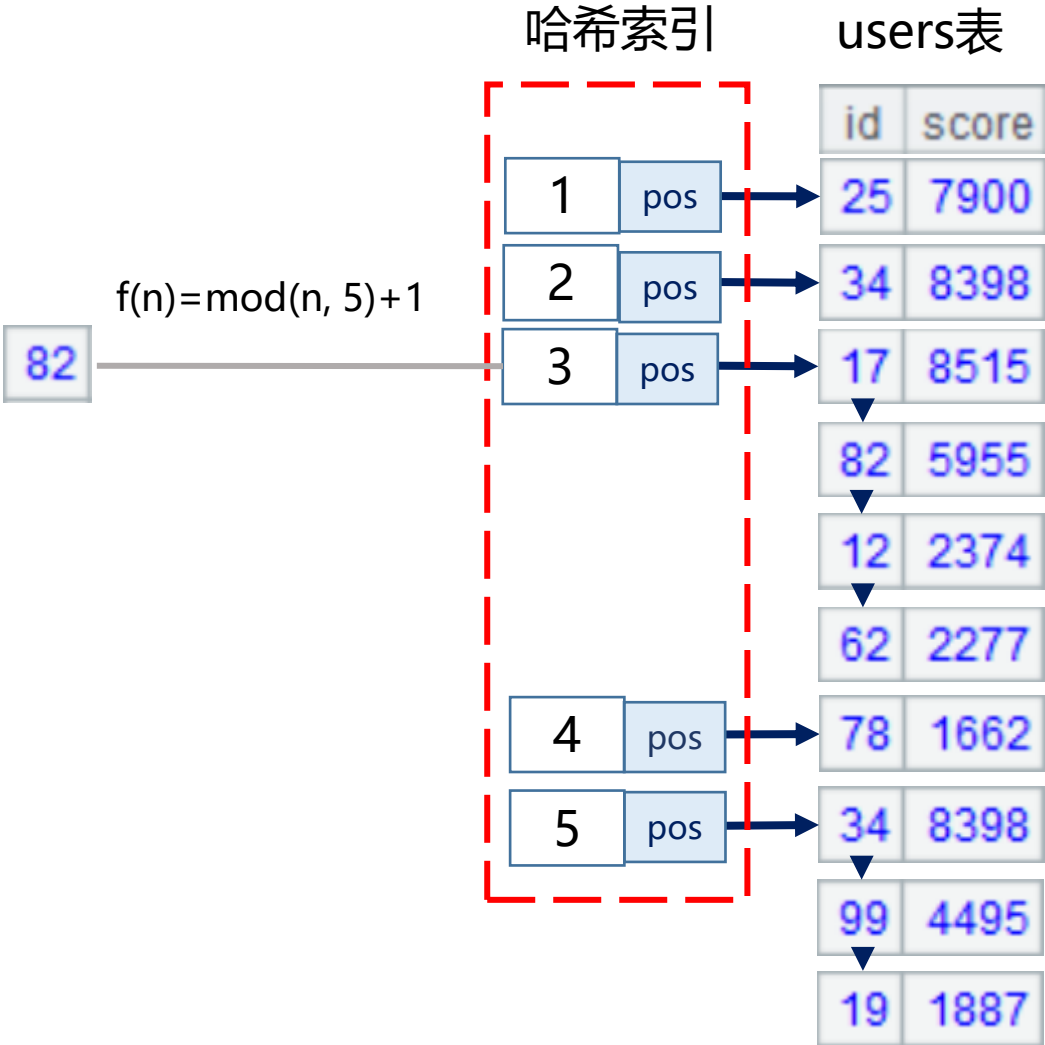
建立哈希索引

用户id无序, 通过 $f()$ 计算哈希值, 和原表的位置一起存储为哈希索引。

哈希函数不单调, 所以哈希值会出现重复, 如: id17、82、12、62都对应哈希值3。

知识点-哈希索引（内存）

找到用户id为82的用户信息



利用哈希索引查找

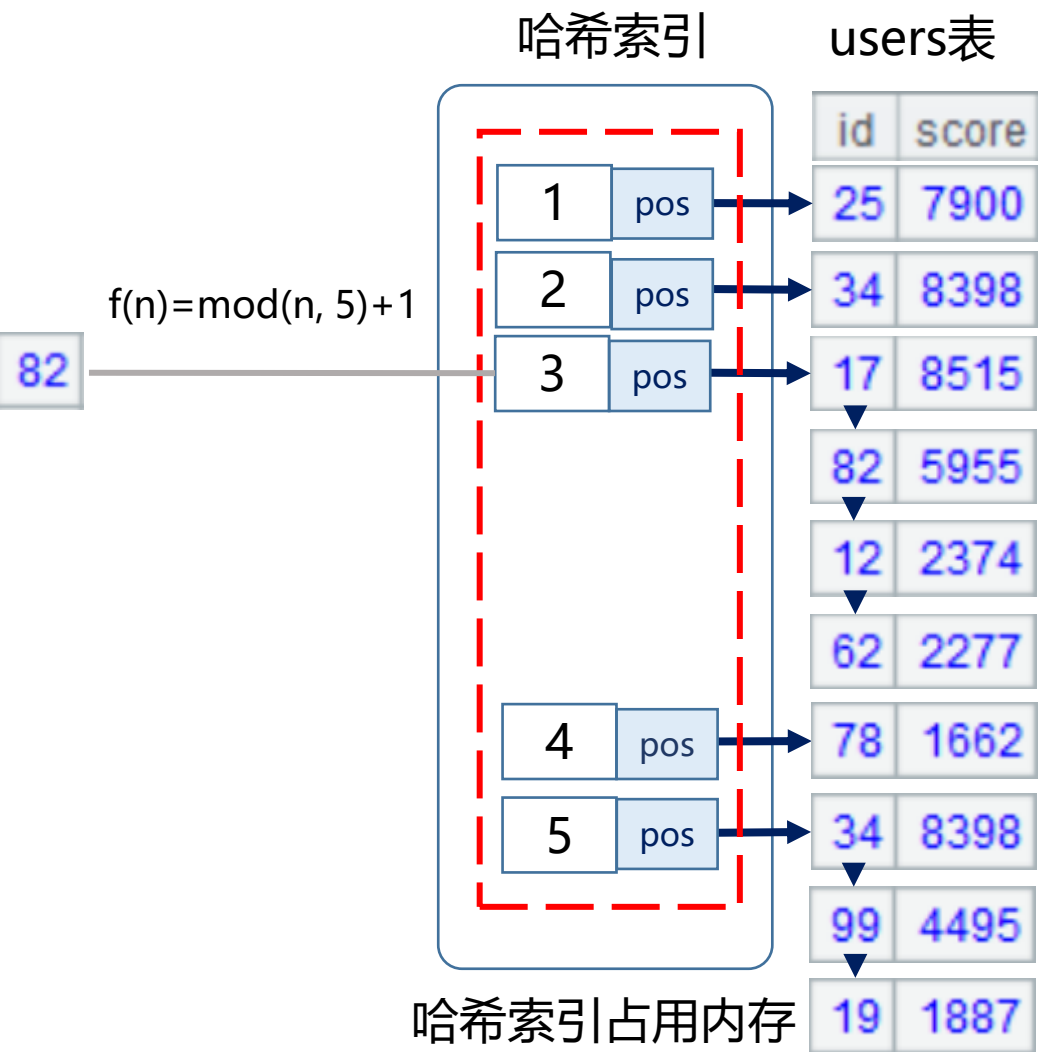
查找id时，用哈希f()算出哈希值。一个哈希值对应多个id的部分需要遍历，但比全表遍历快。

代码示例

	A	B
1	<code>=users.keys(id).index()</code>	/为id建立哈希索引
2	<code>=users.find(82)</code>	/在users表查找

知识点-哈希索引（内存）

使用哈希索引考虑的因素



占用内存和索引长度有关

索引长度越大，占用内存越大。集算器index(n)函数中的n就是索引长度。

重复率和索引长度有关

索引长度越长，键值对应哈希值重复的情况越少。重复越少，查找越快。

索引长度缺省是原序列长度和2000万的较小值。当users表长度为100万的时候，索引长度缺省就是100万，哈希值重复的情况就很少了。

索引长度的确定

索引长度需要因地制宜的确定。要考虑的因素包括数据的实际特征和硬件条件等。

索引长度太长会造成内存占用过多，太短会影响性能。要通过多次尝试找到性能最佳的索引长度。

课堂练习p1.4-哈希索引（内存）

练习：打开p1.4.dfx，用哈希索引改写

	执行时间（毫秒）
遍历	
哈希索引	

	A	B
1	=now()	
2	=customer.keys(id).index()	/建立索引
3	=interval@ms(A1,now())	
4	=now()	
5	for ids	=customer.find(A5)
6	=interval@ms(A4,now())	

思考：

- 1、建立索引也需要时间，怎么办？
- 2、A2改为index（10000），执行时间会如何变化？为什么？

知识点-位置索引（内存）

查找名字叫Julia的用户信息

Index	ID	NAME	GENDER	STATE	BIRTHDAY
1	2	Jonathan	M	Florida	1971-03-07
2	5	Joseph	M	Illinois	1971-07-02
3	3	Abigail	F	Arizona	1972-02-20
4	7	Emily	F	Michigan	1972-07-18
5	4	William	M	New York	1976-01-23
6	10	Samantha	F	Texas	1978-04-26
7	1	Megan	F	California	1979-04-19
8	9	Julia	F	Wisconsin	1980-06-30
9	6	Hannah	F	Pennsylvania	1980-08-27
10	8	Joshua	M	Montana	1985-07-10

生日有序的users表

序号定位

Index	ID	NAME	GENDER	STATE	BIRTHDAY
1	3	Abigail	F	Arizona	1972-02-20
2	7	Emily	F	Michigan	1972-07-18
3	6	Hannah	F	Pennsylvania	1980-08-27
4	2	Jonathan	M	Florida	1971-03-07
5	5	Joseph	M	Illinois	1971-07-02
6	8	Joshua	M	Montana	1985-07-10
7	9	Julia	F	Wisconsin	1980-06-30
8	1	Megan	F	California	1979-04-19
9	10	Samantha	F	Texas	1978-04-26
10	4	William	M	New York	1976-01-23

pos_b

pos_idx(pos_b)

序号定位

Member
3
4
9
1
2
10
8
7
6
5

pusers

pos_idx

为什么要用位置索引

用户表按生日排序，无法再按name排序。查找name，不能直接用二分法，需要借助位置索引。

建立位置索引

先计算按name排序的索引序列pos_idx，再用这个序列建立按name排序的索引表pusers。

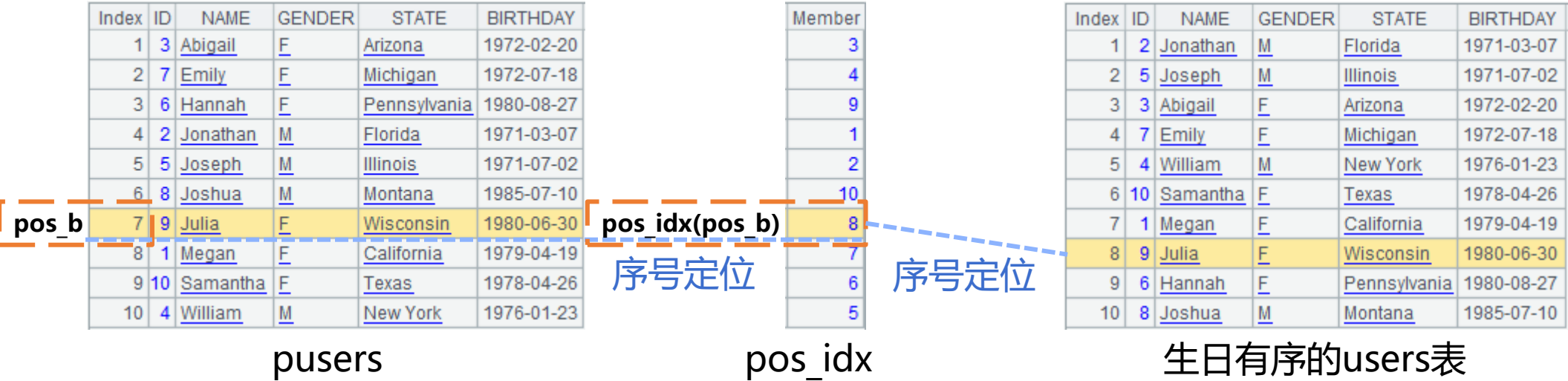
利用位置索引查找

查找用户name，因pusers对name有序，可以对puser用二分法查找。

找到name位置pos_b，再在pos_idx的pos_b位置找users表中的序号，从而找到users表中符合name条件的记录。

知识点-位置索引（内存）

查找名字叫Julia的用户信息



代码示例

(一次性) 建立索引
(复用) 索引查找

	A	B
1	> pos_idx=users.psort(NAME)	/按姓名排序后的位置索引号pos_idx
2	> pusers=users(pos_idx)	/pusers是users表在姓名有序时的索引表
3	> pos_b=pusers.pselect@b(NAME:"Julia")	/用二分法查找Julia在索引表中的序号
4	=users(pos_idx(pos_b))	/索引号的第pos_b个号码值就是原表中Julia的序号位

课堂练习p1.5-位置索引（内存）

打开p1.5.dfx-文件，观察遍历查找的写法，记录执行时间

	执行时间（毫秒）
遍历	
位置索引	

	A
1	=customer.select(name=="James")

练习：改写p1.5.dfx，建立索引

	A	B
1	>pos_idx=customer.psort(name)	>pcustomer=customer(pos_idx)

练习：利用位置索引改写p1.5.dfx，记录执行时间

	A	B
1	>pos_b=pcustomer.pselect@b(name:"James")	=customer(pos_idx(pos_b))

补充阅读 如前所述，基于无序集合理论的关系数据库，没有提供序号定位的手段。即使可以用序号定位时，也只能用主键查找。

知识点-多层序号定位（内存）

查找身份证号11010519730609816

idcard	name	...
.....		
11010519730609816	张三	
.....		
54211019840812023	李四	
.....		
64072219630218415	王五	
.....		

单层序号存在的问题

身份证18位，去掉校验位还有17位。

身份证号是不连续的，直接序号定位需要补齐成连续的序号。也就是说，需要10的17次方个Long型序号，占用内存过多。

多层序号定位要考虑的因素

多层序号适合键值本身很长，但是不连续，中间有很多空号的稀疏情况。

定位索引本身也要占用内存，如果键值本身不是很稀疏，占用内存还很大。

因此，是否采用多层序号，也要根据实际情况因地制宜，多次尝试来选择决定。

知识点-多层序号定位（内存）

身份证如何分多层

位数	含义	序号分层	序号说明
1、2位	省	1	两位整数
3、4位	市	2	两位整数
5、6位	区县	3	两位整数
7-10位	生日年	4	年份和1900年的间隔年数
11、12位	生日月	5	两位整数
13、14位	生日日	6	两位整数
15、16位	派出所	7	两位整数
17位	性别（奇数男）	8	一位整数
18位	校验位	无	由前17位计算得到，查找时忽略

第1层 说明	北京市 (京)	天津市 (津)	河北省 (冀)	山西省 (晋)	内蒙古 (蒙)	辽宁省 (辽)	...	新疆 (新)
	11	12	13	14	15	21	...	65

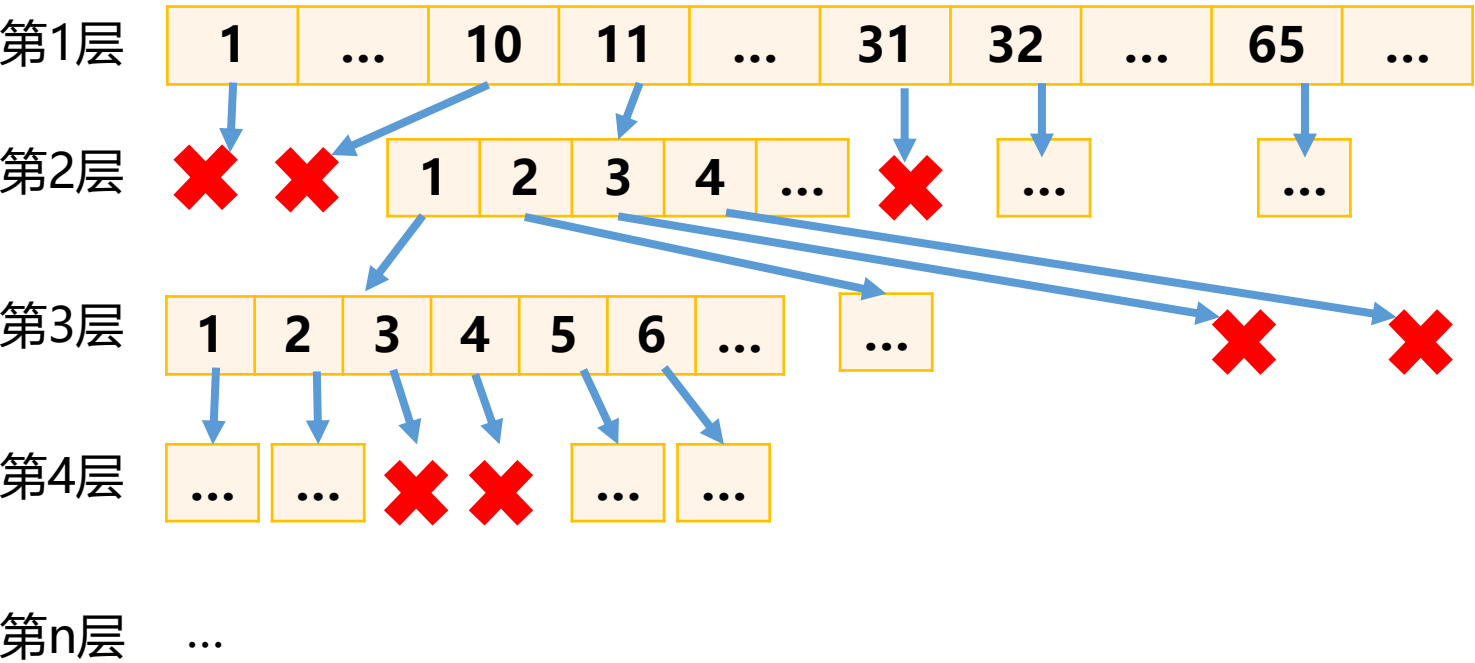
知识点-多层序号定位（内存）

单层序号定位占用内存示意图



最大值不确定时，单层序号最多占用10的17次方个Long型内存空间。

多层序号定位占用内存示意图



省级30多，很多序号没有下级节点。

北京的区县取值只有1、2。

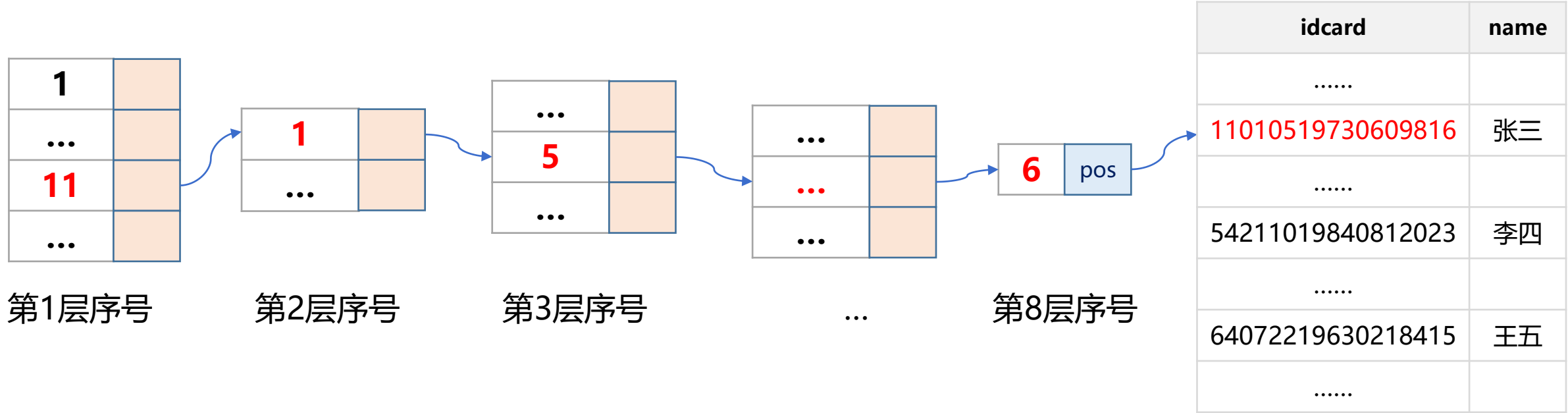
北京的各个区只有10个。

身份证号很稀疏，很多序号没有下级节点，因此树形的多层序号占用的内存比单层序号少很多。

如果键值不稀疏，多层序号占用的内存反而要多于单层序号。

知识点-多层序号定位（内存）

多层序号定位查找11010519730609816



代码示例-多层序号定位（内存）

排号键索引实现多层序号定位的步骤



代码示例

	A
1	=k(11,1,5...)

	A
1	=users.keys(idcard).index@s()

	A
1	=users.find(k(11,1,5...))

排号键索引的物理实现

在物理实现上，集算器排号键采用Long型（8个字节）实现最多8层序号。利用排号键建立的树形索引，实现多层序号定位。

相比集算器排号键索引，用Java的多层嵌套数组也可以实现多层序号，但复杂数组对象成本很高，会抵消序号定位的性能提升。

课堂练习p1.6-身份证转换为排号键

打开p1.6.dfx，居民表residents取出十万条数据，idcards随机取出100个idcard身份证号

练习：身份证号前十位转换为排号键代码如下，补全剩下的三位转换代码

	A
1	=k(int(mid(idcard,1,2)),int(mid(idcard,3,2)),int(mid(idcard,5,2)),interval@y("1900-01-01",date(mid(idcard,7,4),"yyyy")),int(mid(idcard,11,2)),int(mid(idcard,13,2)))

练习：改写p1.6.dfx，将residents表增加一个字段idcardk，存储身份证对应的排号键。

提示：使用derive函数。

课堂练习p1.7-哈希索引长度的作用

打开p1.7.dfx，用其中的遍历法查找10次和哈希索引查找10000次，记录时间

	执行时间（毫秒）
遍历法10次	
哈希索引	
哈希索引限制长度	

练习：将哈希索引长度改为原表长度的百分之一，模拟超大数据量，记录时间

思考：索引长度对性能有什么影响？对索引占用内存有什么影响？
为什么在超大数据量的情况下要限制索引长度？

课堂练习p1.8-排号键索引实现多层序号定位

打开p1.8.dfx，用其中哈希索引查找10000次，记录时间，
注意索引长度是1000

	执行时间（毫秒）
哈希索引限制长度	
多层序号定位	

练习：改写p1.8.dfx，将residents主键改为idcardk，哈希索引改为排号键索引

	A
1	=residents.keys(idcardk).index@s()

练习：改写p1.8.dfx，将idcards中的一百个身份证，用k函数转换为排号键
提示：使用derive函数。

课堂练习p1.8-排号键索引实现多层序号定位

练习：改写p1.8.dfx，用排号键索引，每次查找一百个身份证，共查找一百次，记录执行时间

提示：使用k()、find()函数。

思考：什么情况下使用多层序号定位？什么情况下使用哈希索引？

补充阅读 同样的，基于无序集合理论的关系数据库，没有提供序号定位的手段。即使可以用序号定位时，也只能用主键查找。没有单层定位的手段，也谈不上多层定位了。

知识点-二分法（外存）

有序文本文件查找oid是20100的订单

1 文件总字节数除以2，找到中间行。
oid比20100小。

2 向后 $\frac{1}{2}$ 的字节数除以2，找到中间行。
oid比20100大。

3 向前 $\frac{1}{4}$ 的字节数除以2，找到中间行。
oid比20100大。

oid	orderdate	...
1	2018-01-18	
3	2018-01-18	
...	...	
20003	2018-03-18	
20005	2018-03-18	
...	...	
20010	2018-03-21	
...	...	
25011	2018-04-18	
25013	2018-04-22	
...	...	
30507	2018-05-18	
30513	2018-06-19	
...	...	

内存和外存的差别

数据量大到内存装不下，需要外存（硬盘）文件存储。外存数据无法精确定位找到某一条数据。

有序文本文件二分法查找

文本文件用换行符确定一条记录。

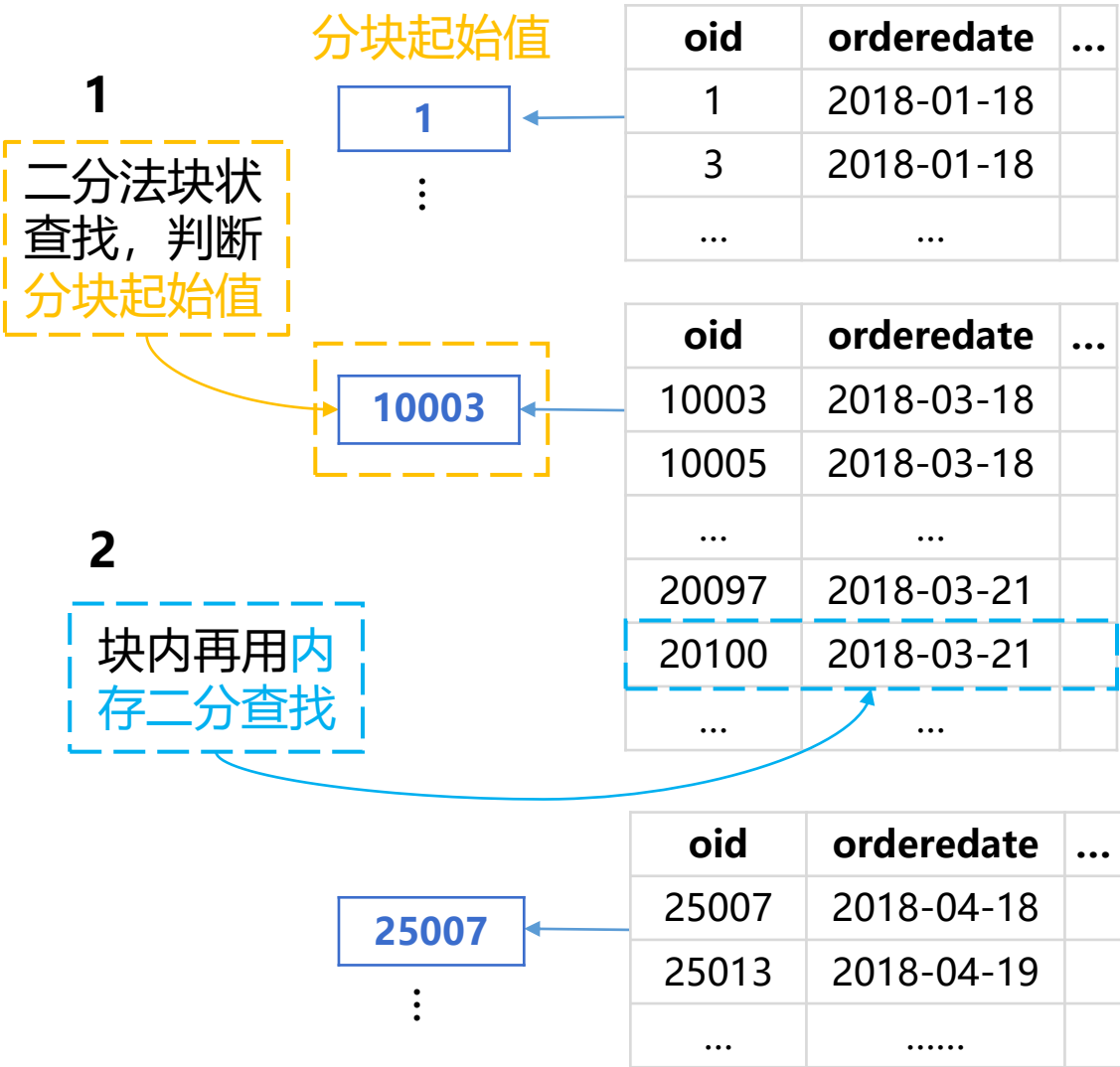
用文件总字节数除以2为中点，解析中点所在的记录，比较订单编号后，根据结果向前或者向后 $\frac{1}{2}$ 继续找中点，直到找到为止。

代码示例

	A
1	=orders_file_txt.iselect@t(20100,oid).fetch()

知识点-二分法（外存）

有序二进制文件查找oid是20100的订单



二进制文件和文本文件的差别

二进制文件有物理分块，文件头部记录了分块的起始位置。可以按照位置直接取得记录，比较起始的订单编号值。

有序二进制文件查找的方法

查找oid时，先用二分法查找分块，根据分块起始值判断键值是否在本块。如果在本块，把本块数据读入内存，用二分法查找键值。

代码示例

	A
1	=orders_file_btx.iselect@b(20100,oid).fetch()

课堂练习p1.9-二分法（外存）

练习：打开p1.9.dfx,用遍历法查找oid==8123456，记录执行时间

	执行时间（毫秒）
遍历	
二分法	

练习：利用二分法（外存）改写p1.9.dfx，比较性能

	A
1	=file("data/btx/orders.btx").iselect@b(8123456,oid).fetch()

思考：下列代码是否正确，为什么？

	A
1	=file("data/btx/orders.btx").iselect@b(date("2010-09-22"),orderdate).fetch()

补充阅读 前面说过，关系数据库的理论基础是无序集合，理论上不能实现二分法查找。
各种数据库在实现时有可能在工程上做些优化，一定程度实现二分法，但不能确保。

知识点-排序索引（外存）

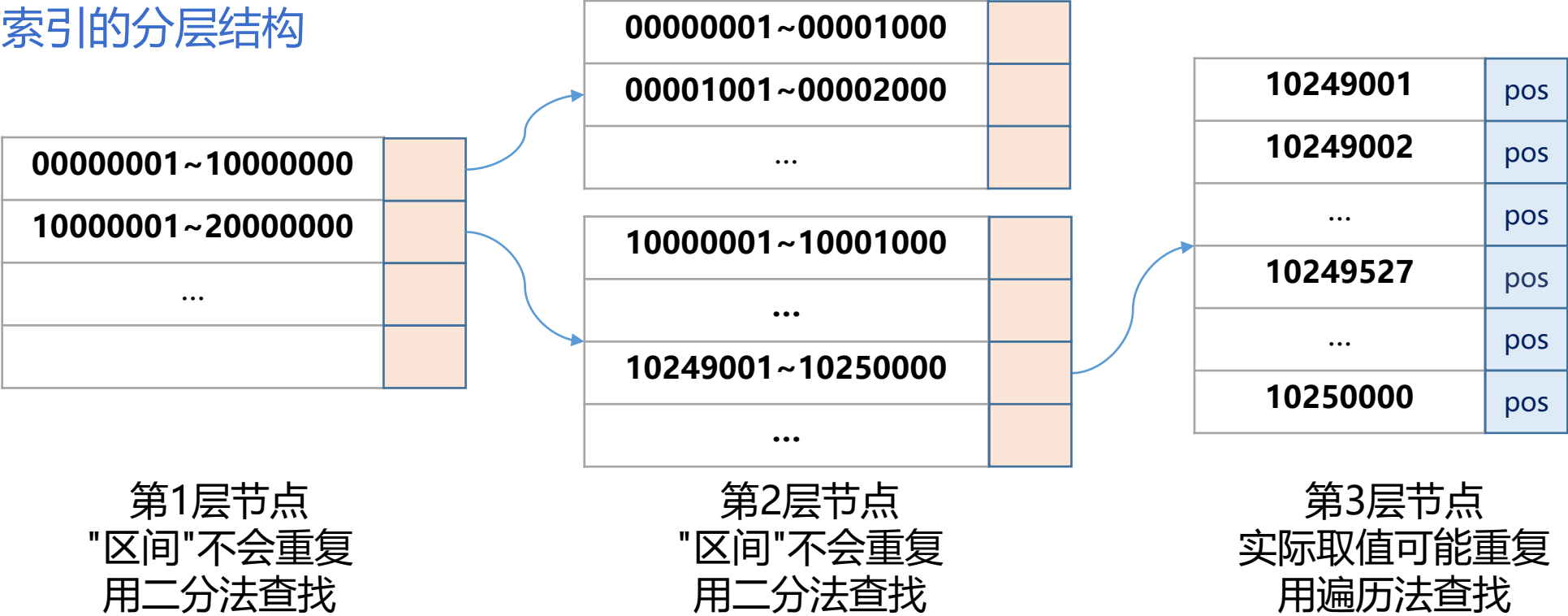
什么是排序索引

把用户编号和原表中的位置，按编号排序另外存放，叫做排序索引。查找时在排序索引中查找编号，得到在原表中的位置。

为什么排序索引要分层

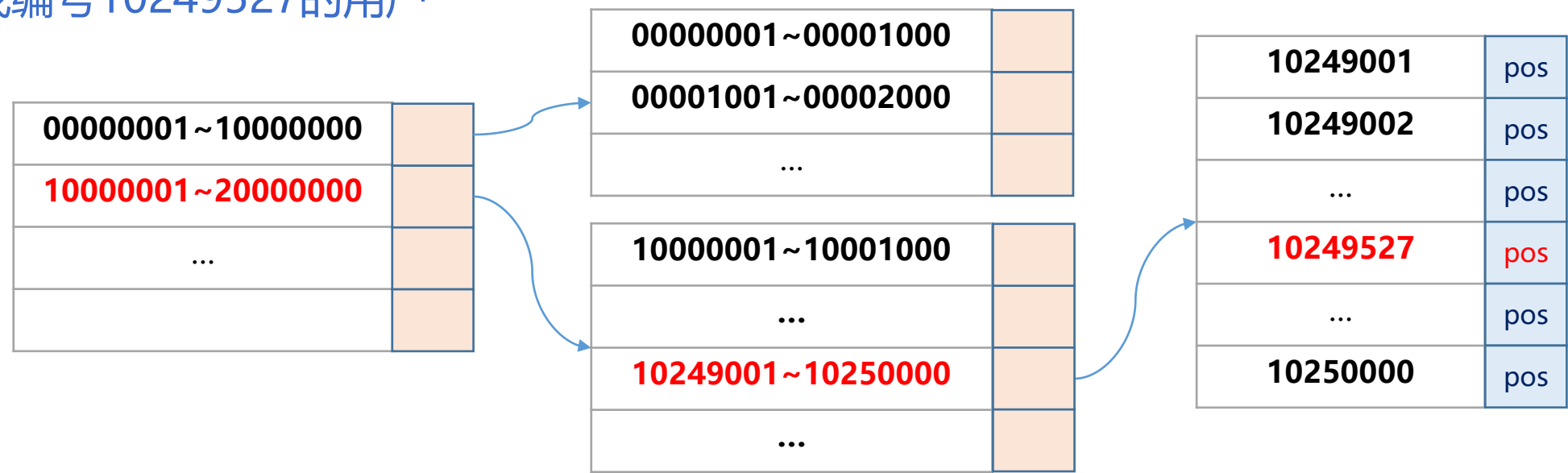
索引一般也较大，需要外存。
采用分层索引，利用键值的区间值逐层的查找。

排序索引的分层结构



知识点-排序索引（外存）

查找编号10249527的用户



代码示例

建索引

	A
1	=user_file.open()
2	=A1.index(id_idx; id)

查找

	A
1	=user_file.open()
2	=A1.icursor(;id==10249527,id_idx)

课堂练习p1.10-排序索引（外存）

练习：打开p1.10.dfx,无索引查找1000个随机的订单编号，记录执行时间

	执行时间（毫秒）
无索引	
排序索引	

练习：改写p1.10.dfx，使用排序索引查找，比较性能，查看产生的索引文件

提示：使用index()、icursor()

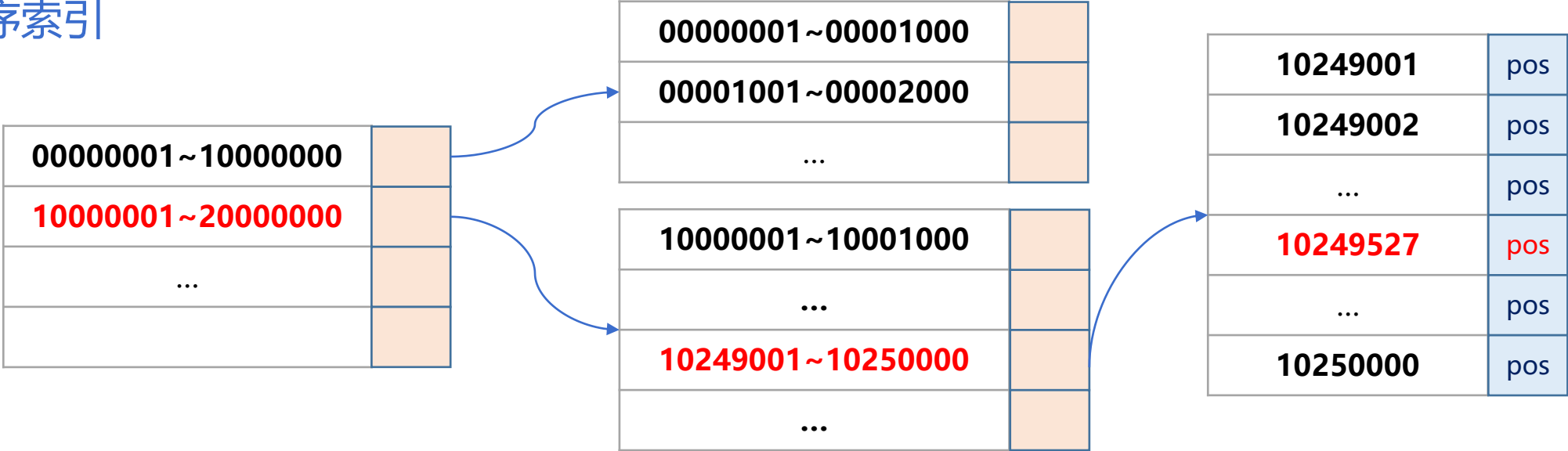
思考：

1、 p1.10中使用的组表是列存还是行存？ 打开orders.dfx找答案。

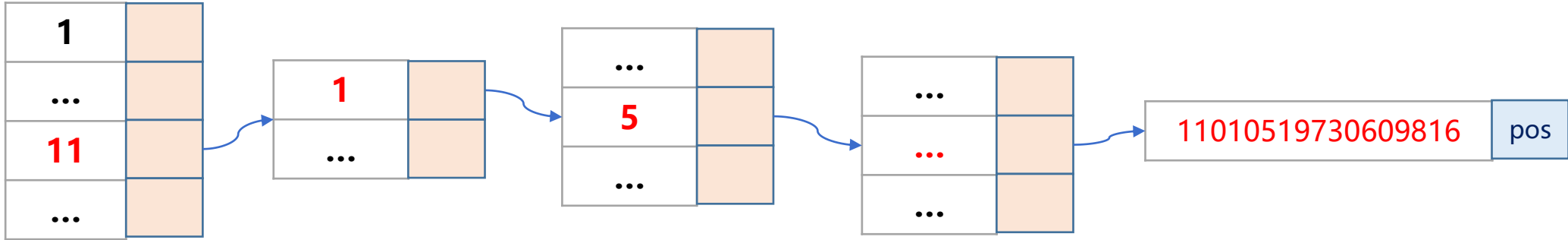
补充阅读 关系数据库的索引不是数据表，是个特殊对象。其本质是建了排序索引，记录数据的位置。利用索引的有序特征，可以实现二分法查询。

知识点-内外存查找技术类比

排序索引

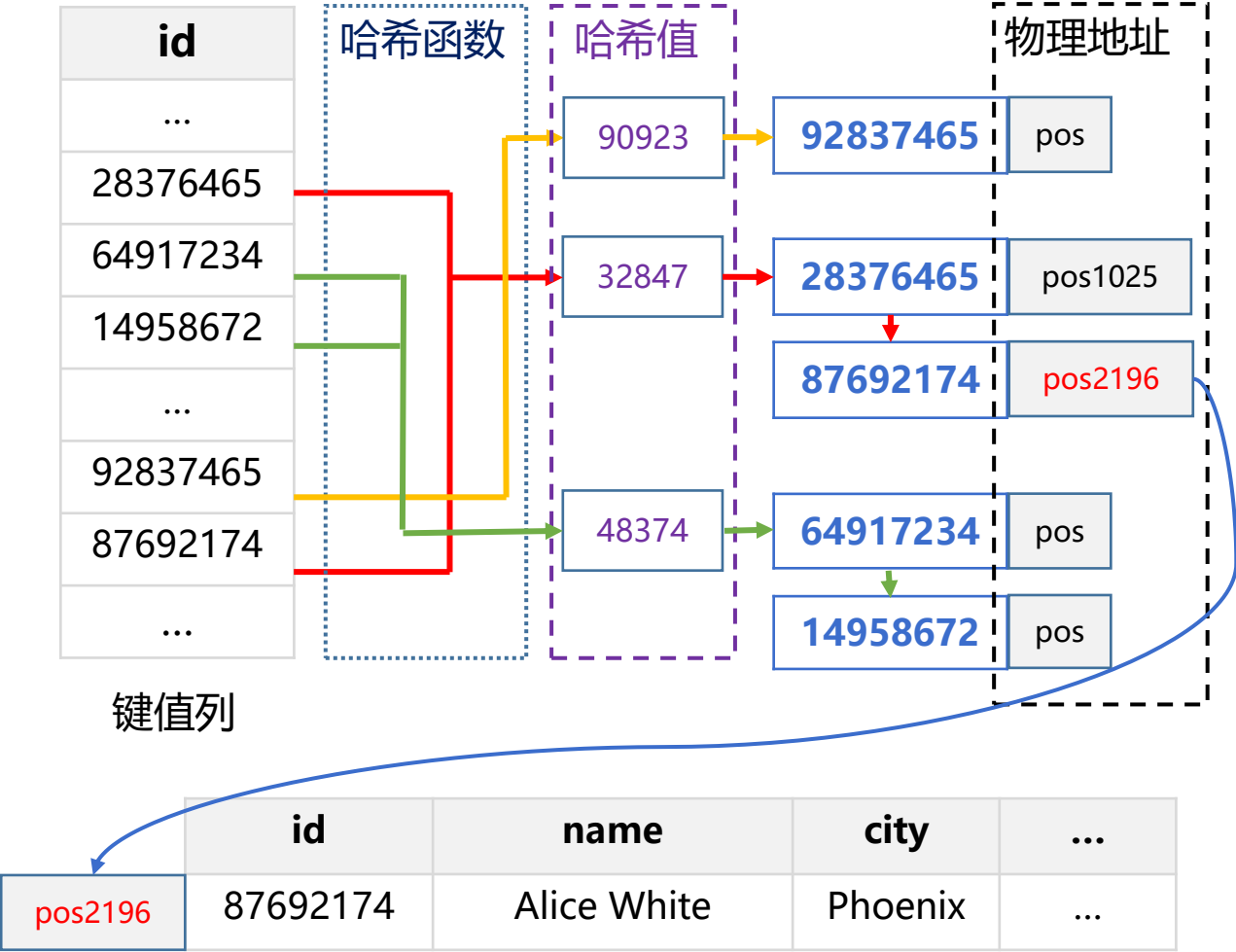


多层序号定位



知识点-哈希索引（外存）

查找编号为10249527的用户



哈希索引和排序索引的比较

哈希索引用键值的哈希值来排序，本质也是排序索引。哈希索引直接定位，排序索引还要在一片中做二分，所以哈希索引性能更快。

哈希函数适合单个键值查找，排序索引适合区间查找。

代码示例

建索引

	A	B
1	=user_file.open()	/打开文件
2	=A1.index(id_idx:hash_den,id)	/为id建立哈希索引

查找

	A	B
1	=user_file.open()	/打开文件
2	=A1.icursor(;id==10249527,id_idx)	/查找编号为10249527的用户

课堂练习p1.11-哈希索引（外存）

练习：打开p1.11.dfx,用排序索引查找1000个随机的订单编号，记录执行时间

	执行时间（毫秒）
排序索引	
哈希索引	

练习：改写p1.11.dfx，使用排序索引查找，比较性能

提示：使用icursor()、index(indexname:h)，h是索引长度。

思考：索引名称可以重复吗？索引长度h设置为多少比较合适？

第一章 查找

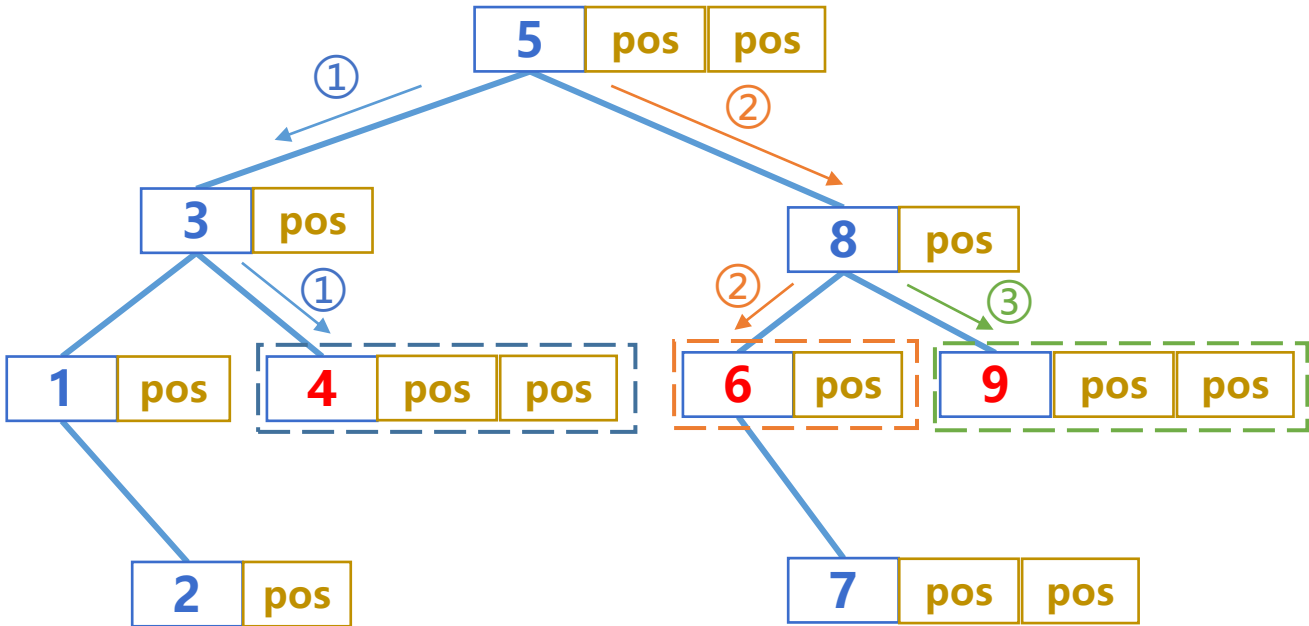
1.2 多键值查找

知识点-键值排序

多键值查找：每次查找多个键值

查找score积分是6、4或者9的用户

说明：用户共9人，编号从1到9



排序索引二叉树示意图

多键值排序

用户键值6、4、9，先把键值排序为[4,6,9]再查找，可以少走回头路。

键值排序的查找过程

- ① 键值4，小与根结点5，转向5左侧；又大于3，转向3右侧，找到4。
 - 下一键值不确定大于5，所以继续和5比。
 - ② 键值6，大于5，转向右侧；又小于8，转向左侧，找到6。
 - 下一键值确定大于5，不确定大于8，所以直接和8比，少比较一次5。
 - ③ 键值9，大于8，转向右侧，找到9。
- 数据量大时，二叉树会有20-30层，每个键值少找10几次，性能提升明显。

代码示例

	A
1	=T.icursor(,[4,6,9].contain(score),s_idx)

说明：icursor函数会对[4,6,9]自动排序

知识点-用索引查找行存文件

索引文件小

只需一个数值即可记录整行的位置。
索引文件和原文件数据条数相同，字段要少很多。

索引查找更快

根据索引找到原文件位置后，只需读取一次原文件即可找到所有字段值。
原文件有压缩的时候，解压缩的量也最小。

索引文件		原文件			
10249001	pos	id	name	gender	...
10249002	pos	...			
...	...	10249001	张三	男	
10249527	pos	...			
...	...	10249527	李四	女	
10250000	pos			
		10250000	王五	女	
		...			

知识点-用索引查找列存文件

索引文件较小

列存文件的索引并不记录每个列的位置，否则索引和原文件一样大，失去索引的意义。索引只记录内部序号，即可快速找到所有列。

索引查找较慢

按照索引找到原文中的位置后，需要读取n个字段，就要读取n次原文件。
原文件解压缩次数也是n。



代码示例-用索引查找行存、列存文件

行存与列存在使用过程中仅创建时的代码需要区分，其余代码均一致

代码示例

行存组表文件创建用参数@r:

	A
1	=file("row.ctx").create@r(#id,data)

列存组表文件创建无需参数:

	A
1	=file("col.ctx").create(#id,data)

行、列存组表文件用索引进行批量数据查找:

	A
1	=file.open().icursor(;A.contain(id),id_idx).fetch()

说明：icursor中支持的过滤条件包括==, >, <>=, <=, contain(), like()。其中，对A.contain()的序列A会自动排序。

课堂练习p2.1~p2.2-行存和列存查找

练习：打开p2.1.dfx，使用列存组表文件col.ctx，
生成行存组表文件row.ctx，并对其id列建索引

	执行时间（毫秒）
列式存储	
行式存储	

练习：打开p2.2.dfx，对行式存储和列式存储的组表文件，分别按同样的1000个键值，
取出相同的记录，记录执行时间

知识点-用带值索引查找数据文件

索引文件稍大

带值索引除了位置之外，还行式存储了需要的少量列值。文件会比普通索引文件要大。

索引查找快

预先存储部分列值后，带值索引可以直接查出这些列值，不需要再访问原文件，所以查找速度很快。

带值索引文件

10249001	pos	张三	男
10249002
...
10249527	pos	李四	女
...
10250000	pos	王五	女

原文件（列存、行存都适用）

id	name	gender	...
...		...	
10249001	张三	男	
...			
10249527	李四	女	
...			
10250000	王五	女	
...			

代码示例-用带值索引查找列存文件

带值索引的创建和使用与不带值索引略有不同

代码示例

创建带值索引

	A
1	=file.open().index(id_F_data_idx;id;data)

创建不带值索引

	A
1	=file.open().index(id_idx;id)

使用带值索引查找

	A
1	=file.open().icursor(;A.contain(id), id_F_data_idx).fetch()

使用不带值索引查找

	A
1	=file.open().icursor(;A.contain(id),id_idx).fetch()

说明：多值、多字段情况的写法示例：index(id_idx;id1,id2,...,idN;data1,data2,...,dataN)。

课堂练习p2.3~p2.4-带值索引

	执行时间 (毫秒)
非带值索引	
带值索引	

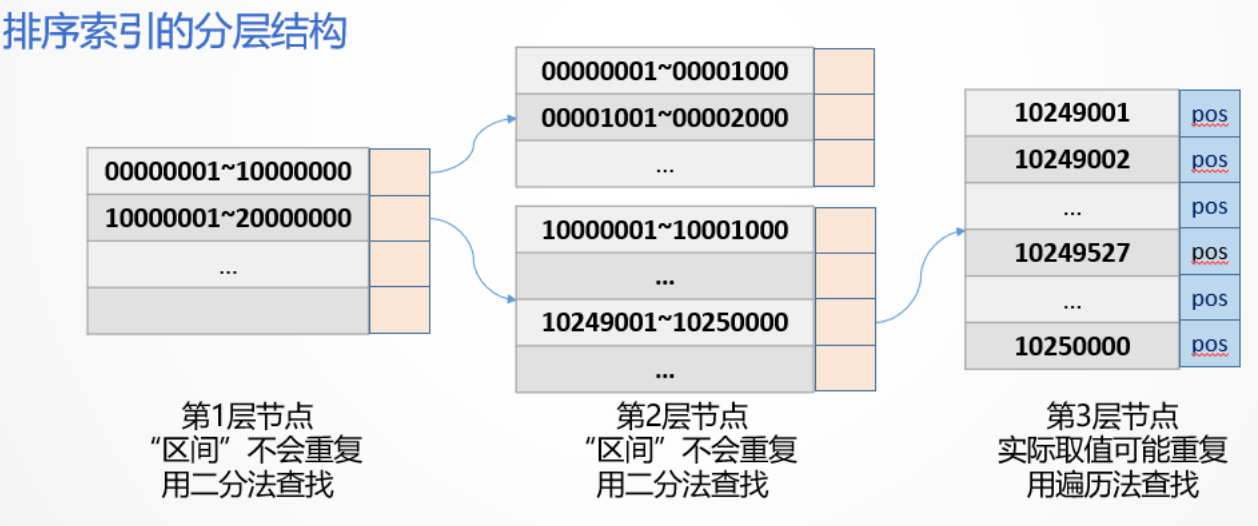
练习：打开p2.3.dfx，对列存组表文件col.ctx，对其id列，创建data的带值索引

练习：打开p2.4.dfx，使用带值与非带值索引，对列存组表文件col.ctx，分别按同样的1000个键值，取出相同的记录，记录执行时间

思考：列存带值索引、行存索引、列存索引的特点是什么？分别适用于哪些场景？

知识点-索引缓存

在内存中预先缓存索引，加速查找



代码示例

	A	B
1	<code>=file("id_data.ctx").open().index@3(id_idx)</code>	/加载三级索引缓存

说明：@3级加载是指比@2加载的层数更多，并不是物理上加载3层索引。实际应用中需要加载@2还是@3，需要根据内存大小，尝试决定。

分层索引

如图索引第3层是原文件的目录，数据规模和原文件相同。数据量大时一般内存装不下，无法缓存。

第2层是键值的区间，数据规模比第3层小很多，第1层又小很多。内存就可以装下了。

分层索引缓存

索引缓存是指：根据内存大小，尽可能预先在内存加载更多层的索引。

对于大数据量、查找大量键值、大并发查找的场景，性能提升明显。

课堂练习p2.5~p2.6-索引缓存

练习：打开p2.5.dfx，多次执行本dfx，记录前5次的时间，找出其中的规律，思考这个现象的原因

第几次	执行时间（毫秒）
1	
2	
3	
4	
5	

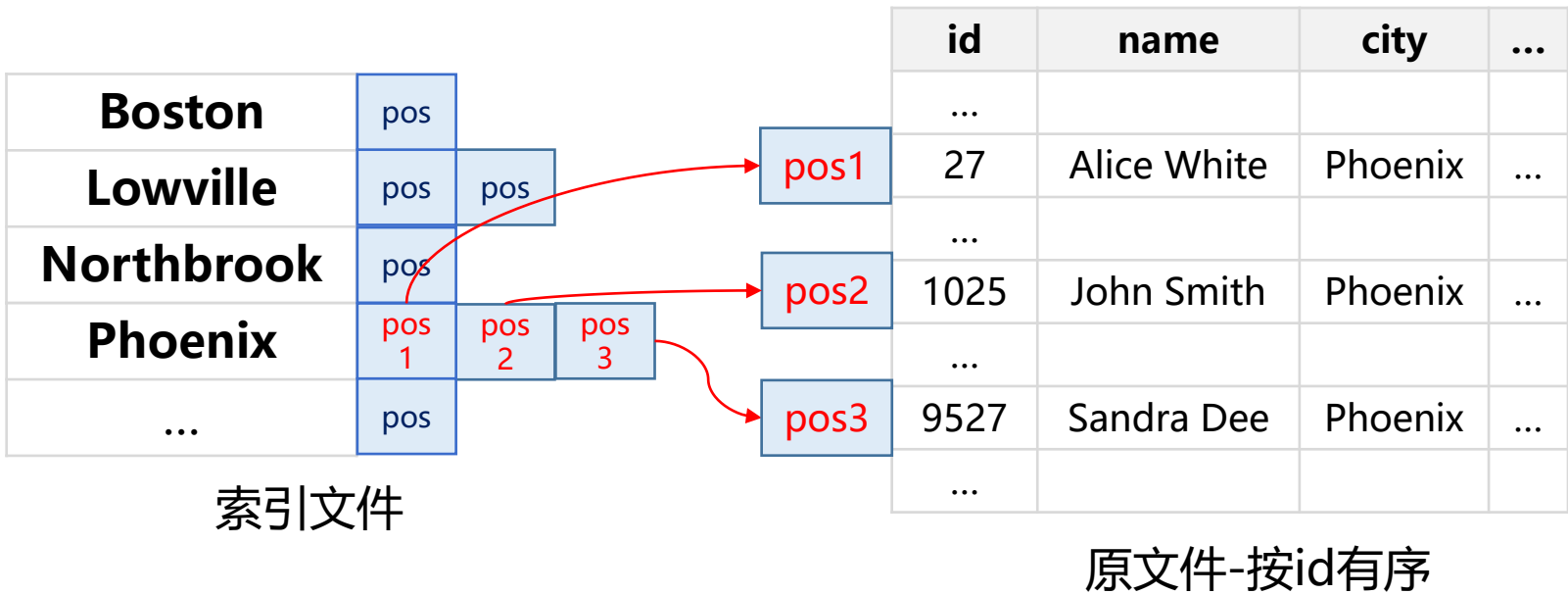
练习：打开p2.6.dfx，执行脚本，观察加载索引缓存时间、执行时间和不加载索引缓存的执行时间

第一章 查找

1.3 结果集查找

知识点-索引返回多条记录

结果集查找：查找结果是集合
查找所属城市为Phoenix的用户



索引返回多条记录

排序索引支持返回多条记录，这些记录在索引中会按原文件中的物理位置排序。
因此，返回的结果集按照id有序。

代码示例

建索引		A	查找		A
	1	=user_file.open()		1	=user_file.open()
	2	=A1.index(city_idx;city)		2	=A1.icursor(;city=="Phoenix",city_idx)

课堂练习p3.1-索引返回多条记录

	执行时间（毫秒）
遍历	
排序索引	

练习：打开p3.1.dfx，用遍历法找orderdate为1982-03-21的订单，记录执行时间

练习：改写p3.1.dfx，使用排序索引查找找orderdate为1982-03-21的订单，比较性能。

提示：使用icursor()、index(indexname)、date()函数。

知识点-物理有序存储

找出客户编号为1001的订单

oid	custid	orderdate	...
10000001048	1002	2018-03-07	
10000001237	1001	2018-03-07	
...			
10000800052	1001	2018-05-13	
11000000053	1004	2018-09-20	
...			
80000000054	1001	2019-06-17	
...	

客户号排序

custid	orderdate	...
...	...	
1001	2018-03-07	
1001	2018-05-13	
1001	2019-06-02	
1002	2018-03-07	
...	...	

物理有序无需索引

订单文件的客户编号物理有序时可以不用索引，利用有序特征，也可以快速得到结果集。

代码示例

排序

	A
1	=orders_file.open().cursor().sortx(custid)
2	=orders_file_sort.create(#custid,...).appden(A1)

查找

	A
1	=orders_file_sort.open().cursor(;custid==1001)

知识点-物理有序存储

按客户或者按日期查找订单

用空间换时间



要按照客户、日期、城市等多种方式查找订单。可将订单文件按照不同的字段排序分别存储成多个文件。

文件占用硬盘变大，但是查找速度变快。

硬盘相对成本较低，占用更多的硬盘来换取更快的查询速度，是用空间换时间的方法。

代码示例

排序	A	
	1	=orders_file.open().cursor().sortx(orderdate)
	2	=orders_file_sort.create(#orderdate,...).appden(A1)

查找	A	
	1	=orders_file_sort.open().cursor(;orderdate=="2018-03-01")

课堂练习p3.2-物理有序

练习：打开p3.2.dfx，一个订单文件对oid有序，第二个订单文件对custid、 oid有序，查找custid为312的订单记录执行时间

练习：改写p3.2.dfx，生成第三个订单文件对orderdate、 oid有序，找orderdate为1982-03-21的订单，比较性能

补充阅读 如前所述，关系数据库建立在无序集合理论上，理论上不能实现同一个表，多种有序方式的冗余存储。

查找custid

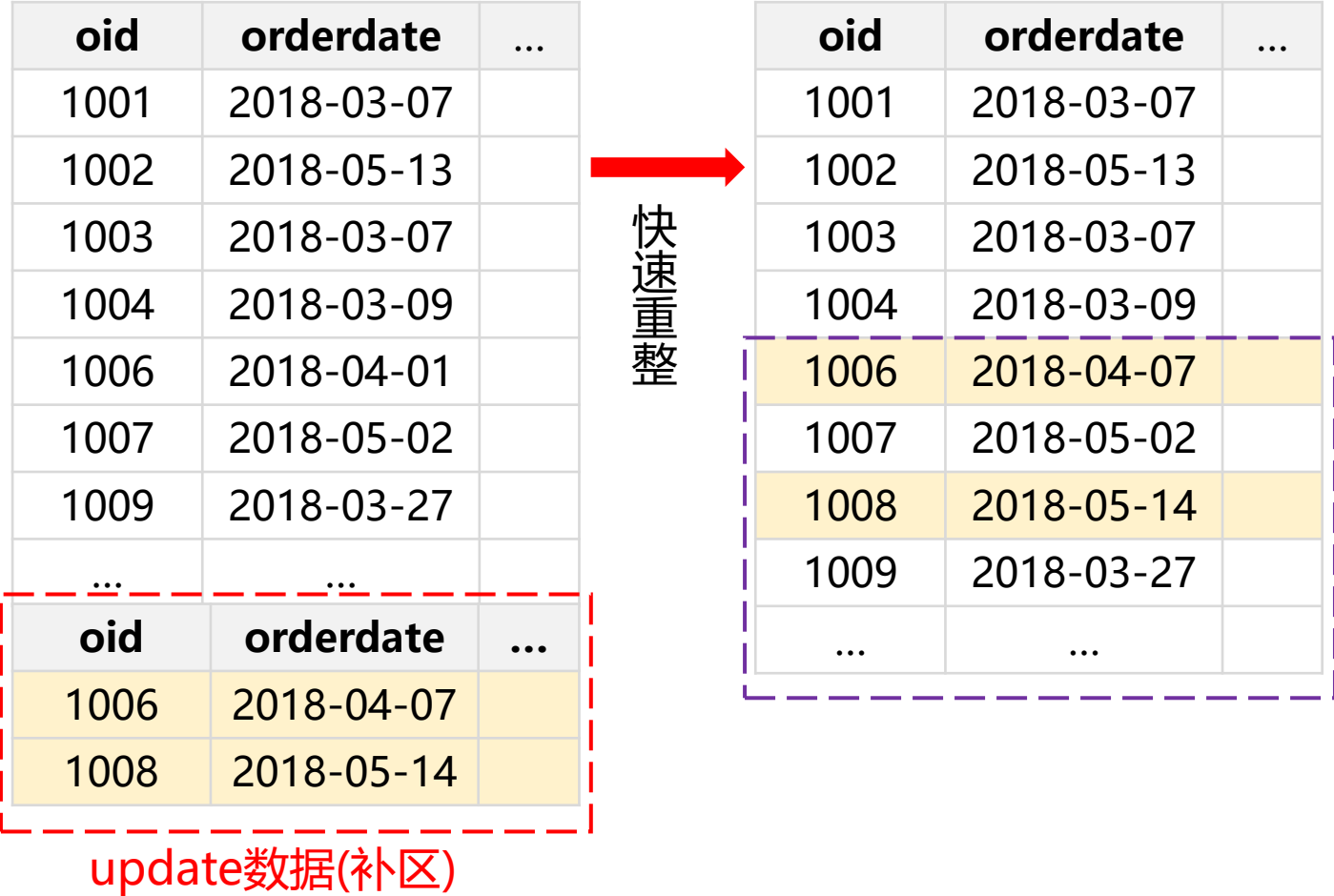
	执行时间（毫秒）
对oid有序	
对custid有序	

查找orderdate

	执行时间（毫秒）
对oid有序	
对orderdate有序	

知识点-数据更新

修改后文件数据的更新



补区

更新数据放入文件的补区。按客户查找订单的结果集，要逐条检查补区的记录，修正结果集。

补区记录增多，查询速度会受到较大影响。

快速重整

将补区的数据更新到原数据中，重整后文件整体有序。

仅重整第一次出现补区数据后的部分，之前的数据不用重写。

代码示例

	A	B
1	=add_file.open().update(update_data)	/修改数据
2	=add_file.reset@q()	/快速重整

课堂练习p3.3-数据更新

练习：打开p3.3.dfx，组表文件在更新之前和之后，用同样的条件查找oid为9000312的订单，记录前后查找速度的差异

查找oid

	执行时间（毫秒）
更新之前	
更新之后	
快速重整后	

练习：改写p3.3.dfx，对组表快速重整，再查找oid为9000312的订单，记录执行时间。

思考：快速重整的时间仍然比较长，如何解决？

知识点-数据更新

文件，日期有序

orderdate	userid	...
2018-03-07	1001	
2018-03-07	1002	
...	...	
2018-05-13	1001	

文件，客户号有序

custid	orderdate	...
1001	2018-03-07	
1001	2018-05-13	
1002	2018-03-07	
...	...	

新增数据

orderdate	custid	...
2018-05-14	1001	
2018-05-14	1003	

文件，日期有序

orderdate	userid	...
2018-03-07	1001	
2018-03-07	1002	
...	...	
2018-05-13	1001	
2018-05-14	1001	
2018-05-14	1003	

文件，客户号有序



数据的每日更新

应用系统产生的新数据需要定时添加到数据文件中。一般是每天添加一次。

日期有序数据的更新

按照日期有序的数据，将每天产生的新数据追加到文件的最后即可。

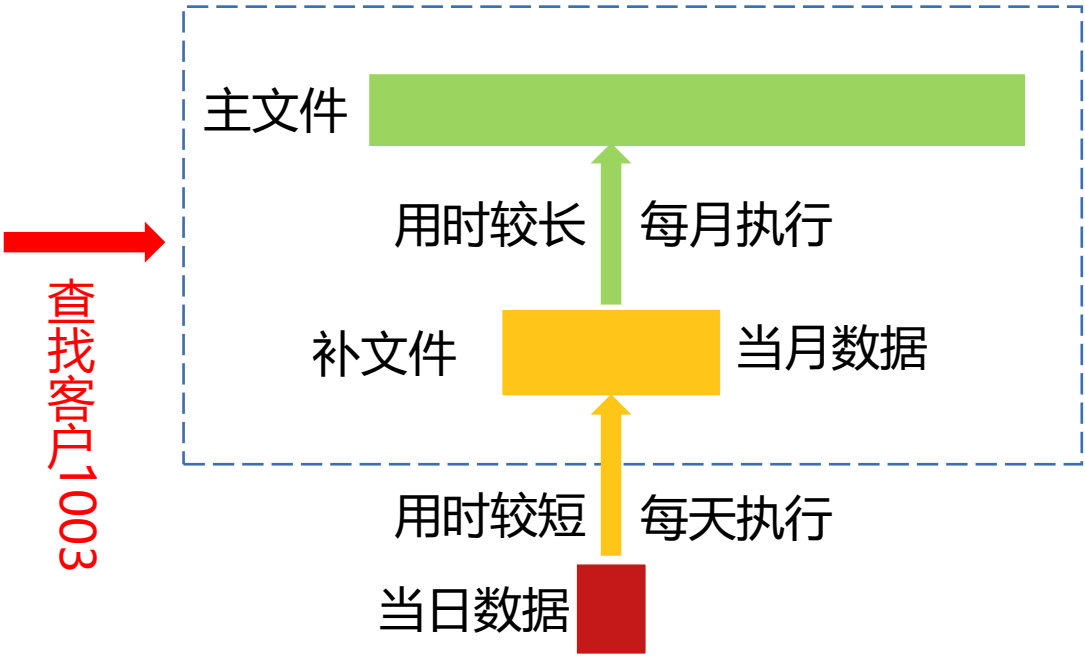
非日期有序数据的更新

按照客户有序的数据，如果直接追加数据，就不是整体有序了。

每天都重新排序的话，占用的时间太长。需要有合理的办法解决。

知识点-数据更新

日增数据的更新



补文件

订单主文件比较大，如果每天重整，耗时太长。

另建一个补文件，也对客户有序。每天新数据归并到补文件，耗时较短。每月只需对主文件重整一次。

包含补文件的查找

查找客户的订单，从主文件和补文件归并查找。

两文件都对客户有序，查找时比一个文件略慢。

代码示例-数据更新

日增数据的更新

代码示例

增量更新

	A	B	C
1	if day==1	=file("main_file").reset()	/月初重整
2	=file("main_file").open().append@a(add_data)		/每日追加补文件

文件合并查询

	A	B
1	=file("main_file.ctx")	/打开主文件
2	=A1.open().cursor(;custid==1003)	/查询自动归并补文件

每日追加

每天定时将当日新增数据有序归并到补文件中。

补文件和当日数据都不是特别多，所以归并时间较短。

每月重整

月初将文件重整，将补文件有序归并到主文件，同时清空补文件。

每月执行一次，时间稍长也可以接受。

索引自动更新

归并和重整自动更新数据文件对应的索引文件。

课堂练习p3.4-数据更新

练习：打开p3.4.dfx，包含补文件查找custid为312的订单，记录执行时间，与之前单文件查找的结果比较

练习：改写p3.4.dfx，重整文件，再对单文件查找custid为312的订单，观察主文件和补文件的变化。

查找custid

	执行时间（毫秒）
补文件	
组表重整后	

第一章 查找

1.4 多条件查找

知识点-区间查找

查询日期大于等于2018-03-07 小于等于2018-05-13的订单

...	pos			oid	custid	orderdate	...
2018-03-06	pos			...			
2018-03-07	pos	pos		10000001048	1002	2018-03-07	
...	pos			10000001237	1001	2018-03-15	
2018-03-15	pos 1	pos 2	pos 3	...			
2018-03-16	pos			10000002052	1001	2018-03-16	
...	pos			10000002053	1004	2018-03-15	
				...			
				80000000054	1001	2019-06-17	
				

索引文件

订单文件

物理有序

原文件按oid有序，用oid区间查找时无需索引。

物理无序

原文件对orderdate无序，需要预先建立排序索引查找。

哈希索引

哈希函数不单调，一个索引值可对应多个orderdate，适合精确查找，不适合区间查找。

代码示例-区间查找

物理有序，直接查询

代码示例 查找oid小于201000的订单

	A	B
1	=orders_file.open().cursor(;oid<201000)	/oid有序，不用索引直接查找

物理无序，索引|查询

代码示例 查找orderdate小于2018-05-13的订单

建索引		A	查找		A
	1	=orders_file.open()		1	=orders_file.open()
	2	=A1.index(orderdate_idx;orderdate)		2	=A1.icursor(;orderdate<date("2018-05-13"),orderdate_idx)

课堂练习p4.1-区间查找

练习：打开p4.1.dfx，对oid有序的组表查找oid区间，记录执行时间。

同样的组文件查找orderdate区间，记录执行时间

	执行时间（毫秒）
查找oid	
查找orderdate	
排序索引 查找orderdate	

练习：改写p4.1.dfx，使用排序索引查找orderdate区间的订单，比较性能

提示：使用icursor()、index(indexname)、date()函数。

oid区间：
oid>100000 && oid<300000

orderdate区间：
orderdate>=date("1980-01-10") && orderdate<=date("1980-01-20")

知识点-多字段联合索引

生日和城市两个条件查找用户

生日小于1985年1月1日且城市为Beijing

birthday	city			
.....	pos		
1984-12-29	Beijing	pos		
1984-12-29	Shenzhen	pos		
1984-12-30	Shanghai	pos	pos	
1984-12-31	Beijing	pos		
1984-12-31	Beijing	pos	pos	pos
1984-12-31	Shenzhen	pos		
1985-01-01	Beijing	pos		
.....	pos		

索引文件

联合索引

生日和城市联合索引中，生日先排序。相同的生日对应多个城市的，城市在同一个生日片内排序。

联合索引不适合的情况

生日和城市联合索引，可以对生日、城市两个条件查找，也可以对生日单独查找。
因为城市只是在一个生日片区内有序，不是整体有序的，所以不适合对城市单独查找。

代码示例-多字段联合索引

代码示例

建立生日和城市的联合索引

	A	B
1	=users_file.open().index(index-b-c;birthday,city)	/生日和城市的联合索引

使用联合索引查询两个字段的条件

	A	B
1	=users_file.open().icursor(;birthday<date("1985-01-01") && city=="beijing"))	/每个生日中，城市有序，适合使用联合索引

使用该联合索引查找城市为beijing的用户

	A	B
1	=users_file.open().icursor(;city=="beijing")	/索引对city不是整体有序，不适合使用联合索引

课堂练习p4.2-多字段联合索引

练习：打开p4.2.dfx，对oid有序的组表查找orderdate、custid两个字段条件的订单，记录执行时间

	执行时间（毫秒）
无索引查找	
联合索引 查找两个条件	
联合索引 查找custid	

练习：改写p4.2.dfx，使用orderdate、custid联合索引查找两个字段条件的订单；再查找custid一个字段条件的订单，比较性能

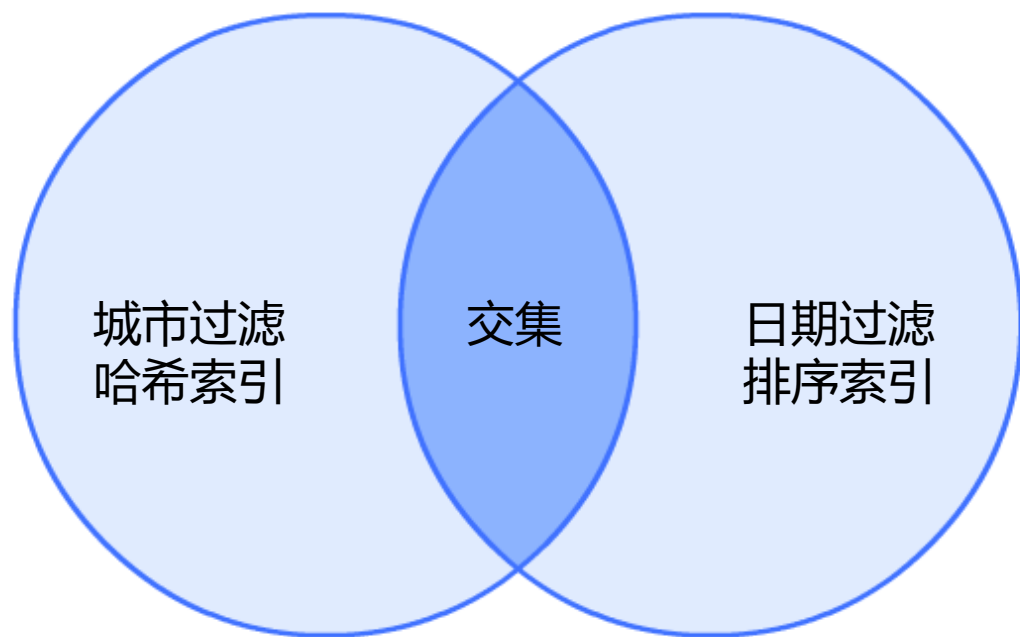
提示：使用icursor()、index(indexname)、date()函数。

```
custid条件:  
custid>160 && custid<170  
orderdate条件:  
orderdate>=date("1980-01-10") && orderdate<=date("1980-01-12")
```

知识点-多字段分别索引

生日和城市两个条件查找用户

生日小于1985年1月1日且城市为Beijing



分别索引

生日和城市分别建两个独立的索引。

生日是区间查找所以建立排序索引；

城市是精确查找所以建立哈希索引。

查找结果求交集

生日和城市分别用各自的索引查找结果，然后将两个结果集求交集。

代码示例-多字段分别索引

代码示例

建立生日和城市的联合索引

	A	B
1	=users_file.open()	/打开文件
2	=A1.index(city_idx:1;city)	/对城市建立哈希索引
3	=A1.index(bday_idx;birthday)	/对生日建立排序索引

城市为Beijing和生日小于1985年1月1日的用户做交集

	A	B
1	=users_file.open()	/打开文件
2	=A1.icursor(;city=="beijing" && birthday<date("1985-01-01"))	/查找城市为Beijing，生日小于1985年1月1日的用户，两者做交集

课堂练习p4.3-多字段分别索引

练习：打开p4.3.dfx，对oid有序的组表查找orderdate、custid两个字段条件的订单，记录执行时间

	执行时间（毫秒）
无索引查找	
分别索引 查找两个条件	

练习：改写p4.3.dfx，使用orderdate、custid分别建立索引查找两个字段条件的订单

提示：使用icursor()、index(indexname)、date()函数。

custid条件：

Custid==160

orderdate条件：

orderdate>=date("1980-01-10") && orderdate<=date("1980-01-12")

补充阅读 关系数据库的理论支持一定的集合化，但是不够彻底，不支持分别索引，结果求交集。
只能采取联合索引的方式。

知识点-多条件次序

找出性别为女性且出生日期小于1990年01月01日的用户

女性用户较少，但生日普遍小于1990年

id	...	sex	birthday
.....		
100048		F	1984-01-11	
100049		M	1974-03-26	
100050		F	1989-12-15	
100051		M	1997-06-07	
100052		M	1983-09-13	
100053		M	1984-11-21	
100054		F	1988-08-02	
.....		

多条件次序

多个查询条件是从左到右顺序执行。

将结果集小的条件放在前面，后边的条件过滤的执行时间会减少。

性别为女性的查询结果比生日小于1990年1月1日的总数少很多，所以放在前面（左面），性能更好。

代码示例

	A	B
1	=users.select(sex=="F" && birthday<date(1990-01-01))	/先过滤性别再生日
2	=users.select(birthday<date(1990-01-01) && sex=="F")	/先过滤生日再性别

课堂练习p4.4-多条件次序

练习：打开p4.4.dfx，对oid有序的组表查找
orderdate、custid两个字段条件的订单，
记录执行时间

	执行时间（毫秒）
修改顺序前	
修改顺序后	

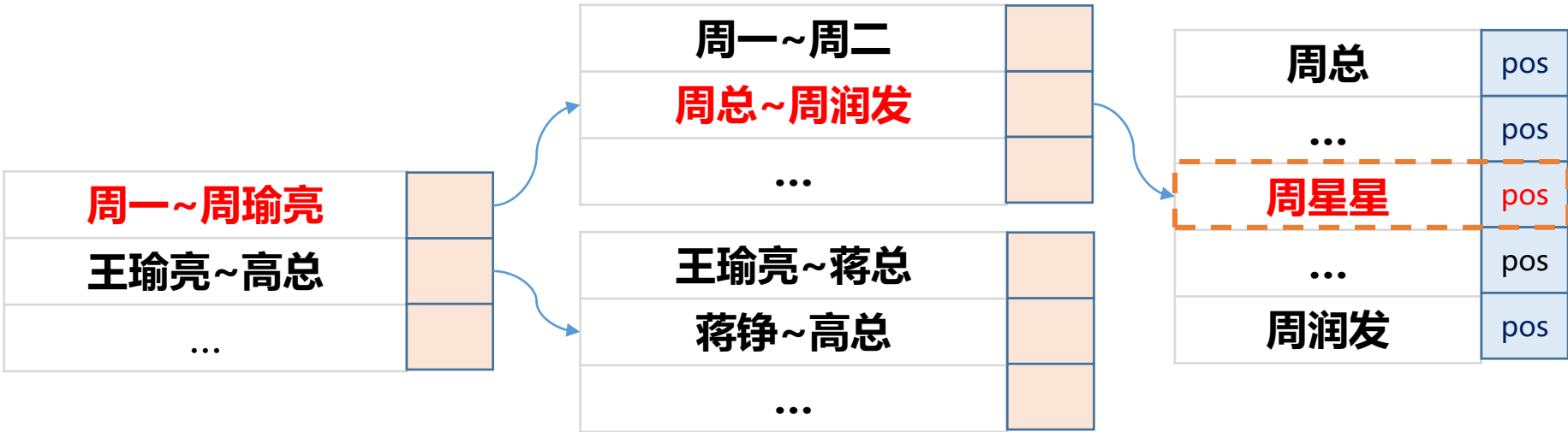
练习：改写p4.4.dfx，修改条件的顺序，记录执行时间

```
custid条件:  
Custid==160  
orderdate条件:  
orderdate>=date("1980-01-10") && orderdate<=date("1980-01-12")
```


知识点-全文检索-排序索引

like("X*")型, 可用排序索引

like("周星*")



代码示例

	A	B
1	=users_file.open().index(name_idx;name)	/对姓名建立排序索引
2	=users_file.open().icursor(like(name,"周星*"), name_idx).fetch()	/使用排序索引

课堂练习p4.5-全文检索-排序索引

练习：打开p4.5.dfx，对包含html字段content的组表查找以"r"开头的行，记录执行时间

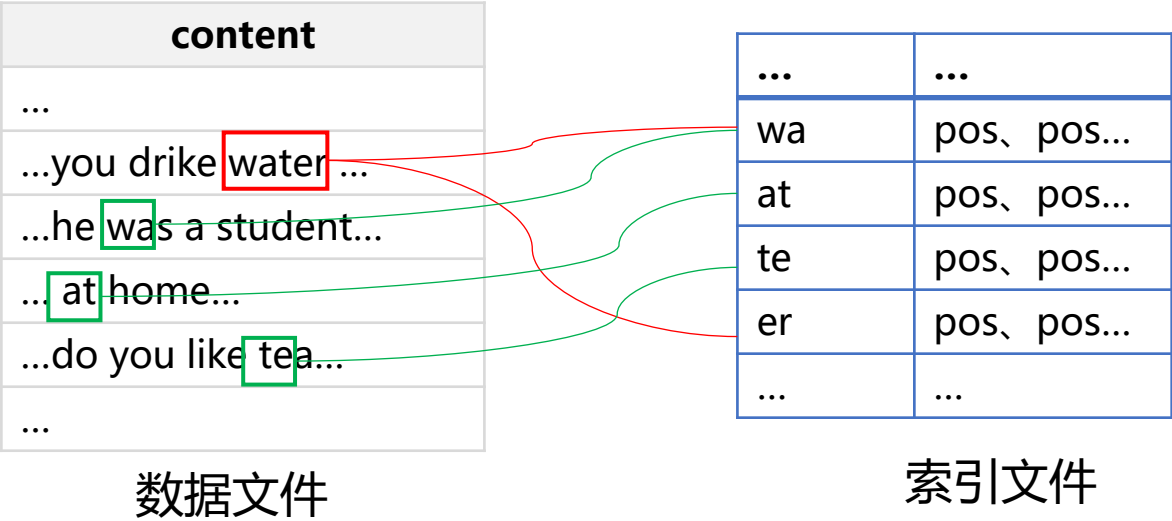
	执行时间（毫秒）
遍历查找	
排序索引查找	

练习：改写p4.5.dfx，用排序索引完成查找，记录执行时间

提示：使用icursor()、index(indexname)

知识点-全文检索-全文索引

查找content like("*water*"), 要用全文索引



代码示例

	A	B
1	=data_file.open().index@w(content_idx_fulltext;content)	/对content建立全文索引

用任意字符组合建索引

如果用任意字符组合建索引，组合数无穷多，索引太大，无法实现。

用连续n个字符建立索引

对单词water建全文索引，用连续的2个字符为单位，记下位置pos。

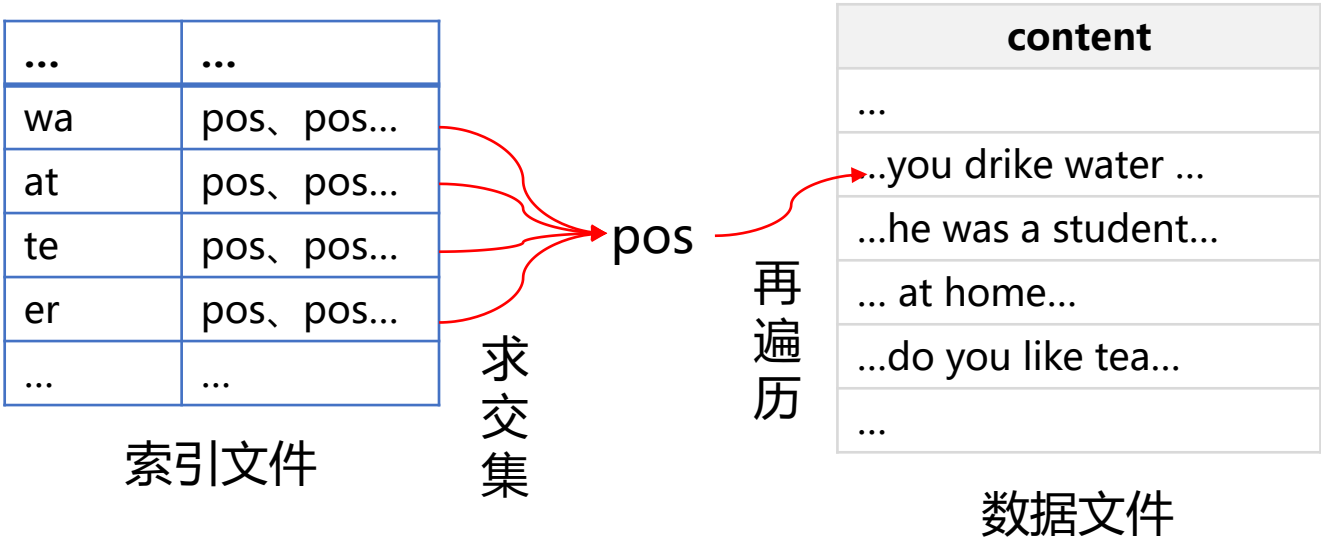
后续再出现同样的组合，例如wa、at等，也要记下位置pos。出现不同的组合，就用新的组合记下位置。

其他单词以此类推。

连续的字符个数为n。n一般不超过4即可。

知识点-全文检索-全文索引

查找content like("*water*"), 要用全文索引



代码示例

	A	B
1	=data_file.open().icursor(like(content,"*water*"),content_idx_fulltext).fetch()	/使用全文索引查询

使用全文索引查询

要查询关键词water, 先把关键词拆分成连续的两个字母, 共4个组合。分别在索引中找到组合对应的位置, 求交集。

对交集集中的多个位置, 再遍历content字段, 找到包含water的行。

全文索引特点

关键词应该是中文或者英文的词, 因此必须是汉字或者英文字母组成。

关键词小于连续的字符个数n时, 也可以使用全文建索, 但是返回的位置集合多, 速度会变慢。

课堂练习p4.6-全文检索-全文索引

练习：打开p4.6.dfx，对包含html字段content的组表查找包含"cursor"单词的行，记录执行时间

	执行时间（毫秒）
遍历查找	
全文索引查找	

练习：改写p4.6.dfx，用全文索引完成查找，记录执行时间

提示：使用icursor()、index@w(indexname)

第二章 遍历

2.1 存储方案

2.2 常规遍历

2.3 分组排序

2.4 高级遍历

课前准备1-硬件和主目录

硬件 内存：8G以上 硬盘：空余10G以上 CPU：主频1G以上

主目录 在硬盘适当位置解压缩附件 “traversal.zip” ， 并将集算器主
目录设置成 “traversal” 文件夹
设置方法：集算器-菜单-工具-选项-主目录

课前准备2-生成数据文件

要求：上课前执行并读懂生成数据文件的脚本，回答三个问题：

A，数据文件数据量是多少？有哪些字段？

B，有哪些字段，生成后占用硬盘有多大？

C，有什么特征：有序、列存、行存、分段等等。

1、执行脚本“主目录\dfx\orders.dfx”，生成订单组表文件“主目录\data\ctx\orders.ctx”和文本文件“主目录\data\txt\orders.txt”。

2、执行脚本“主目录\dfx\cols.dfx”，生成订单组表文件“主目录\data\ctx\cols.ctx”和集文件“data/btx/cols.btx”。

3、执行脚本“主目录\dfx\log.dfx”，生成日志文件“主目录\data\txt\log.txt”。

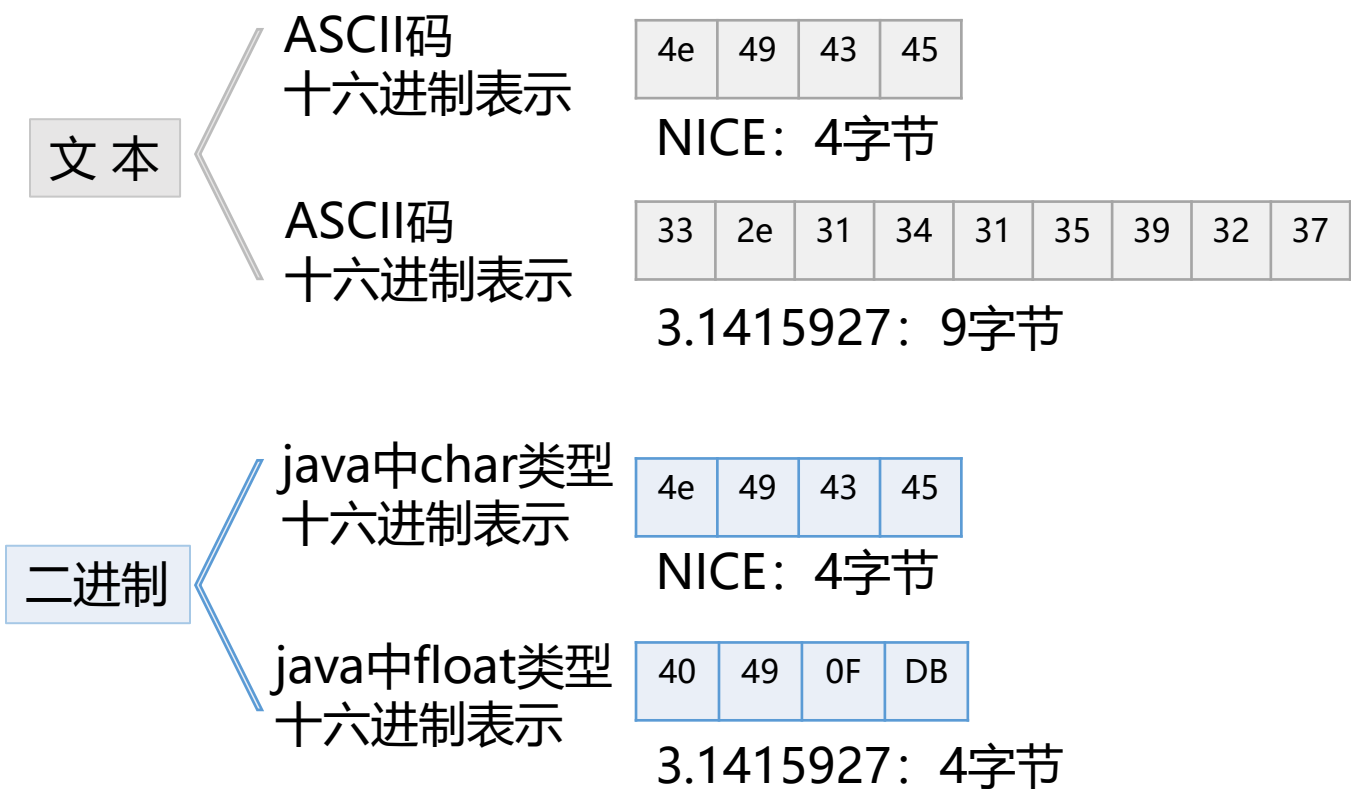
第二章 遍历

2.1 存储方案

知识点-存储格式

二进制比文本存储更节省空间

存储字符 “NICE” 和数值 “3.1415927”



二进制值前还要加 “头”，存储值类型和可变长度

数据库表的遍历

数据库表是二进制存储，库内遍历快，但外部取出数据遍历慢。

文本文件的遍历

编码通用，但占空间较大。需逐个解析字段值的数据类型，性能差。

二进制文件的遍历

占用空间小，无需数据类型解析，性能最好。

代码示例-存储格式

代码示例

基于数据库存储

	A
1	=now()
2	=connect("oracle")
3	=A2.cursor("select * from customer")
4	for A3,10000
5	=A2.close()
6	=interval@s(A1,now())

基于二进制存储

	A
1	=now()
2	=file("ctx/customer.ctx").open()
3	=A2.cursor()
4	for A3,10000
5	=A3.close()
6	=interval@s(A1,now())

基于文本存储

	A
1	=now()
2	=file("txt/customer.txt")
3	=A2.cursor(, " ")
4	for A3,10000
5	=A3.close()
6	=interval@s(A1,now())

3千万数据遍历实测比较

性能排名	存储格式	时间(秒)
1	ctx(二进制)	37
2	txt	42
3	Oracle	293

课堂练习p1.1-存储格式

练习：打开p1.1.dfx，记录文本文件的遍历时间

	执行时间（毫秒）
文本	
二进制	

练习：修改p1.1.dfx，改为遍历二进制文件，比较时间

提示：二进制文件 “主目录/data/ctx/orders.ctx” 。

思考：观察文本文件和二进制文件的大小有何差异，为什么？

知识点-并行和分段

多线程并行，需要文件分段



分段要求1：每段数据量基本相同

并行任务的最终用时，是最慢的线程所用时间。同一机器中各线程的处理能力基本相当，因此数据分段要能做到尽量平均，使各线程的计算时间基本相同。

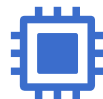
知识点-并行和分段

并行数需要动态调整



并行数不能超过文件最大分段数。

如果文件最多只能分四段，那么即使有更多的CPU核，也只能四并行计算。



并行数如果超过CPU的核数，就不是真正的并行了。

生产环境中CPU的核数是多种多样的，数据准备阶段很难事先预知CPU核数。



计算机硬盘的IO能力是有限的。

如果硬盘读取线程过多，可能会导致磁盘频繁跳跃寻址，从而降低性能。

分段要求2：分段数可灵活动态指定

实际计算用的线程数最好是根据当时场景动态决定，范围从几个到几十个都有可能，这要求能够按任意的数量将数据分段。

知识点-并行和分段



分段要求3：每个分段连续紧凑存储

硬盘不适合频繁随机访问（即使固态硬盘也不适合频繁小量的随机访问），为了保证遍历性能，每个线程要处理的数据在硬盘上要尽量连续存储，而不是频繁跳跃。因此，分段内最好没有空白空间。



分段要求4：允许数据追加

数据并不是固定不变的，要随着时间不断增长。分段文件每次追加数据时，要做到不必重新整理所有数据，只需要把追加的数据补上即可保持分段。

知识点-文本分段

文本文件满足分段的四个要求

每段条数	起始字节	结束字节
14	0	x
15	x+1	...
...		
13

分n段
假设每段约15条

	sid	amount
1		
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
23	6	5566

sid	amount
...	...
2	2304
...	...

x字节位置

数据文件sales.txt

文本文件分段方法

用数据文件总字节数除以分段数n，得到分段的平均字节数x。

线程1从0字节读取到x字节，假设是数值“2304”的第1位。继续读取到换行符，结束第1段。

线程2从x+1开始读取到换行符后，以其下一个字符为第2段开始。以此类推，去头补尾，保证记录完整，满足分段的四个要求：

- 1.每段字节数=总字节数÷分段数，保证每段数据量基本相同
- 2.分段数只要小于总行数，即可灵活动态指定
- 3.每个分段没有空白空间，连续紧凑存储
- 4.文本文件允许数据追加

代码示例-文本分段

代码示例

分8段，读取第2段

	A
1	=file_sales.cursor@t(;2:8).fetch()

4线程并行遍历

	A	B
1	fork to(4)	=file_sales.cursor@t(;A1:4)
2		for B1,10000

课堂练习p1.2-文本分段

练习：打开p1.2.dfx，记录文本文件的遍历时间

	执行时间（毫秒）
单线程	
2线程	
4线程	
8线程	
16线程	

练习：修改p1.2.dfx，改为并行遍历文本文件，比较时间

提示：使用fork关键字。

注意检查“菜单-工具-选项-最大并行数”，要大于等于16并行数。

有兴趣可以继续测试更大的线程数。

思考：线程数是不是越大越好？为什么？

知识点-二进制分段

二进制文件字段和记录的存储

文本文件

编号	名字	积分	
1001	Luck	1900	回车符
1002	James	3000	回车符

文本要解析值确定数据类型，性能差。

二进制文件

编号		名字		积分	
头	1001	头	Luck	头	1900
头	1002	头	James	头	3000

头数据包含内容
数据类型：字符串
长度：5

头数据包含内容
数据类型：整数

二进制无记录分隔符

二进制没有“回车”这种记录分隔符，不能用分隔符的方法判断记录结束。

每个字段值都有“头”

“头”包含“数据类型”，和可变长的值，还有“长度”。省掉解析数据类型的时间，可以提高性能

“头”数据也是二进制，要尽可能简短，少占用空间。

读取字段和记录的方法

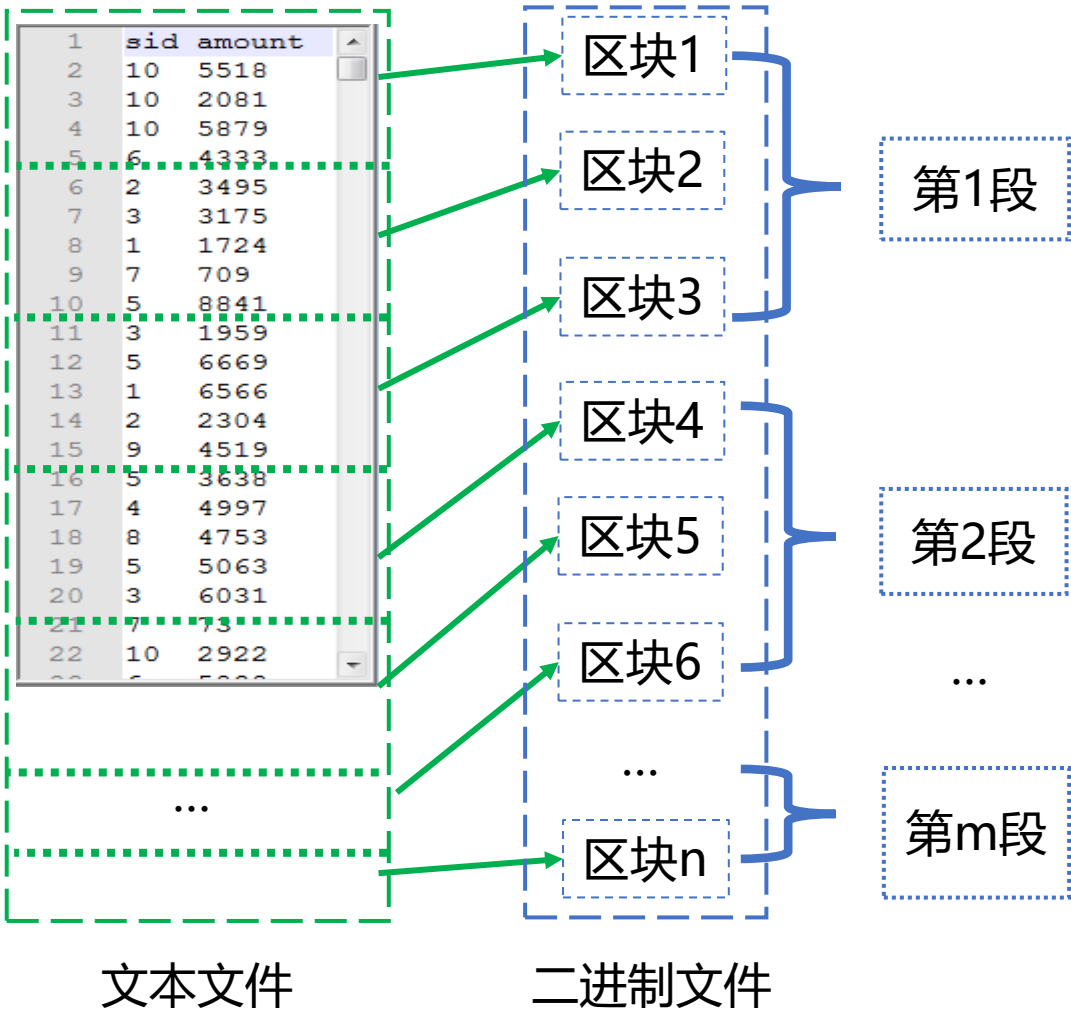
二进制文件开始的部分是文件头，包含字段名称等必要的信息。

读取一条记录的时候，先读“头”确定字段类型和长度，然后按照长度读取值，取完最后一个字段，本条记录结束。

所以，读取时要知道记录起始位置。

知识点-区块分段

二进制文件的区块分段方案



区块分段方案

把数据预先存入多个大小相同的区块，分段时以区块为单位。只要区块足够小，就满足分段的四个要求。

字段数量非常多，需要列存的时候，采取区块内列存方式。

区块分段存在问题

分块数要足够多才能保证平均分块，而分块又要足够大才能让列存产生效果，这两者就是个矛盾，要数据量很大时才合适，有一定的局限性。

知识点-倍增分段

文件头预留分段索引区

倍增分段需要在二进制文件头部预留固定长度的索引区，为了讲解方便，假设其长度为4。就是说，共有4个号位，用来存储分段开始的字节位置。

下面用pos1、pos2..代表第1、2..条记录开头的字节位置。

图中绿色表格表示二进制文件。

步骤一，写入四条数据

	1号位	2号位	3号位	4号位
	pos1	pos2	pos3	pos4
pos1	第1条数据			
pos2	第2条数据			
pos3	第3条数据			
pos4	第4条数据			

写入4条记录，四个号位分别存储4条记录的起始位置。分段索引区已满。

步骤二，写入第五条

	1号位	2号位	3号位	4号位
	pos1	pos3	pos5	
pos1	第1条数据			
pos2	第2条数据			
pos3	第3条数据			
pos4	第4条数据			
pos5	第5条数据			

写入第5条记录。索引区1号位不动，2号位改为原3号位内容。3号位写入pos5，4号位清空。

知识点-倍增分段

步骤三，再写入三条

	1号位	2号位	3号位	4号位
	pos1	pos3	pos5	pos7
pos1	第1条数据			
...	...			
pos5	第5条数据			
pos6	第6条数据			
pos7	第7条数据			
pos8	第8条数据			

再写入3条，共8条。索引区1号位存储1、2两条的起始位置，其他号位类似。四个号位已满。

步骤四，写入第九条

	1号位	2号位	3号位	4号位
	pos1	pos5	pos9	
pos1	第1条数据			
...	...			
pos5	第5条数据			
...	...			
pos8	第8条数据			
pos9	第9条数据			

写入第9条记录。索引区1号位不动，2号位改原C号位内容。3号位写入pos9，4号位清空。

步骤五，继续追加写入

继续追加数据的时候以此类推。

分段效果

每个号位相当于一个分块。

初始时每块一条记录。步骤二是第一次倍增，每块增加为两条记录。步骤四是第二次倍增，每块增加为四条记录，以此类推。

每次倍增，块数最大值不变，每块记录数增加一倍。

知识点-倍增分段

预留索引区长度增加为10个



分段效果

分段索引区固定大小，不会随着数据追加而增大。
追加数据无需重构文件，只要修改索引区的值即可。

满足分段4个要求

- 1.非空号位记录数相等，每段平均分区块，数据量基本相同。
- 2.区块足够多时，例如1024个，分段数即可灵活指定。
- 3.分段内是连续区块，区块内是连续记录，可连续紧凑存储。
- 4.数据追加时，索引区填空、倍增即可，无需重构文件。

知识点-列式存储

列存：数据列分别存储

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

5000万行存文件取
产品ID、数量、地区
进行遍历，耗时40秒

5000万列存文件取
产品ID、数量、地区
进行遍历，耗时13秒

列存适合较少列的遍历

对于总列数很多，但取出列数较少的场景，列存可以减少硬盘读取数据量，遍历性能优于行存。

列存不适合很多列的遍历

数据是按列连续存放的，需要取出很多列计算时，会发生硬盘随机访问现象，性能未必比行存好。并行会加剧这个问题，特别是机械硬盘。

应用时，要因地制宜决定是否采用列存。

代码示例

	A
1	=col_ctx.open().cursor(productid,quantity,area)
2	for A1,1000000
3	=row_ctx.open().cursor(productid,quantity,area)
4	for A3,1000000

课堂练习p1.3-列存和行存

练习：打开p1.3.dfx，记录列存文件的遍历时间

	执行时间（毫秒）
列存文件	
行存文件	

练习：修改p1.3.dfx，改为使用行存文件，取相同列遍历，比较时间

思考：当取全部列时，两种存储方式，哪种遍历时间更快？

知识点-列存倍增分段

订单ID	1号位	2号位	3号位	4号位	5号位	6号位	7号位	8号位	9号位	10号位
	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

预留索引区
长度为10

用户ID	1号位	2号位	3号位	4号位	5号位	6号位	7号位	8号位	9号位	10号位
	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

产品ID	1号位	2号位	3号位	4号位	5号位	6号位	7号位	8号位	9号位	10号位
	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

...

倍增分段适用于列存

倍增分段方案以记录数为基础，对于列存也适合。不会出现分块大小与数量的矛盾。

列存不会错位

各列都采用倍增分段方式追加数据后，分段点对于各列都落在同一条记录上，不会错位。

补充阅读 业界常用的列存分段是分块方案：把数据分成若干块，块内是列存，分段以块为单位。分块数要足够多才能保证平均分段，而分块又要足够大才能让列存产生效果，这两者就是个矛盾。

知识点-有序压缩

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

行存-按地区排序

行存不易压缩

行存是按行存储的，国家USA和相邻存储的性别M一般不会相同，不能合并压缩，只能依次存储。

解压缩性能

解压缩要将USA整块解压，适合需要全部数据的遍历计算。对于随机查找少量记录的场景，如果也要整块解压反而降低性能，所以不适合列存有序压缩。

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

列存-按地区排序

列存有序合并压缩

行存是按列存储的。按国家排序后，USA连续存储并出现3次，可只存储一个值和次数。

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

列存-按性别排序

排序字段的选择



国家的字符数比性别要长，按照长的字段排序的压缩率更高。值重复越多，压缩效率越高。

代码示例-有序压缩

排序写入的代码示例

	A	B
1	=file("employee.ctx").open().cursor(level,height,weight,city,id,name,sex).sortx(level,height,weight,city)	
2	=file("employee_sort.ctx").create(#level,#height,#weight,#city,id,name,sex)	
3	=A2.append(A1)	=A2.close()

排序后压缩的列存文件容量对比

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

补充阅读

关系数据库的理论基础是无序集合，不能保证数据在物理存储时的有序性，无法利用存储上的次序实现有序压缩。

课堂练习p1.4-有序压缩

练习：打开p1.4.dfx，记录oid有序的列存文件的遍历时间与文件大小

	执行时间（毫秒）	文件大小（MB）
oid有序		
orderdate、custid有序		

练习：修改p1.4.dfx，改为按orderdate、custid有序的列存文件，比较遍历时间与文件大小

思考：按custid、orderdate排序是否合适？为什么文件大小有差异？

知识点-内存压缩

列存压缩可减少内存占用



内存对Java影响很大

Java把数据读入内存对象化后，占用空间会比占用硬盘变大很多。
当内存不足，Java的GC会不断回收内存，严重影响性能。

使用时对象化

读入数据暂保留原样，使用时再对象化，可减少内存占用，但会损失性能。

列存有序压缩

列存有序压缩配合“使用时对象化”的办法可进一步减少内存占用。但会加大对象化的复杂度，需根据实际情况权衡选择。

代码示例

	A
1	=file_ctx.open().cursor().fetch
2	=A1.memory@z(;id<=100)

课堂练习p1.5-内存压缩

练习：打开p1.5.dfx，将orders.ctx的部分数据读入内存。

练习：改写p1.5.dfx，用内存压缩方式读入。

提示：用memory()函数。

思考：memory()函数需要加选项么？

知识点-minmax索引

有过滤条件的遍历

统计合同额在29000-30000之间的订单数

订购日期	...	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:43275 max:47628	min:28456 max:31283	min:25670 max:28365	

订购日期	合同额	订单号	...
2017-03-05	28456	7364875	...
2017-03-05	29137	7364876	...
...
2017-03-05	30294	7517645	...
...

minmax索引

数据分块存储时，可记录每块的最大最小值，可提高条件过滤的遍历性能。

图中画出的是订购日期、合同额的最大最小值。

minmax索引过滤步骤

合同额的过滤区间为29000~30000，则绿色虚线的数据块或者最大值小于29000，或者最小值大于30000，一定不符合过滤条件，所以可以直接跳过。

只有和过滤区间有交集的数据块才需要遍历过滤。

代码示例

	A
1	=file_ctx.open().cursor(;amt>=29000&&amt<=30000))
2	=A1.groups(;count(~))

课堂练习p1.6-minmax索引

练习：打开p1.6.dfx，执行并记录时间

	执行时间（毫秒）
不用索引	
minmax索引	

练习：改写p1.6.dfx，改为使用minmax索引，遍历并计时

第二章 遍历

2.2 常规遍历

知识点-延迟计算

立即计算

使用文件游标时，有些计算会立即执行，比如游标的分组聚合`cs.groups()`。

代码示例

立即计算：
所有订单的价格总和

	A
1	<code>=order_cursor.groups(;sum(price):total)</code>

立即计算得出结果

延迟计算

使用文件游标时，有些计算在游标上定义之后不会马上计算，在游标遍历时才实际计算。比如游标的过滤`cs.select()`和生成新游标`cs.new()`。

内存计算：
订单折扣小于0.5的价格总和

	A
1	<code>=order.select(discount<0.5)</code>
2	<code>=A1.groups(;sum(price):total)</code>

A1立即计算条件过滤的结果
A2利用A1的结果计算汇总

延迟计算的作用

延迟计算可以一次遍历完成多步计算，不必生成中间结果占用空间。
而且，代码和内存运算差不多，容易理解。

延迟计算：
订单折扣小于0.5的价格总和

	A
1	<code>=order_cursor.select(discount<0.5)</code>
2	<code>=A1.groups(;sum(price):total)</code>

A1没有立即计算，无中间结果，
A2一次遍历，计算过滤和汇总

课堂练习p2.1-延迟计算

练习：打开p2.1.dfx，该dfx为立即计算dfx，
执行并记录时间

	执行时间（毫秒）
立即计算	
延迟计算	

练习：改写p2.1.dfx，将立即计算的表达式修改为延迟计算的写法，
执行并记录时间

知识点-游标前过滤

列存文件游标前过滤

折扣率大于0.9，按区域分组，合同额求和

列存文件

区域	合同额	折扣率	...
...	
1089	2845.6	0.8	...
2316	2913.7	0.95	...
...
8001	3029.4	0.6	...
...

代码示例

游标前过滤

	A
1	=order_file.open().cursor(discount>0.9)
2	=A1.groups(area;sum(price):amount)

减少读取的列

按照条件，先读取折扣率字段，如果不满足条件，就可以不再读取区域和合同额。

利用minmax索引

按块读取折扣率时，先判断minmax索引，如果不符合条件，即可跳过这一块。

游标过滤

将游标分批读入内存，边读边过滤。无法利用上述两个办法提速。

游标过滤

	A
1	=order_file.open().cursor()
2	=A1.select(discount>0.9).groups(area;sum(price):amount)

课堂练习p2.2-游标前过滤

练习：打开p2.2.dfx，该dfx为游标过滤，
执行并记录时间

	执行时间（毫秒）
游标前过滤	
游标过滤	

练习：改写p2.2.dfx，改为游标前过滤，执行并记录时间

补充阅读 Java和C++从硬盘读取数据需要对象化，Java生成对象非常慢，C语言相对快很多。如果把很多列都读入内存再判断，会生成很多没有用的对象，对性能影响较大。

知识点-过滤条件

包含多个过滤条件的遍历

条件1：月薪小于50,000的员工
条件2：姓“张”的员工
说明：红色字表示真正执行的计算

条件1	条件2	&&后
ture	ture	true
false	false	false
ture	false	false
ture	ture	ture
ture	false	false
ture	ture	ture

条件1在前，共计算11次

条件2	条件1	&&后
ture	ture	true
false	false	false
false	ture	false
ture	ture	ture
false	ture	false
ture	ture	ture

条件2在前，共计算9次

多个过滤条件的关系

多个条件是“并且”关系时，要注意：如果前面的子项为假时，后面就不会再计算了。

书写多个条件的顺序

月薪小于50,000的员工数很多（条件1），姓“张”的员工数（条件2）相对较少。把结果集较小的条件2写到前面，后面条件项1在较小的前项结果集再过滤，会减少计算次数。

代码示例

	A
1	=employee_ctx.open().cursor(;like(name,"张*") && salary < 50000).fetch()

课堂练习p2.3-过滤条件

练习：打开p2.3.dfx，执行并记录时间

	执行时间（毫秒）
orderdate在前	
amt在前	

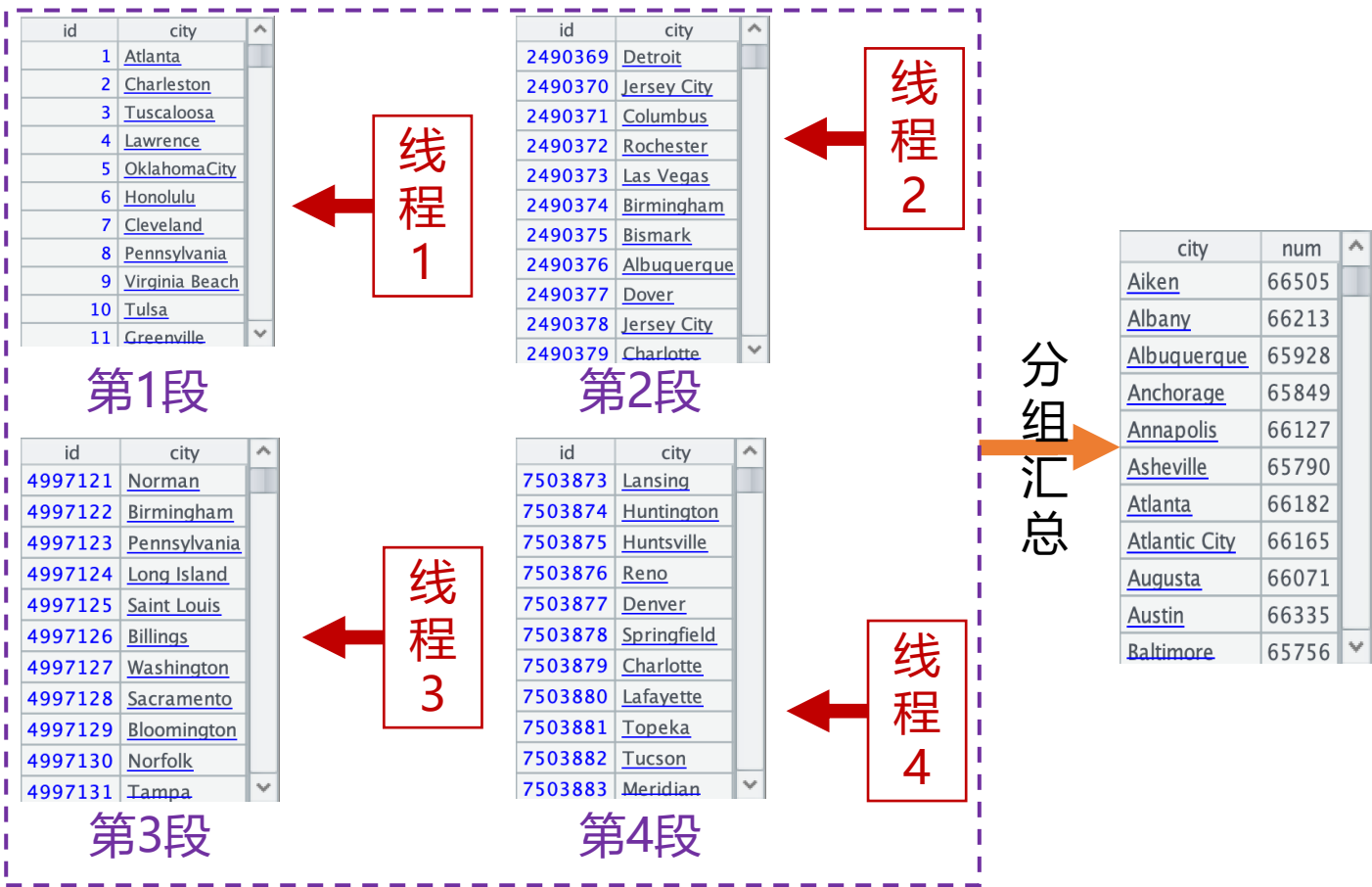
练习：改写p2.3.dfx，将金额过滤条件放前面，执行并记录时间

补充阅读 关系数据库例如Oracle，对过滤条件有自动的优化机制，但是很难实现人为的精确控制。

知识点-多路游标（内存）

用多路游标并行遍历内存数据

按城市统计用户数



内存

内存多路游标

内存数据分段后，可以使用多路游标，用多个线程并行遍历计算。图中4个线程同时计算分组汇总，要比单线程快。

内存多路游标的使用

用多路游标计算分组聚合的方法，与普通游标一样。

代码示例

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

课堂练习p2.4-多路游标（内存）

练习：打开p2.4.dfx，执行并记录时间

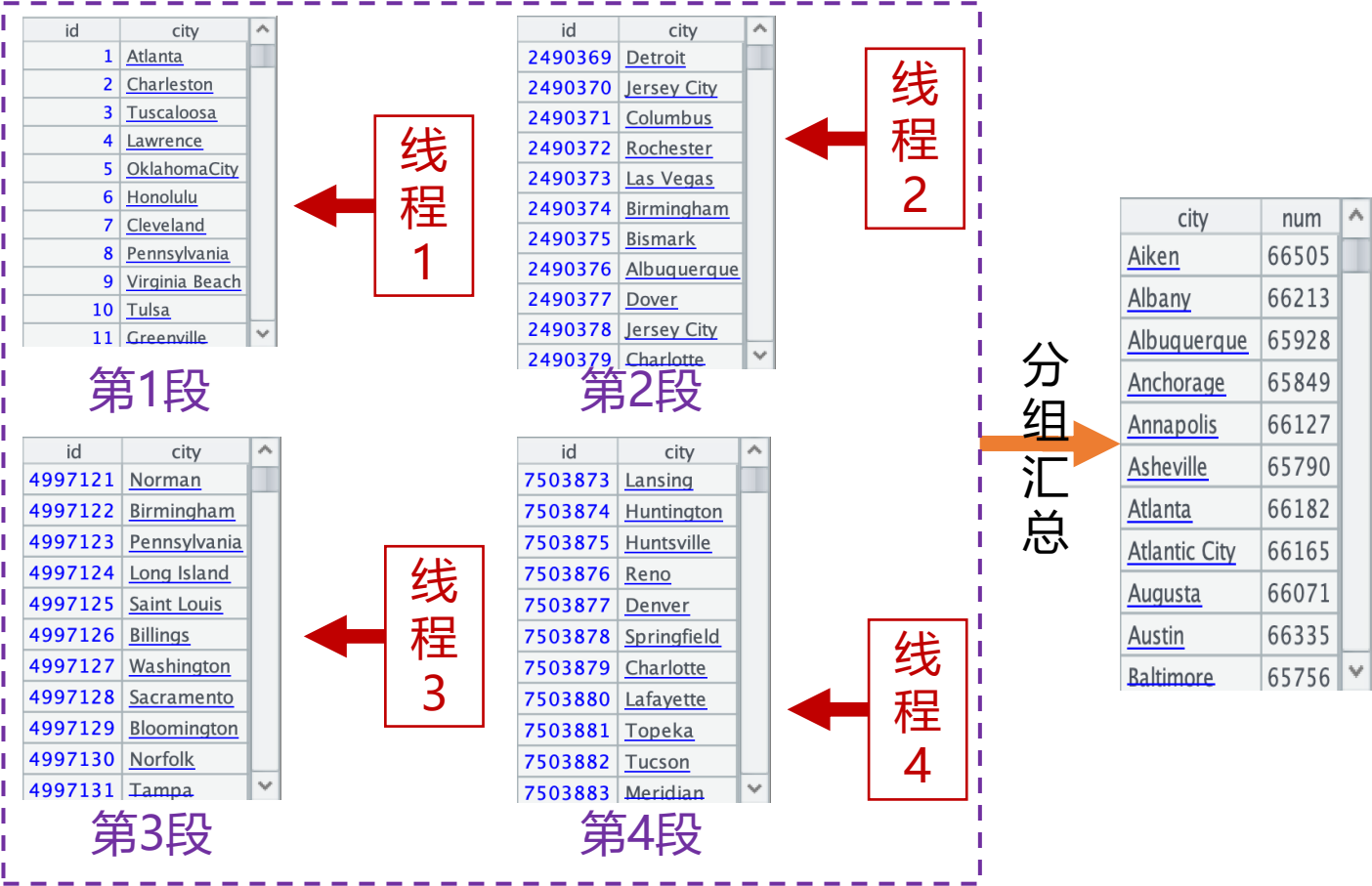
	执行时间（毫秒）
单路游标	
多路游标	

练习：改写p2.4.dfx，将单路游标改为多路（4路）游标，执行并记录时间

知识点-多路游标（外存）

用多路游标并行遍历硬盘文件

统计各城市下身高度大于170cm的男性用户



硬盘文件

外存多路游标

硬盘文件分段后，也可使用多路游标。图中4个线程同时计算分组汇总，要比单线程快。同时读取文件，一般也比单线程快。
硬盘并发能力比内存差很多。用到的列很多时，列存、机械硬盘情况下，多路游标不一定快。

外存多路游标的使用

用多路游标计算分组聚合的方法，与普通游标一样。

代码示例

```
1 =user_ctx.open().cursor@m(gender == "M" && height>170;4)
2 =A1.groups(city;count(~):num)
```

课堂练习p2.5-多路游标（外存）

	执行时间（毫秒）
单路游标	
多路游标	

练习：打开p2.5.dfx，执行并记录时间

练习：改写p2.5.dfx，将单路游标改为多路（4路）游标，执行并记录时间

第二章 遍历

2.3 分组排序

知识点-小分组

分组结果集较小，内存可以装下

oid	type	amt
1	249	306.98
2	392	804.41
3	379	649.21
4	897	812.69
5	505	855.37
6	821	234.11
7	622	879.85
8	624	438.41
9	142	371.72
10	201	746.29
11	321	603.40

硬盘文件

分组
汇总

key type	count
30	1
47	3
55	1
84	1
91	1
92	1
102	1
104	1
108	1
138	1
142	1

内存分组结果

小分组

订单数据几亿条，需在硬盘上存储。按类别统计订单数量。类别几万个，分组结果集较小，内存可以装下，因此称为小分组。

小分组执行步骤

用游标分批读取硬盘上的订单数据，在内存中依次用哈希法找到本组，向结果集聚合。

例如：当前要处理的记录订单类别是92。在内存结果集中用哈希法查找到92，对应计数加1。如果找不到，结果集就增加一条记录，值是“类型：91；计数：1”。

汇聚过程中产生结果集的数据量不会超过类别的总数，因此不占用过多内存。

代码示例

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

知识点-大分组

分组结果集很大，内存无法装下

硬盘文件

oid	...	custid	amt	^
1	...	249	306.98	
2	...	392	804.41	
3	...	379	649.21	
4	...	897	812.69	
5	...	505	855.37	
6	...	821	234.11	
7	...	622	879.85	
8	...	624	438.41	
9	...	142	371.72	
10	...	201	746.29	
11	...	321	603.40	
12	...	438	505.25	

结果输出游标

custid	count	^
30	1	
47	3	
55	1	
84	1	
91	1	
92	1	
102	1	
104	1	
108	1	
138	1	
142	1	
144	1	

分批汇总

多路归并

自动估算内存可容纳的数据条数，决定分多少批

tmp
tmpdata4458383290554497206
tmpdata6838364572503028301
tmpdata85319640366422427
tmpdata4668858076644814483

硬盘缓存文件

大分组

订单数据量几亿条，在硬盘上存储。按客户号分组统计订单数量。客户几千万个，分组结果内存装不下，因此称为大分组。

大分组计算步骤

用游标分批读硬盘上的订单数据，分组汇总后生成临时缓存文件，这些缓存文件都对客户号有序。

用多路游标有序归并缓存文件，输出客户号有序的单路游标。具体的，从多路游标中找最小客户号，再找是否有相同客户号，如有就汇总，然后输出。

大分组注意事项

大分组用时较长，不适合实时计算。
输出游标取数结束或关闭后，临时文件将自动删除。

知识点-大分组

分组结果集很大，内存无法装下

硬盘文件

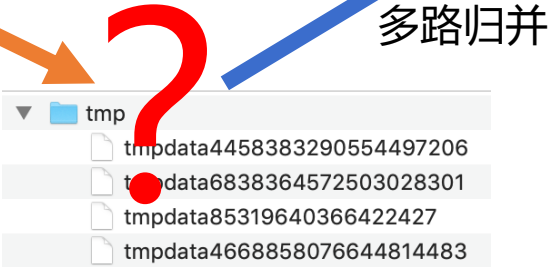
oid	...	custid	amt
1	...	249	306.98
2	...	392	804.41
3	...	379	649.21
4	...	897	812.69
5	...	505	855.37
6	...	821	234.11
7	...	622	879.85
8	...	624	438.41
9	...	142	371.72
10	...	201	746.29
11	...	321	603.40
12	...	438	505.25

结果输出游标

custid	count
30	1
47	3
55	1
84	1
91	1
92	1
102	1
104	1
108	1
138	1
142	1
144	1

分批汇总

自动估算内存可容纳的数据条数，决定分多少批



硬盘缓存文件

判断是否要写缓存文件的机制

预先计算出来内存可以容纳的客户分组数量，称为缓冲区行数。

大分组开始计算后，分组结果集会在内存中汇聚变大，达到缓冲区行数之后，就必须写出到缓存文件中。

之后，清空内存结果集继续计算大分组。

缓冲区行数

如果直到大分组完成也没有达到缓冲区行数，就不会写缓存文件，大分组就变成小分组了。

缓冲区行数缺省自动计算，也可以人为指定。学习大分组时可设置较小值，强制输出缓存文件。

代码示例

	A
1	<code>=order_cursor.groupx(custid;count(~):count).fetch(10)</code>

课堂练习p3.1-大分组

练习：打开p3.1.dfx，观察按照custid大分组结果，记录时间。

	执行时间（毫秒）
小分组	
大分组	

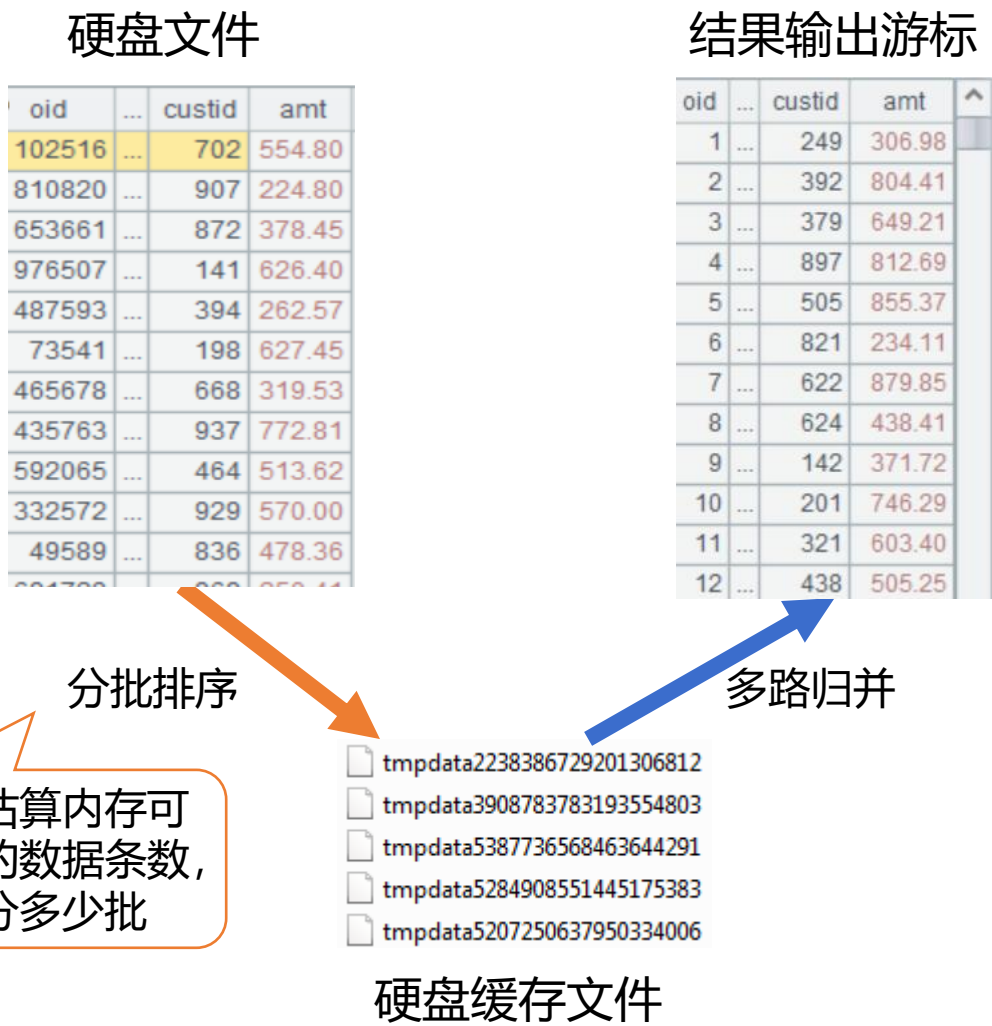
练习：改写p3.1.dfx，改成按照custid字段的小分组结果，记录时间。

思考：

- 1、为什么大分组n设置为400？
- 2、大分组为什么只能返回单路游标，不能返回多路游标？

知识点-大排序

要排序的数据量很大，内存无法装下



大排序

订单数据量几亿条，需在硬盘上存储。按订单编号排序，结果集内存装不下。

大排序计算步骤

用游标分批读取硬盘上的订单数据，排序后生成硬盘上的临时缓存文件，这些缓存文件都对订单号有序。

用多路游标有序归并缓存文件，输出客户号有序的单路游标。

大排序注意事项

大排序用时较长，不适合实时计算。

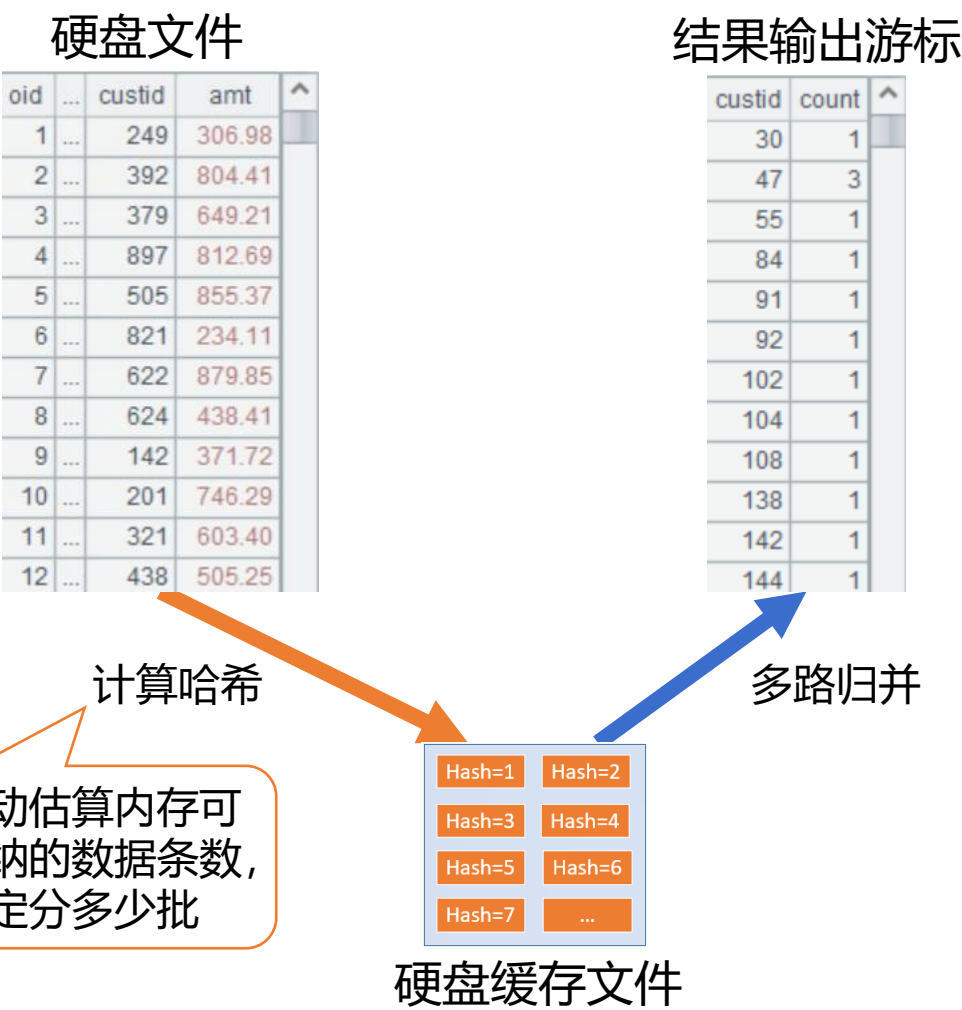
大排序也有缓冲区行数，作用和大分组类似。

代码示例

	A
1	=order_ctx.open().cursor().sortx(oid).fetch(10)

知识点-哈希大分组

分组结果集很大，内存无法装下



哈希大分组

订单数据量几亿条，在硬盘上存储。按客户号统计订单数量，客户几千万个，分组结果内存装不下。

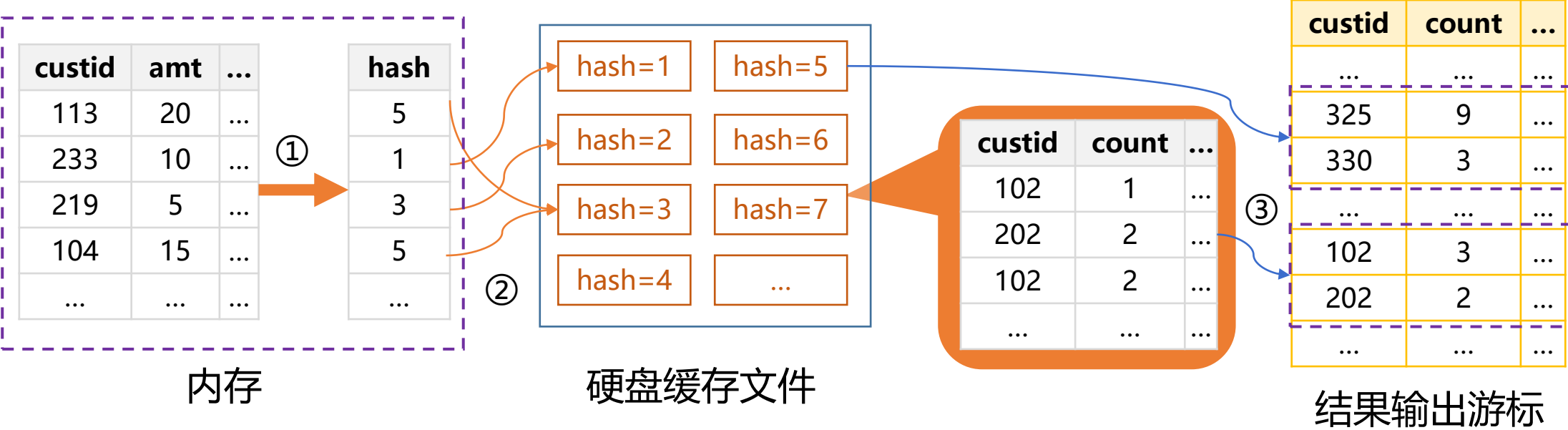
哈希大分组注意事项

- 哈希大分组用时较长，不适合实时计算。
- 输出游标取数结束或关闭后，临时文件将自动删除。
- 哈希大分组也有缓冲区行数，作用和大分组一致。
- 注意哈希大分组的结果无序。

代码示例

	A
1	=order_cursor.groupx@u(custid;count(~):count).fetch(10)

知识点-哈希大分组



哈希大分组计算步骤

①订单数据分批读入内存，计算客户号哈希值。

同一客户号哈希值一定相同。
哈希函数不单调，一个哈希值对应多个客户号。

②按哈希值，将汇总结果追加到缓存文件。

一个客户号会汇总到同一文件。
一个文件有多个客户号，如果太多，无法小分组，要重新哈希。

③缓存文件按哈希值依次在内存中按客户号小分组、汇总，输出单路游标。

由于哈希函数对排序字段非递增，所以分组结果无序。

课堂练习p3.2-哈希大分组

练习：打开p3.2.dfx，观察按照custid字段的大分组结果。

	执行时间（毫秒）
大分组	
哈希大分组	

练习：改写p3.2.dfx，改成哈希大分组，比较执行时间。

思考：

1、哈希大分组为什么只能返回单路游标，不能返回多路游标？

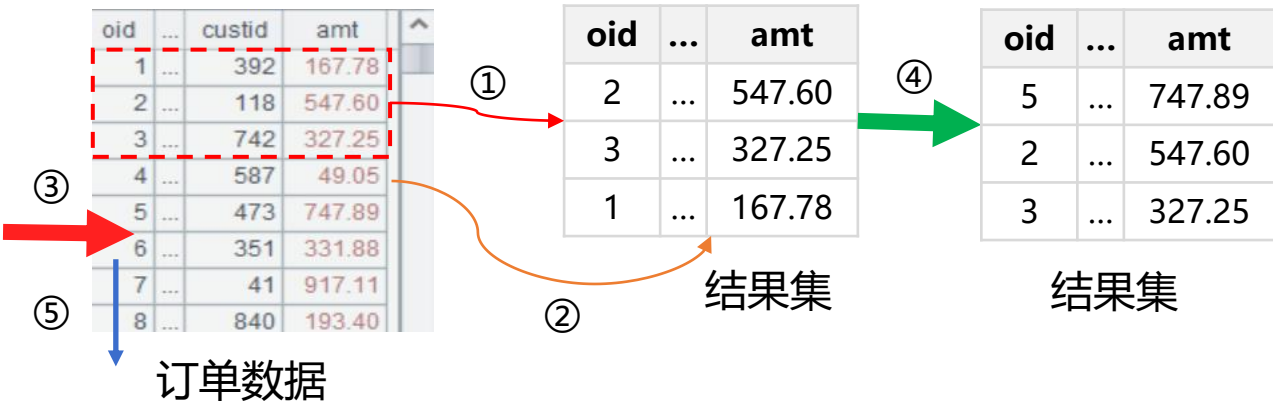
知识点-聚合理解

计算订单金额最大的前三名

不把topN理解成聚合



把topN理解成聚合



不把topN理解成聚合

关系数据库的做法是：大排序之后再取前三个。大排序，特别是硬盘上数据文件的排序非常耗时，所以性能差。

把topN理解成聚合

- ①从订单数据中取前三条记录，按照金额降序排列，形成结果集。
- ②取订单数据第四条记录，和结果集比较。第四条金额小于前三条，所以结果集不变。
- ③继续取第五条记录，和结果集比较。
- ④第五条金额比结果集所有记录都大，所以放在结果集第一条，结果集原来的第一、二条变为二、三条。原来的第三条舍弃。
- ⑤继续计算第六条记录，以此类推。一次遍历，计算出前三名。

知识点-聚合理解

计算订单金额最大前三名

代码示例

	A
1	=order_ctx_cursor.groups(top(-3;amt))

每个客户的金额top3的订单



全集上的topN

全部数据中取topN，例如：所有订单中的金额前三名。

分组内的topN

求按照客户分组，每组内的金额前三名订单。
将topN理解成聚合后，分组内的topN原理上和全集的topN计算步骤基本相同。不同点是每个分组分别做聚合。

代码示例

	A
1	=order_ctx_cursor.groups(custid;top(-3;amt):top3)

课堂练习p3.3-聚合理解

练习：打开p3.3.dfx，观察全部订单金额top10记录。

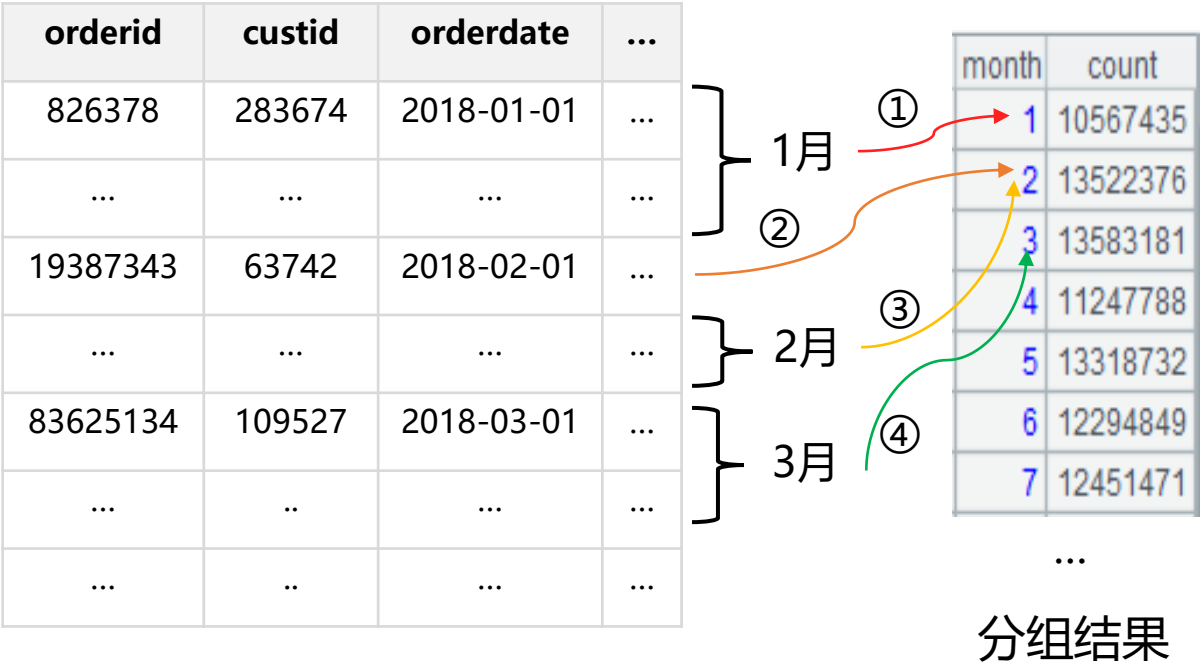
练习：改写p3.3.dfx，改成按照订单日期字段分组，求分组金额top3记录。

补充阅读 关系数据库并不把topN当成聚合运算，而是返回结果集的一种修饰符。也就是说，SQL会先对所有数据大排序，之后只取前 N 条返回。大排序非常耗时，影响性能。

知识点-有序分组

原数据按分组字段有序，小结果集

订单按日期有序，统计2018年每月订单数



订单数据

分组结果

说明：物理实现上，应该批量读取到内存中分组。这里所说的步骤为了容易描述而做了简化。

小分组

- ①从订单数据中读取1月数据，从第二条开始，每条的月份都和结果集最后一条相同，所以都汇聚到结果集同一结果中。
- ②读入第一条2月份的数据。月份2和结果集中最后一条不同，所以在结果集写入新一条数据，月份为2。
- ③读入2月份其他数据，月份和结果集最后一条相同，所以汇聚到同一条中。
- ④其他月份以此类推。

小分组有序分组优点

内存中依次聚合的时候，只和结果集最后一条比较即可，不需哈希查找结果集，所以性能更好。

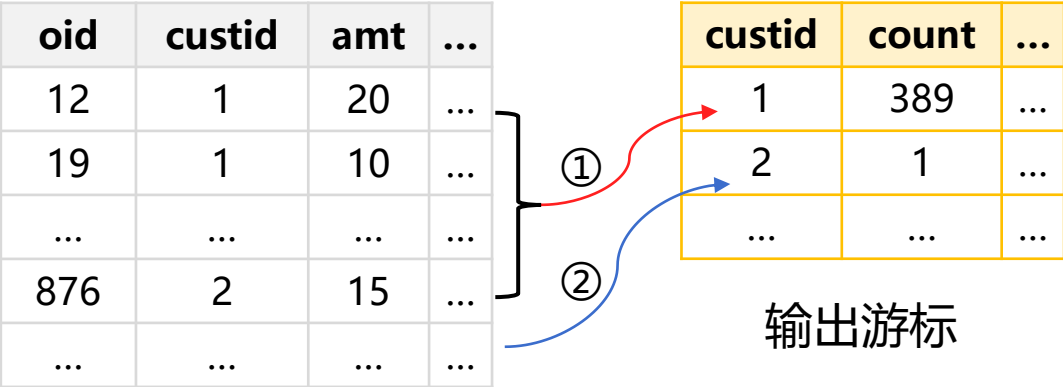
代码示例

	A
1	=order_ctx_cursor.groups@o(month(orderdate):month;count(~):count)

知识点-有序分组

原数据按分组字段有序，大结果集

订单按日期有序，统计2018年每月订单数



订单数据

输出游标

说明：物理实现上，读入内存中分组和输出游标都是批量。这里的步骤做了简化。

大分组

- ①从订单数据中读取客户1，每条客户号都和内存结果集最后一条相同，所以都汇聚到结果集同一条结果中。
- ②读入第一条客户2的数据。客户号和内存结果集中最后一条不同，所以在结果集写入新一条数据，客户号为2，同时游标输出客户1、计数389。
- ③其他订单数据以此类推。

大分组有序分组优点

内存中依次聚合的时候，只和结果集最后一条比较即可，不需哈希查找结果集，所以性能更好。

而且，不需缓存文件，直接输出结果。

代码示例

	A
1	=order_ctx_cursor.group(custid;count(~):count).fetch(100)

课堂练习p3.4-有序分组

练习：打开p3.4.dfx，观察按照custid字段的小分组结果。

	执行时间（毫秒）
小分组	
有序分组	

练习：改写p3.4.dfx，先按照分组字段custid排序输出到文件中，然后计算小分组结果对比执行时间。

提示： `cs.sortx(custid)`

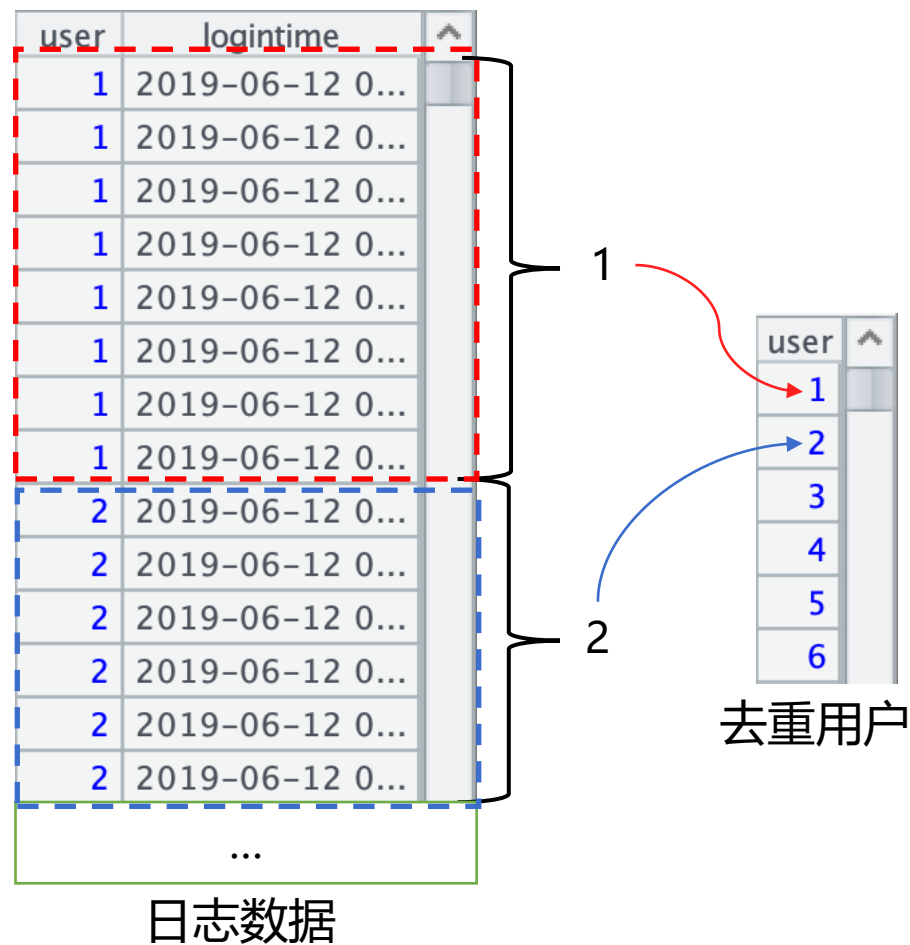
补充阅读 关系数据库的理论基础是无序集合，不能保证数据在物理存储时的有序性，无法利用存储上的次序，无法实现有序分组。

后面需要有序的计算都属于类似的情况，不再赘述。

知识点-有序去重

原数据按某字段有序，归并分组快速去重

计算日志数据中出现过的去重用户数量



小结果集

相当于分组后只取分组字段，因为有序，只和相邻的比。去重计算结果在内存中再计数。

代码示例

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

大结果集

相当于分组后只取分组字段，因为有序，只和相邻的比。去重计算结果输出到游标，通过游标计数，不必全读入内存。

代码示例

	A
1	=Log_cursor.group@1(user).skip()

课堂练习p3.5-有序去重

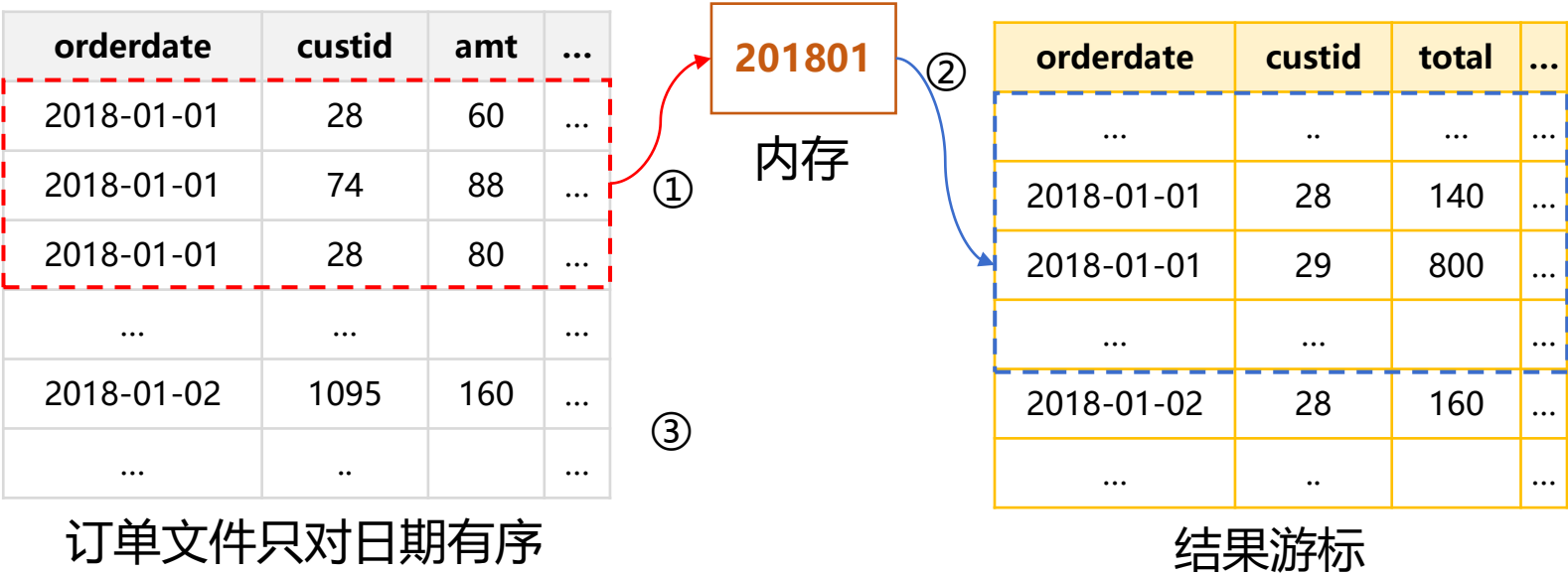
练习：打开p3.5.dfx，观察custid字段去重的小结果集计算。

练习：改写p3.5.dfx，改成采用大结果集方式去重计算。

思考：两者有什么区别？

知识点-半序分组

按日期、客户分组汇总订单金额



半序分组的优势

利用前半部分字段有序，避免大分组读写硬盘临时缓存文件，提高性能。

半序分组的条件

- 1、原数据按照日期有序，客户号无序。
- 2、每天数据量都不太大，内存能装下。

半序分组计算步骤

- ①读取一天的数据，在内存中按客户号分组汇总。
- ②游标输出一天的汇总结果。
- ③继续计算其他日期。

代码示例

	A
1	=order_cursor.group@q(orderdate;custid;sum(amt):total).fetch(100)

课堂练习p3.6-半序分组

练习：打开p3.6.dfx， orders_sort_custid.ctx对 custid字段有序，观察custid、 orderdate字段大分组结果，记录执行时间。

	执行时间（毫秒）
大分组	
半序分组	

练习：改写p3.6.dfx，改成半序分组方式计算，结果对比执行时间。

知识点-半序排序

按日期、客户分组汇总订单金额

orderdate	custid	amt	...
2018-01-01	28	60	...
2018-01-01	74	88	...
2018-01-01	28	80	...
...
2018-01-02	1095	160	...
...

订单文件只对日期有序



orderdate	custid	total	...
...
2018-01-01	28	60	...
2018-01-01	28	74	...
...
2018-01-02	1095	160	...
...

结果游标

半序排序的优势

利用前半部分字段有序，避免大排序读写硬盘临时缓存文件，提高性能。

半序排序的条件

- 1、原数据按照日期有序，客户号无序。
- 2、每天数据量都不太大，内存能装下。

半序排序计算步骤

- ①读取一天的数据，在内存中按客户号排序。
- ②游标输出一天的排序结果。
- ③继续计算其他日期。

代码示例

	A
1	=order_cursor.group@qs(orderdate;custid).fetch(100)

课堂练习p3.7-半序排序

练习：打开p3.7.dfx，orders_sort_custid.ctx对custid字段有序，观察custid、orderdate字段大排序结果，记录执行时间。

	执行时间（毫秒）
大排序	
半序排序	

练习：改写p3.7.dfx，改成半序排序方式计算，结果对比执行时间。

知识点-序号小分组

按照客户号统计订单数量

序号小分组计算步骤

订单数据分批读入内存。

根据客户号数值，直接在内存结果集找到对应的位置，计算计数值。

例如：客户号为1002，直接定位到结果集的1002号位置。

序号小分组优点

结果集直接按照位置定位，比哈希计算查找本组要快，原数据量很大时，性能提升明显。

custid	odate	...
1002	2017-11-19	...
909	2017-12-21	...
908	2017-12-21	...
1002	2018-01-22	...
...		...

分批读入内存计算

序号	custid	count	...
...
908	908	9	...
909	909	3	...
...			
1002	1002	10	...
...

内存结果集直接定位

代码示例

	A
1	=order_cursor.groups@n(custid;count(~):count)

知识点-序号小分组

按照年月统计订单数量

odate	...	ym
2017-11-02	...	201711
2017-12-19	...	201712
2018-01-22	...	201801
2017-11-21	...	201711
...

分批读入内存计算

序号	odate	count	...
...
201711	2017-12-19	9	...
201712	2018-01-22	3	...
这里有89个空白位置			
201801	2017-11-21	10	...
...

内存结果集直接定位

假如月份从201711开始，那么这里有201710个空白。
年月分组会浪费很多内存。

序号分组适用场景

序号分组适合分组表达式值是自然数区间的情况，值要尽量连续，不能太离散。

不适用场景

年月计算出来的数值，月份最大是12。而内存结果集的序号是连续的，会出现很多空白。

解决办法

可将年月转为从1900年01月开始的月数，就可以避免空白了。

课堂练习p3.8-序号小分组

练习：打开p3.8.dfx，观察custid字段的小分组结果，记录执行时间。

	执行时间（毫秒）
小分组	
序号小分组	

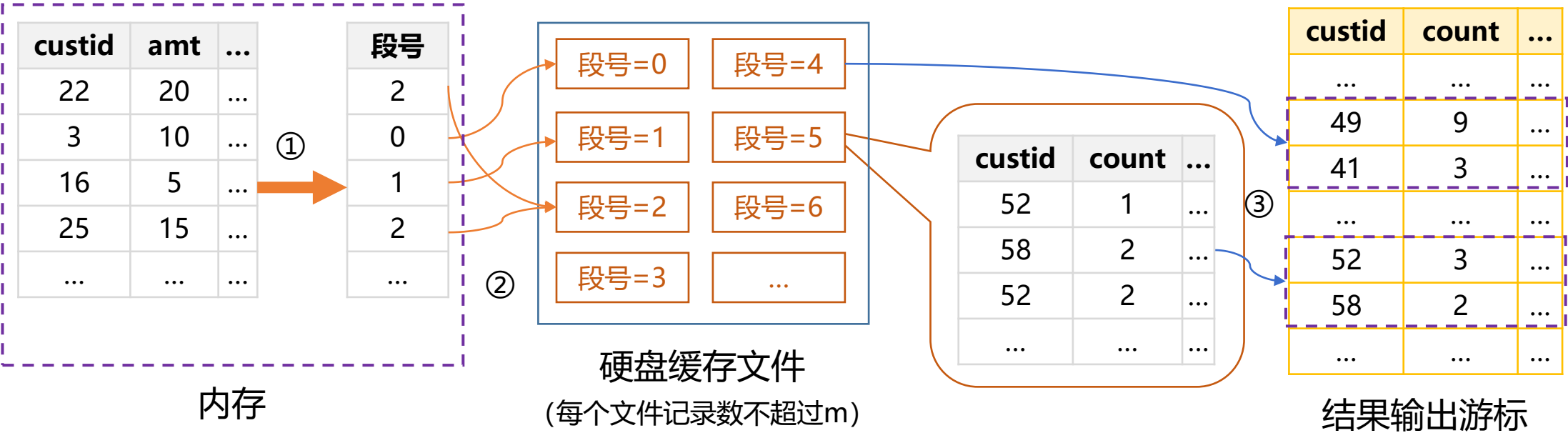
练习：改写p3.8.dfx，改成序号分组方式计算，结果对比执行时间。

补充阅读 基于无序集合理论的关系数据库，没有提供序号定位的手段。即使可以用序号定位时，也只能用主键查找。

知识点-序号大分组

按照客户统计订单数量

假设内存最多容纳记录数 $m=10$



序号大分组计算步骤

- ①分批把订单数据读入内存，客户号除以 m 取整作为对应段号。按段号在内存直接定位本段位置。
- ②把内存分组汇总结果追加到不同段号的缓存文件。相同客户号的记录会存入同一个文件。同一段文件有多个客户号。
- ③缓存文件按段号依次在内存中按照客户号小分组、汇总，输出单路游标。由于段号实际上是对客户号递增，所以结果也对客户号有序。

知识点-序号大分组

按照客户统计订单数量

代码示例		A
	1	=order_ctx_cursor.groupx@n(custid;count(~):count)

序号大分组适用场景

序号大分组适合分组表达式值是自然数区间的情况，值要尽量连续，不能太离散。要根据实际情况因地制宜的选择是否采用序号大分组。

分段表达式

序号大分组需要估算内存最多容纳的记录数，但是经常估计不准，会降低性能。
可预先确定一个分段表达式来计算段号。例如：客户约有一千万，可用客户号除以十万的商作为段号。总段数约一百个，每段约十万条记录，内存可以放的下。

代码示例		A
	1	=order_ctx_cursor.groupx@n(custid;count(~):count;custid\100000)

课堂练习p3.9-序号大分组

练习：打开p3.9.dfx，观察custid字段大分组结果，记录执行时间。

	执行时间（毫秒）
大分组	
序号大分组	

提示：假设内容能容纳400行结果集，所以缓冲区行数应设置为400。

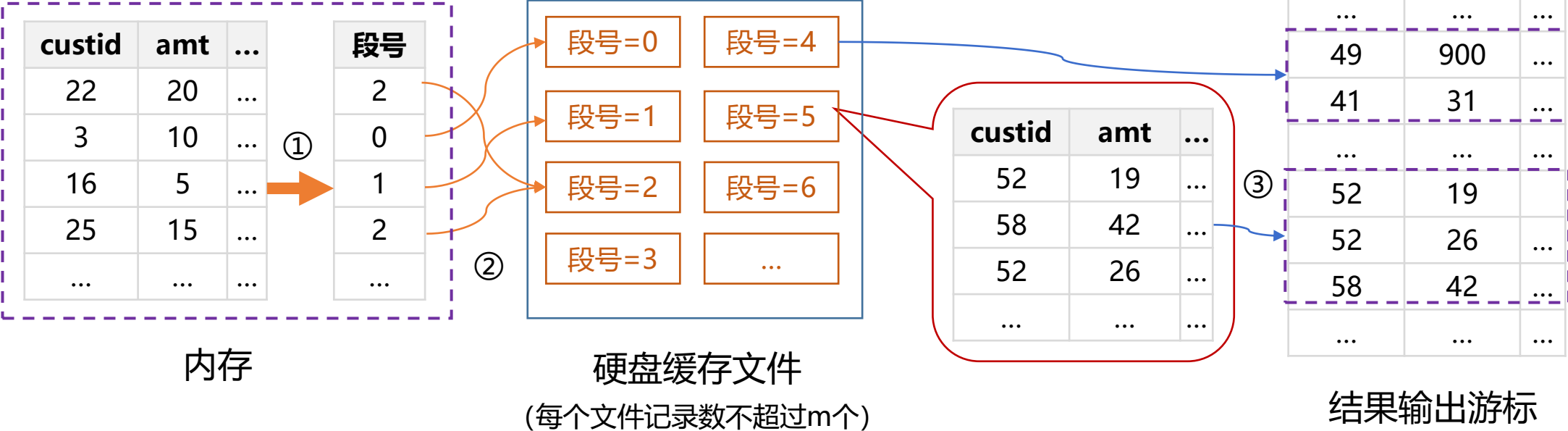
练习：改写p3.9.dfx，改成序号大分组方式计算，结果对比执行时间。

提示：假设内存能容纳400行结果集，总数据量有500条，分段表达式应该是custid\400,分为两段。

知识点-序号大排序

订单按照客户排序

假设内存最多容纳记录数 $m=10$



序号大分组计算步骤

- ①分批把订单数据读入内存，客户号除以 m 取整作为对应段号。按段号在内存直接定位本段位置。
- ②内存明细数据追加到每个段号的缓存文件。相同客户号的记录会存入同一个文件。同一段文件有多个客户号。
- ③缓存文件按段号依次在内存中按照客户号排序，输出单路游标。由于段号实际上是对客户号递增，所以结果对客户号有序。

知识点-序号大排序

订单按照客户排序

代码示例

	A
1	=order_ctx_cursor.sortx@n(custid)

序号大排序适用场景

序号大排序适合排序表达式值是自然数区间的情况，值要尽量连续，不能太离散。要根据实际情况因地制宜的选择是否采用序号大排序。

分段表达式

序号大排序需要估算内存最多容纳的记录数，但是经常估计不准，会降低性能。

可预先确定一个分段表达式来计算段号。例如：订单约有一个亿，客户有一千万，可用客户号除以一万得到段号。总段数约一千个，每段约十万条记录，内存可以放下。

代码示例

	A
1	=order_ctx_cursor.sortx@n(custid;custid\10000)

课堂练习p3.10-序号大排序

练习：打开p3.10.dfx，观察oid字段大排序结果，记录执行时间。

	执行时间（毫秒）
大排序	
序号大排序	

提示：假设内存能容纳一百万条，缓冲区设置为一百万。

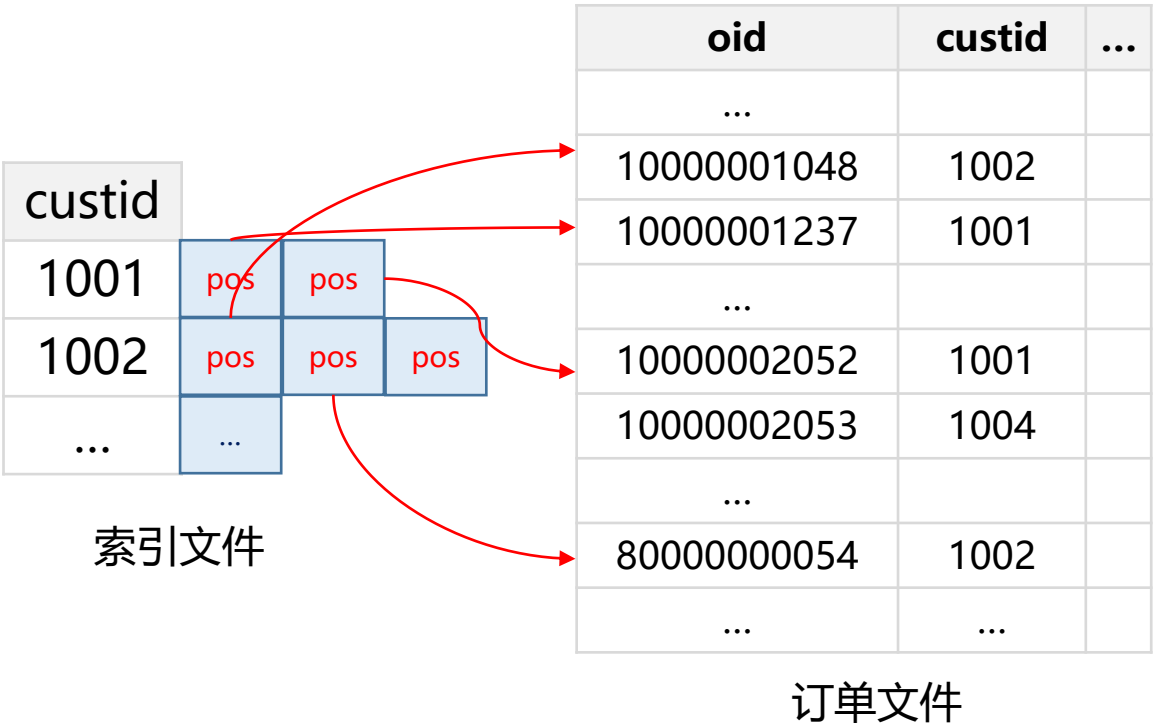
练习：改写p3.10.dfx，改成序号大排序方式计算，结果对比执行时间。

提示：假设内存能容纳一百万条，总数据量是一千万，所以分段表达式设置为oid\1000000，总段数是10。

知识点-利用索引排序

按照客户号对订单数据排序

订单文件本身是按照订单号有序



代码示例

```
A
1 =order_file.open().icursor@s(true,custid_idx).fetch(1000)
```

利用索引排序

客户号索引文件对客户号有序，利用索引查找订单文件的结果也就对客户有序。

利用索引排序的特点

利用索引排序很快有部分结果输出，不像大排序要等全部排完序才有输出。

但是，因为原文件无序，所以全部结果全部输出完比较慢。

利用索引排序适合的场景

例如：大文件排序后打印的情况。利用索引排序可尽快开始打印。而且，打印本身较慢，排序慢一些也感觉不到了。

排序后导出excel下载、低速率传输到异地都是类似的场景。

课堂练习p3.11-利用索引排序

练习：打开p3.11.dfx，观察custid字段大排序方式计算，记录执行时间。

	执行时间（毫秒）
大排序	
索引排序	

练习：改写p3.11.dfx，改成利用索引排序取出部分记录，记录执行时间。

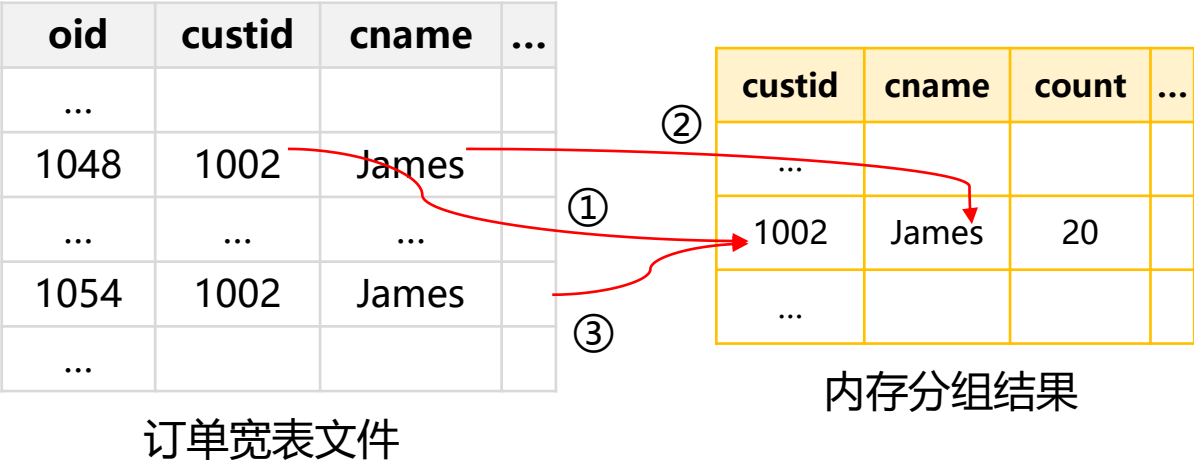
提示：索引已经预先建好，名称是custid_idx

思考：如果将排序结果取出一百万条，那个快？为什么？

知识点-分组维度冗余

按客户编号、姓名分组计算订单数量

姓名是冗余字段，编号相同的客户姓名一定相同



代码示例

小 分组		A
	1	=order_cursor.groups(custid;cname,count(~):count)
大 分组		A
	1	=order_cursor.groupx(custid;cname,count(~):count)

带 “冗余项”

- ①读入一条订单，客户号1002。计算1002的哈希值，在内存中查找到1002。
- ②计算James哈希值，组内查找James。找到后汇总。
- ③读入每条订单记录，都重复上两个步骤。

舍去 “冗余项”

舍去客户名，仅按照客户号分组，直接取本组第一个客户名，结果不变。

也就是省去②，每条记录都不必计算客户名哈希值，且不必查找客户名，记录数很多的时候性能提升明显。

小分组和大分组都可以提高性能。

课堂练习p3.12-分组维度冗余

练习：打开p3.12.dfx，观察orderdate, custid, cname三个字段小分组，记录执行时间。

	执行时间（毫秒）
三字段分组	
两字段分组	

练习：改写p3.12.dfx，改成orderdate, custid, 分组，另一个字段cname直接取，记录执行时间。

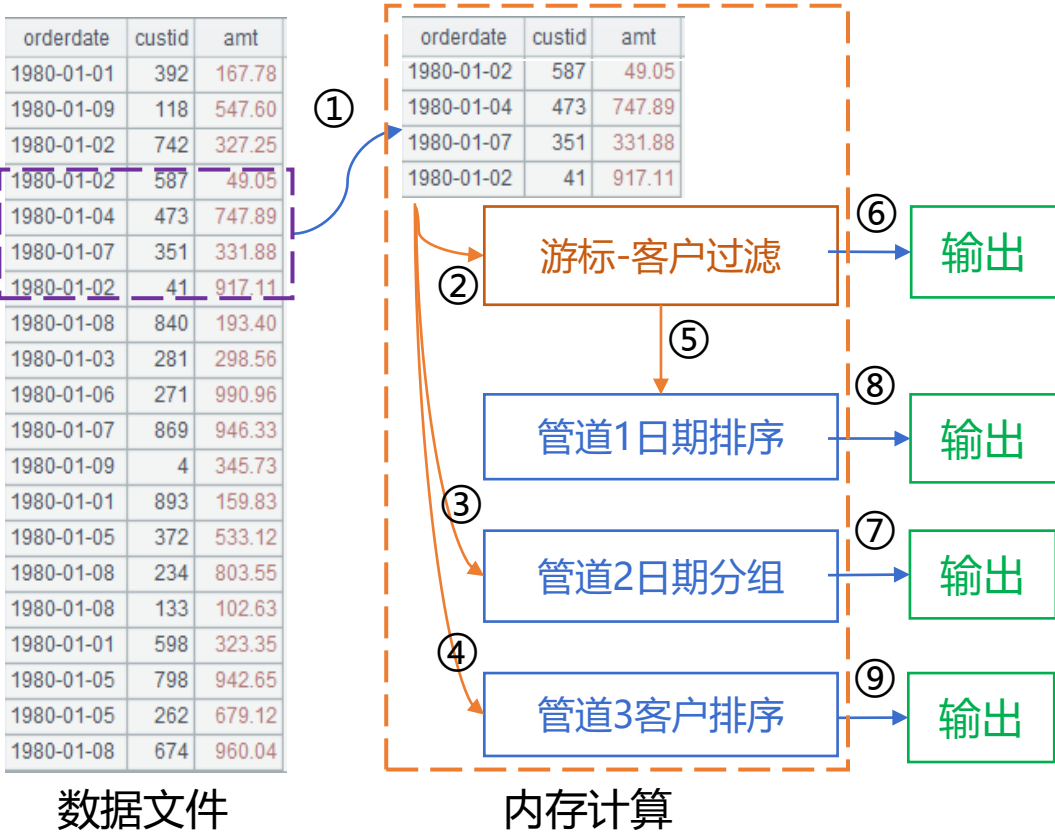
补充阅读 SQL可写为select id,max(name),sum(amt) from t group by id，也要比按照两个字段分组快，但是最后要计算max，不如直接取本组第一条的name要快。

第二章 遍历

2.4 高级遍历

知识点-遍历复用

一次遍历完成四种计算



游标：客户过滤； 管道1：客户过滤后日期排序；
管道2：客户分组汇总；管道3：日期分组汇总。

管道和游标的区别

单个游标定义的计算，在一次遍历时顺序执行，得到一个结果。

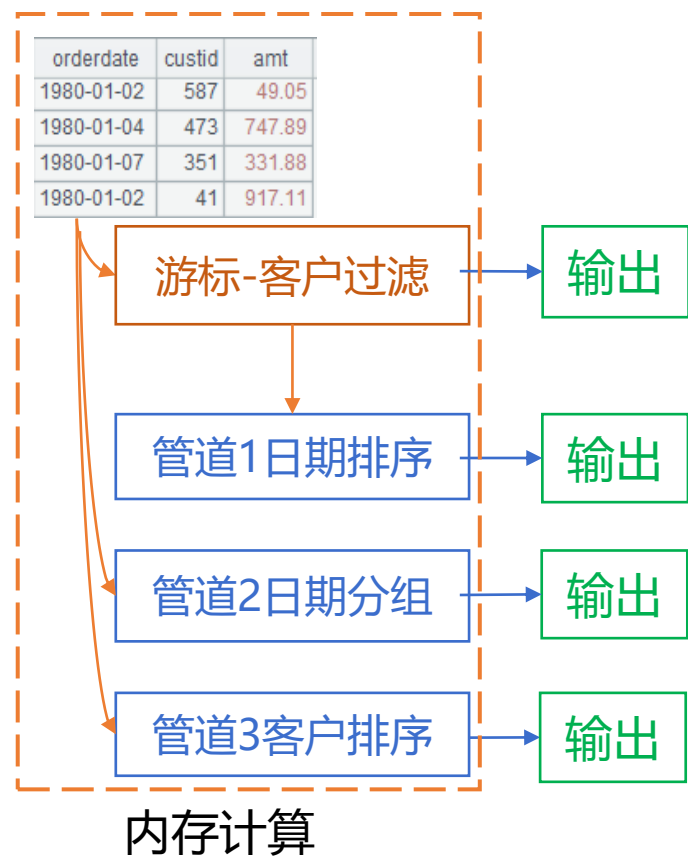
游标添加管道，在一次遍历时可同时计算，输出多个结果。

管道计算步骤

- ①游标分批读取数据进入内存。
- ②游标计算客户条件过滤。
- ③数据推送给管道2计算分组。
- ④数据推送给管道3计算排序，需要时写缓存文件。
- ⑤客户条件过滤结果推送给管道1，计算排序，需要时写缓存文件。
- ⑥游标输出全部结果。
- ⑦-⑨管道2、1、3依次输出结果。

代码示例-遍历复用

一次遍历完成四种计算

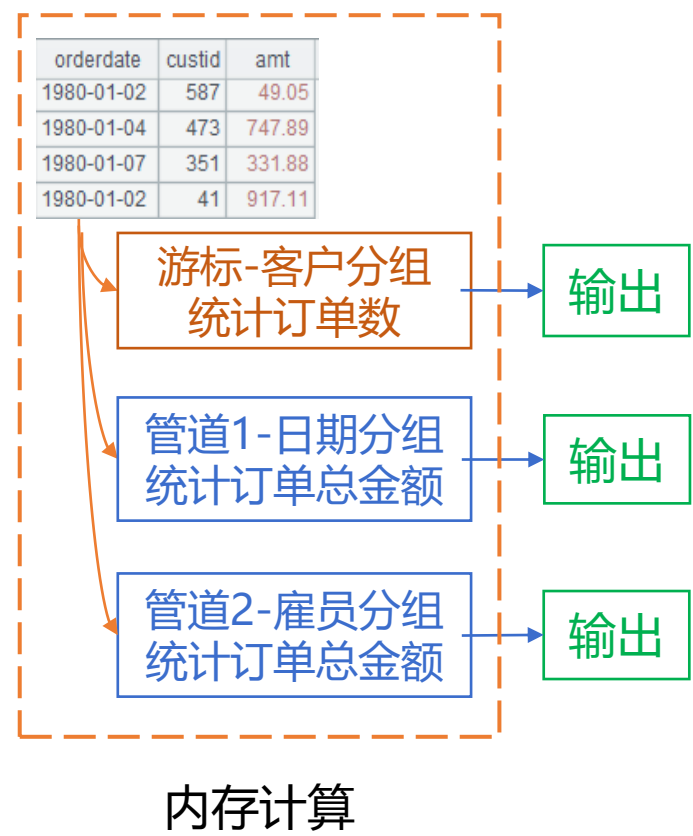


	A	B
1	=file.cursor@b()	/建立游标
2	=channel(A1).groups(orderdate;sum(amt):total)	/游标结果推送管道2
3	=channel().sortx(custid)	/建立管道3定义排序计算
4	=A1.push(A3)	/游标结果推送管道3
5	=A1.select(custid==473)	/游标定义过滤计算
6	=channel(A5).sortx(orderdate)	/游标过滤结果推送管道2
7	=A5.fetch()	/游标取全部结果
8	=A2.result()	/管道2输出结果
9	=A3.result().fetch(100)	/管道3结果游标取100条数据
10	=A6.result().fetch()	/管道1结果游标取所有记录

注意：游标必须全部遍历，否则管道结果输出不全。
例如：A7如果写作A5.fetch(100)，那么管道也只能得到部分结果。

代码示例-遍历复用

一次遍历完成多种分组



	A	B
1	=file.cursor@b()	/建立游标
2	cursor A1	=A2.groups(custid;count(~))
3	cursor	=A3.groups(orderdate;sum(amt))
4	cursor	=A3.groups(empid;sum(amt))

遍历复用比较常见的应用是，一次遍历完成多种不同方式的分组。

课堂练习p4.1-遍历复用

练习：打开p4.1.dfx，使用游标，分别统计每个客户号的订单数和每日销售总额。

	执行时间（毫秒）
仅用游标	
管道	

练习：改写p4.1.dfx，改成使用管道，一次遍历输出以上前者的两个统计结果，比较与前者的执行时间。

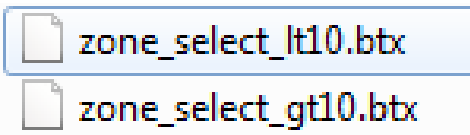
提示：分别用channel()函数和cursor关键字两种方式实现。

知识点-数据分拆

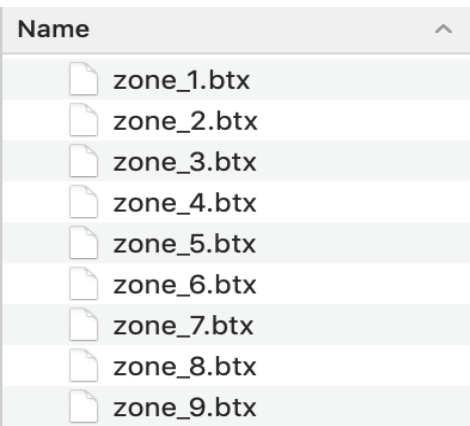
数据拆分到多个文件中

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

用户文件



过滤：区域号大于10和小于等于10的数据拆分为两个文件



分组：1000个区域分组后的数据独立存储到各文件

数据拆分

仅一次遍历，条件过滤或者分组的结果可以拆分到不同的文件中，也可以推送到多个管道中继续计算。

代码示例

过滤

	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)

分组

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

课堂练习p4.2-数据拆分

练习：打开p4.2.dfx，遍历两次，将原订单文件金额大于等于800以及小于800分为两个文件保存。

	执行时间（毫秒）
两次遍历	
一次遍历	

练习：改写p4.2.dfx，遍历一次，将原订单文件金额大于等于800以及小于800分为两个文件保存。

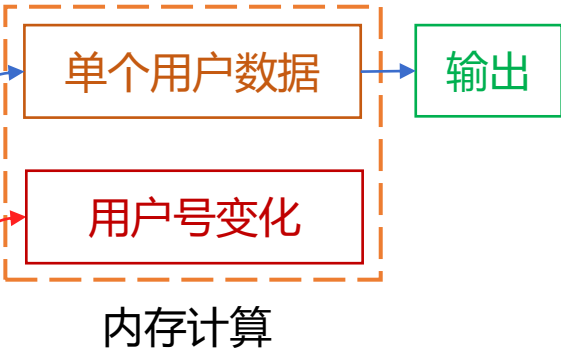
知识点-有序游标-字段变化

寻找拿铁爱好者

拿铁爱好者：2018年购咖啡和牛奶数量均大于10的用户
找到拿铁爱好者，把他们的订单存为latte2018.btx文件

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

某网上超市2018年
销售数据，按用户有序



用户行为分析

计算的特征：总数据量很大，但单个用户数据量不大。
计算很复杂，但是都针对单个用户。

有序游标

预先将数据按照用户排序。每次读入一个用户的数据，读到用户号变化为止，在内存中复杂计算后输出。一次遍历完成用户行为分析计算。

代码示例

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

课堂练习p4.3-有序游标-字段变化

练习：编写p4.3.dfx文件。

要求：对custid有序的文件orders_sort_custid.ctx，将那些在1980和1981年总消费额大于1850000的订单信息保存为order666.btx。

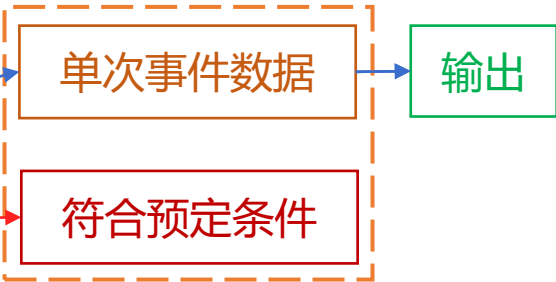
知识点-有序游标-变化条件

查询最大单次事件记录行数，以及对应的usertag

日志中以---flag---为事件的起始，也是上一个事件的结束



日志文件



日志分析

计算的特征：多行为一个整体事件，要根据条件划分事件。每个事件数据量不大，总的数据量很大。

有序游标

日志数据一般都按照时间排序。每次读入一个事件的数据，读到符合预定条件的记录，也就是下一个事件开始为止。在内存中复杂计算后输出。一次遍历完成日志分析计算。

代码示例

	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)

注：B2中A2.len()-2，因为要去掉flag以及usertag这两行

课堂练习p4.4-有序游标-条件变化

练习：编写代码p4.4.dfx。

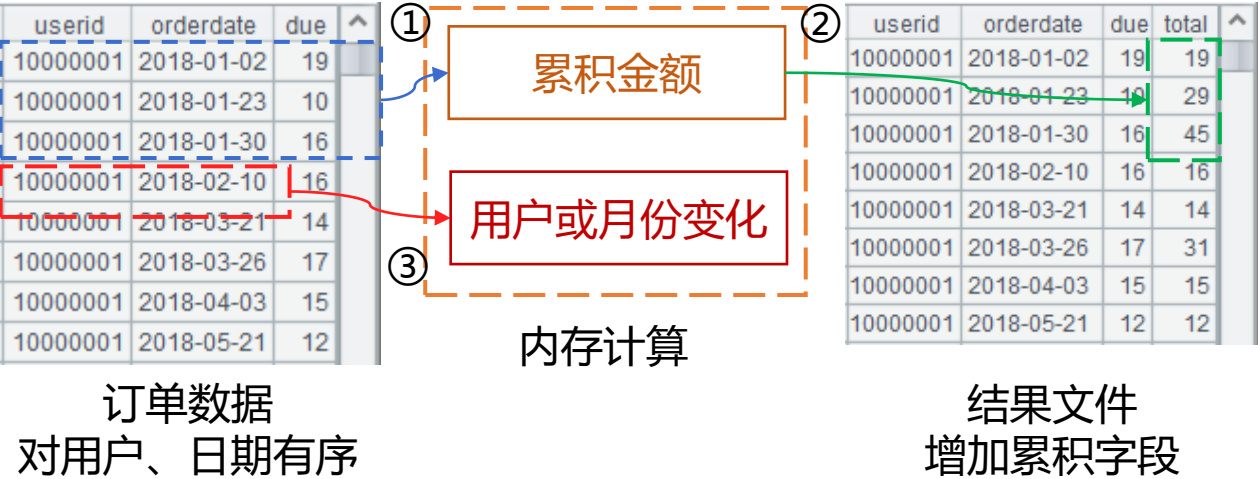
要求：分析日志文件log.txt，统计日志行数最多的单次事件。

提示：日志中以 “—flag--” 为每次事件的分割标示。

知识点-组内迭代

组内计算累积金额

每个用户，每月截止到当日的累积销售额



代码示例

	A	B
1	=order_cursor.derive(iterate(~~+~.due,0;userid,month(orderdate)):total)	/iterate说明: ~~为上一个迭代结果, ~.due为当前行金额; 0是迭代初始值, 分号后是迭代结果清零的条件
2	=A1.fetch(;userid)	

组内迭代

用户、日期有序时，当前行金额和上一行累积值相加，就是当前行累积值。
组内迭代一次遍历即可完成。

组内迭代计算步骤

- ①从文件中分批读入数据到内存，增加累积值计算列，迭代值初始值是0。
- ②用户、月份相同时，逐条计算金额和上一个迭代值之和，即本行累积值，同时赋值给迭代值。输出一批结果到文件。
- ③当用户或者月份变化时，累积值初始值清零。继续计算下一组。

课堂练习p4.5-组内迭代

练习：在p4.5.dfx准备数据的基础上，实现下面的需求。

要求：对按客户号custid和日期排序的数据文件 “orders_cid_odate_sort.ctx” ，按照客户号和日期分组，在组内计算累积金额。

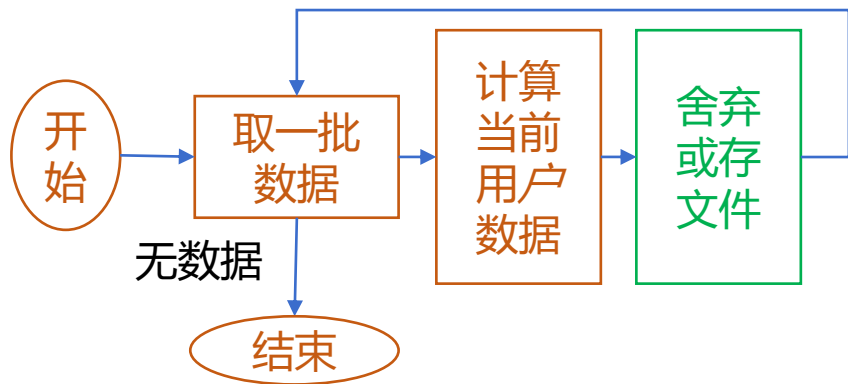
知识点-程序游标

寻找拿铁爱好者-游标继续计算

计算结果不保存为文件，而是形成新游标继续做其他计算

保存为文件继续计算

计算结果保存为文件，也可以继续参与其他计算。但保存文件需要写硬盘，耗时很长，影响性能。

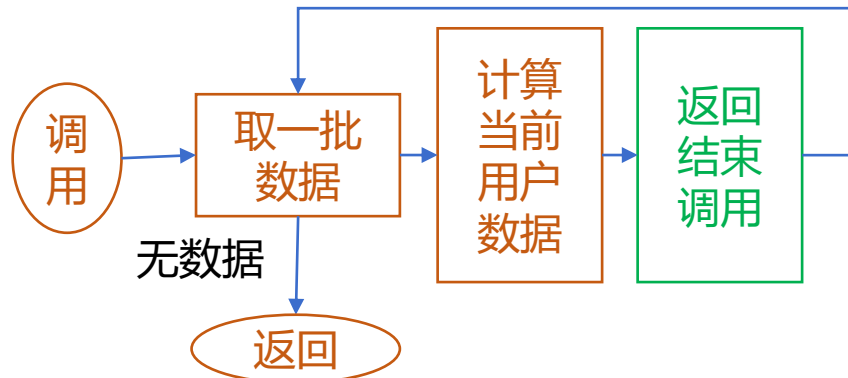


保存为文件流程图

函数或子程序无法满足要求

函数或者子程序被调用时，循环中返回第一批结果后会结束调用了。

即使不结束调用，也是循环把所有的结果连续返回，起不到游标的作用。

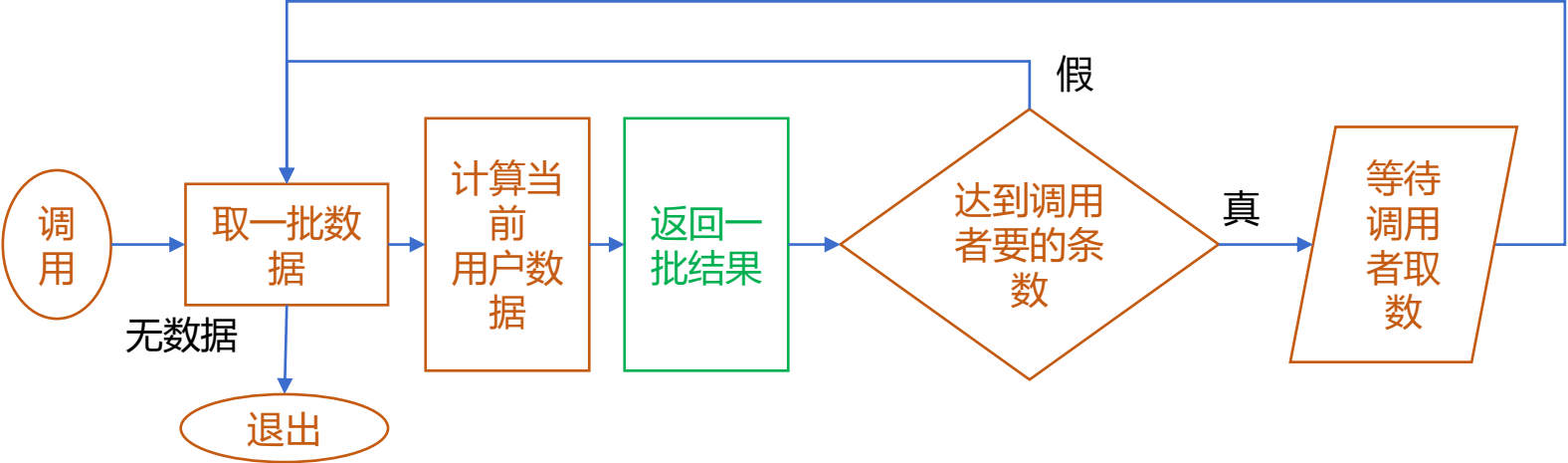


调用子程序流程图

知识点-程序游标

程序游标

计算结果形成新游标继续做其他计算，且返回调用者要求的数据条数。



代码示例

	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 B2
5	=cursor@c(A1).groups(userid;top(-2;orderdate):top2)			

课堂练习p4.6-程序游标

练习：编写p4.6.dfx。

要求：数据文件orders_sort_custid.ctx对custid有序，求在1980和1981年总消费额大于1850000的那些订单。返回游标，只取1000条记录。

提示：可基于p4.3进行改造。

补充阅读 关系数据库的存储过程等计算工具，不提供程序游标功能，要把每个步骤执行结果存成临时表给下一个步骤使用，临时表写入硬盘会影响性能，尤其是数据量较大的时候。

知识点-手工并行

手工并行比函数并行更为灵活

并行统计各月文件中采购牛奶数量大于3的用户的当月全部订单信息

	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@t(B2)

A2分12个线程，每个线程利用有序游标计算当月采购牛奶数据量大于3的用户，并将该用户当月的全部订单信息导出到对应月份的文件（milk_gt3_月份.txt）中。

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

课堂练习p4.7-手工并行

练习：编写p4.7.dfx。

要求：用手工并行的方式，对订单数据组表文件orders.ctx，按月份分组，统计每个月单笔消费金额大于500的订单数。

提示：对组表文件使用fork来完成该计算任务。

说明：手工并行可以用于加快从数据库取数的速度。

第三章 连接

3.1 连接运算理解

3.2 外键表

3.3 主子与同维表

3.4 子查询转换

课前准备1-硬件和主目录

硬件 内存：8G以上 硬盘：空余10G以上 CPU：主频1G以上

主目录 在硬盘适当位置解压缩"join.zip"文件夹，并将集算器主目录
设置成"join"文件夹

设置方法：集算器-菜单-工具-选项-主目录

点[这里](#)下载tbl数据文件，放到“主目录\data\tbls”目录中

课前准备2-生成数据文件

要求

上课前执行并读懂生成数据文件的脚本，回答三个问题：

A，数据文件数据量是多少？有哪些字段？

B，有哪些字段，生成后占用硬盘有多大？

C，有什么特征：有序、列存、行存、分段等等。

执行脚本

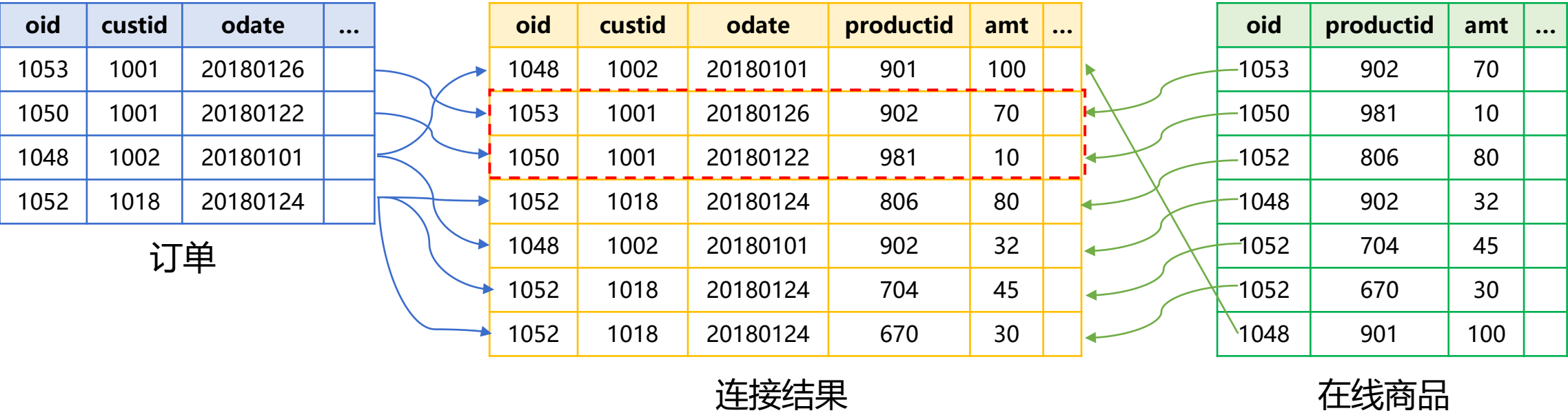
执行脚本"主目录\dfx\create\main.dfx"，生成组表和集文件文件。

第三章 连接

3.1 连接运算理解

知识点-连接运算

计算编号1001客户的订单总额

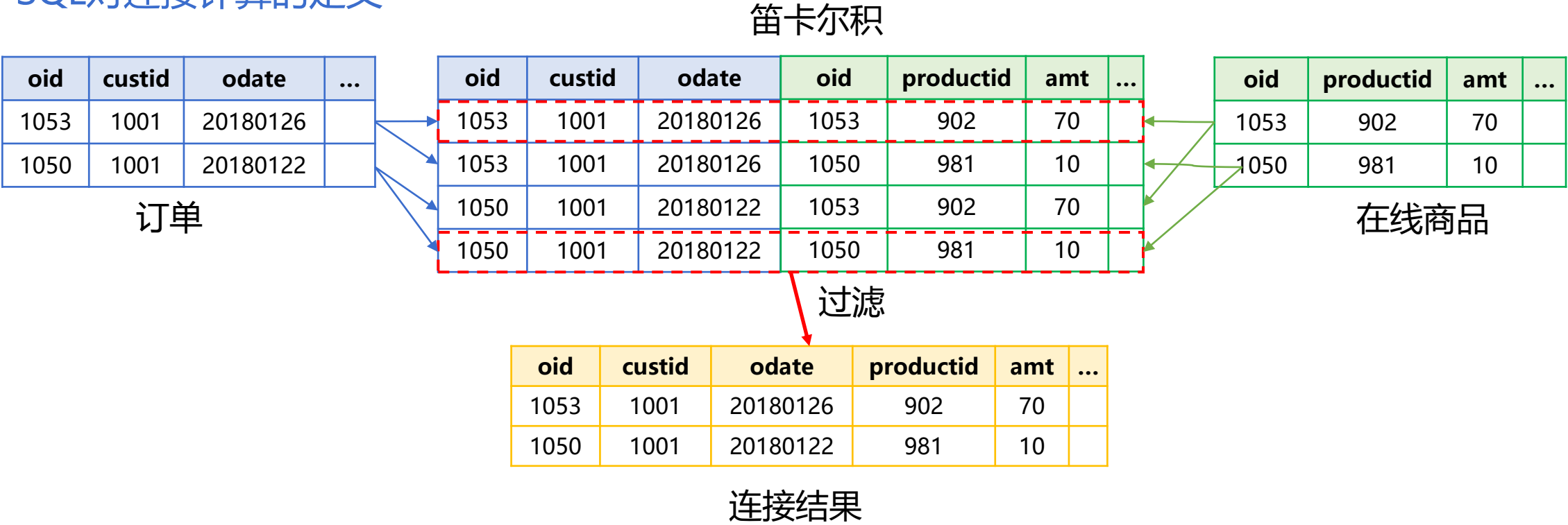


连接运算

订单表中每个订单一条记录，包含编号、客户号、日期等；
在线商品中，包含每个订单中多个产品分别的产品编号、金额等。
必须用订单号将订单表和在线商品连接起来，才能完成需求。

知识点-连接传统计算方法

SQL对连接计算的定义



SQL计算原理（以内存为例）

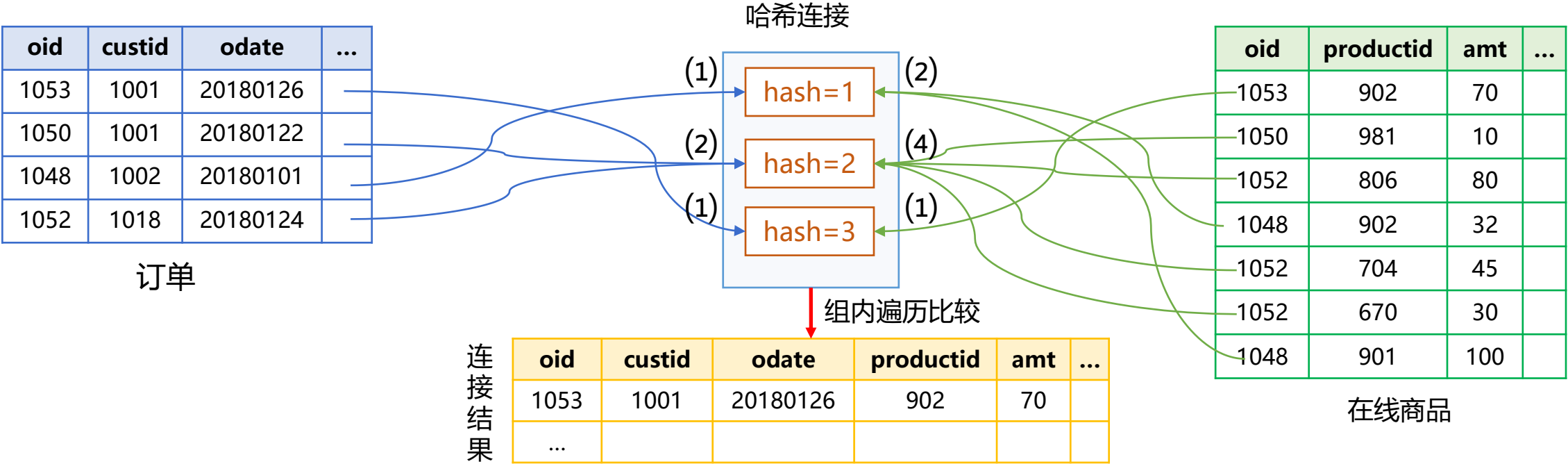
假设订单表、在线商品各有两条记录。先计算笛卡尔积得到四条记录，然后过滤出订单编号相同的两条，得到连接结果。

笛卡尔积法比较计算的次数

笛卡尔积结果过滤，订单编号比较 $2 \times 2 = 4$ 次。若两个表的记录数是N和M，那么就要比较 $N * M$ 次。

知识点-连接传统计算方法

关系数据库哈希法连接（全内存）



哈希法连接（全内存）原理

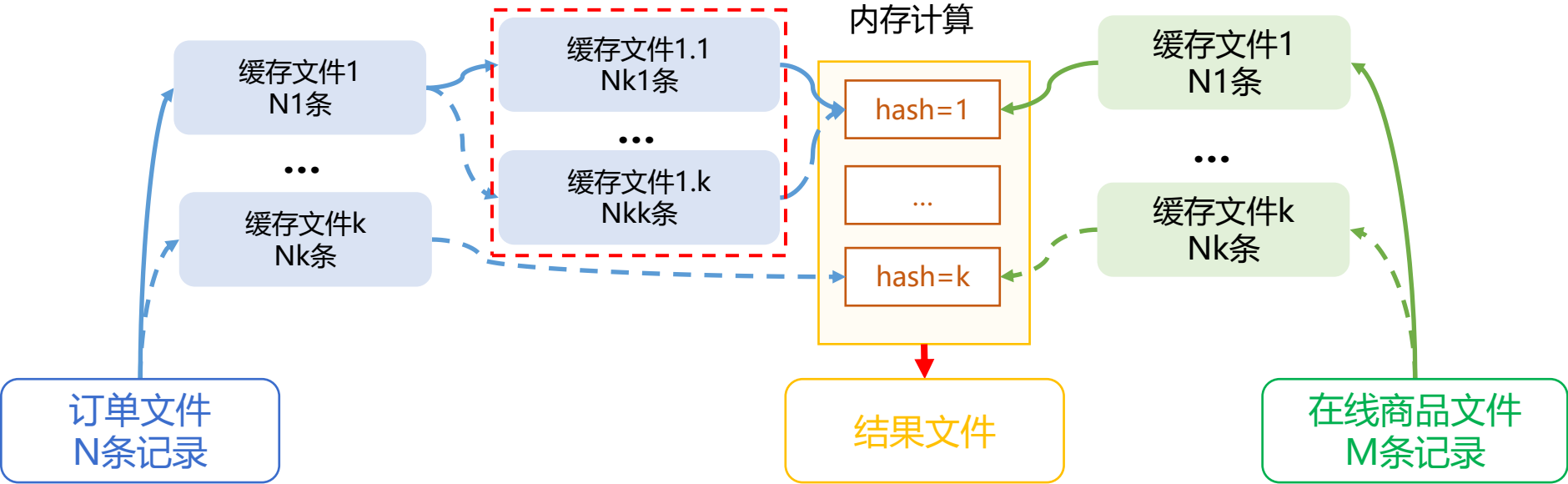
假设订单表四条记录、在线商品表七条记录。按照订单编号哈希值分为三组，再在组内用笛卡尔积法做连接，遍历比较。

哈希法连接（全内存）比较次数

笛卡尔积法需比较 4×7 共28次。哈希法比较次数为 $1 \times 2 + 2 \times 4 + 1 \times 1 = 9$ 次。若哈希值取值范围 k ，则比较次数 $N_1 \times M_1 + N_2 \times M_2 + \dots + N_k \times M_k$ 。笛卡尔积法比较次数 $N \times M = (N_1 + \dots + N_k) \times (M_1 + \dots + M_k)$ ，显然哈希法少很多。

知识点-连接传统计算方法

关系数据库哈希法连接（外存）



哈希法连接（外存）原理

两个表大到内存无法放下时，关系数据库仍采用哈希法分段计算。根据关联字段的哈希值，将数据分成若干段（缓存文件），每个都足够小到能装入内存，再用内存哈希算法连接。

二次哈希分段

缓存文件1太大，无法装入内存，需要二次哈希分段。如图中红框所示，再分为多个缓存文件。

知识点-连接运算剖析

等值连接的常见类型

提高连接性能的方法



同维表

外键表

主子表

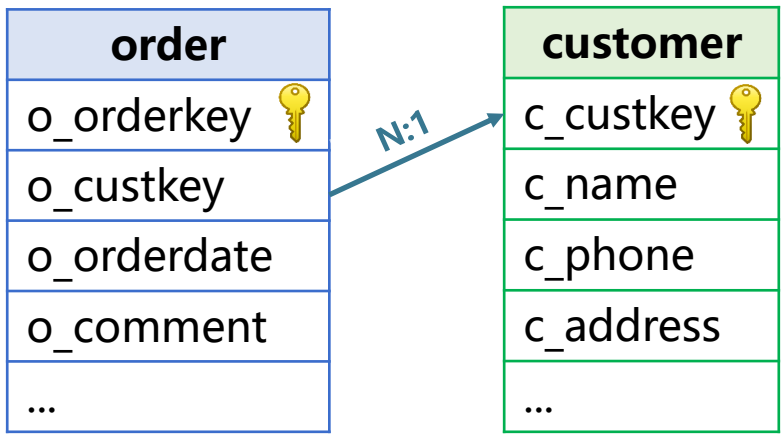
现实应用中绝大多数连接都是等值连接。

外键表、主子表和同维表，涵盖了绝大多数等值连接的情况。

充分利用这三种连接的特征，可获得连接计算更简单的书写形式和更高效的运算性能

知识点-外键表

订单表和客户表



连接关系

外键表是多对一的关系，主要是内连接和左连接,一般不会用到全连接。

事实表和维表

表A的某些字段与表B的主键关联，A 表称为事实表，B 表称为维表。

订单表的客户号与客户表的主键客户号关联。订单表是事实表， 客户表是维表。

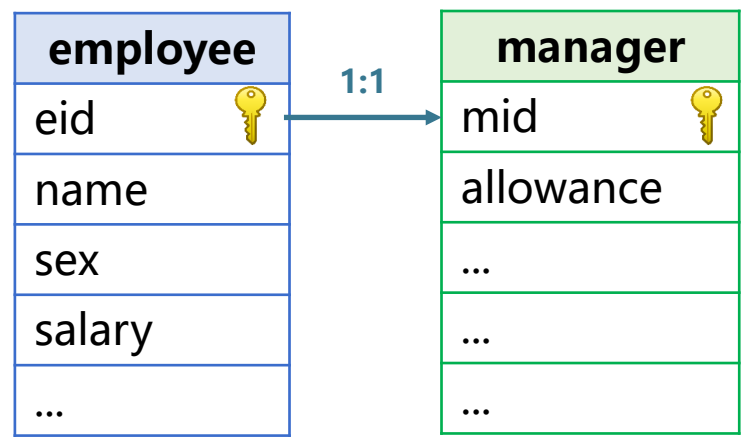
外键表

A表中与B表主键关联的字段称为A指向B的外键， B也称为A的外键表。

订单表的客户号是订单表指向客户表的外键， 客户表是订单表的外键表。

知识点-同维表

员工表和经理表



同维表

表A的主键与表 B 的主键关联，A和B互称为同维表。

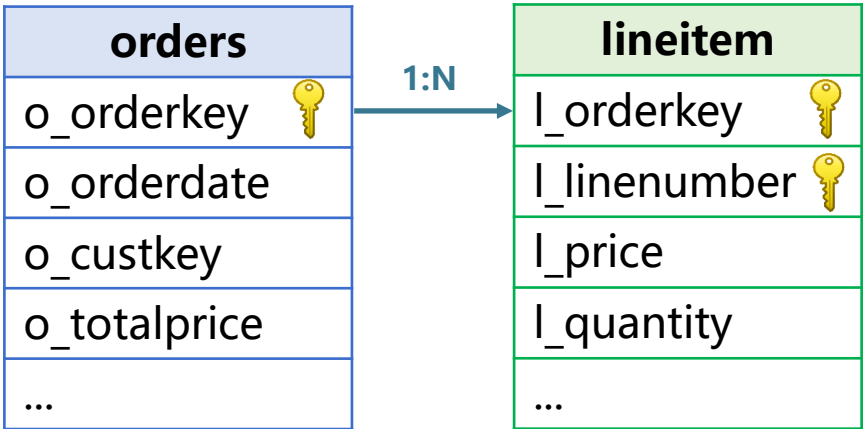
员工表的主键员工号与经理表的主键经理号关联。
员工表和经理表互为同维表。

连接关系

同维表是一一对应的关系，内连接、左连接和全连接的情况都会有。

知识点-主子表

订单表和在线商品表



主表和子表

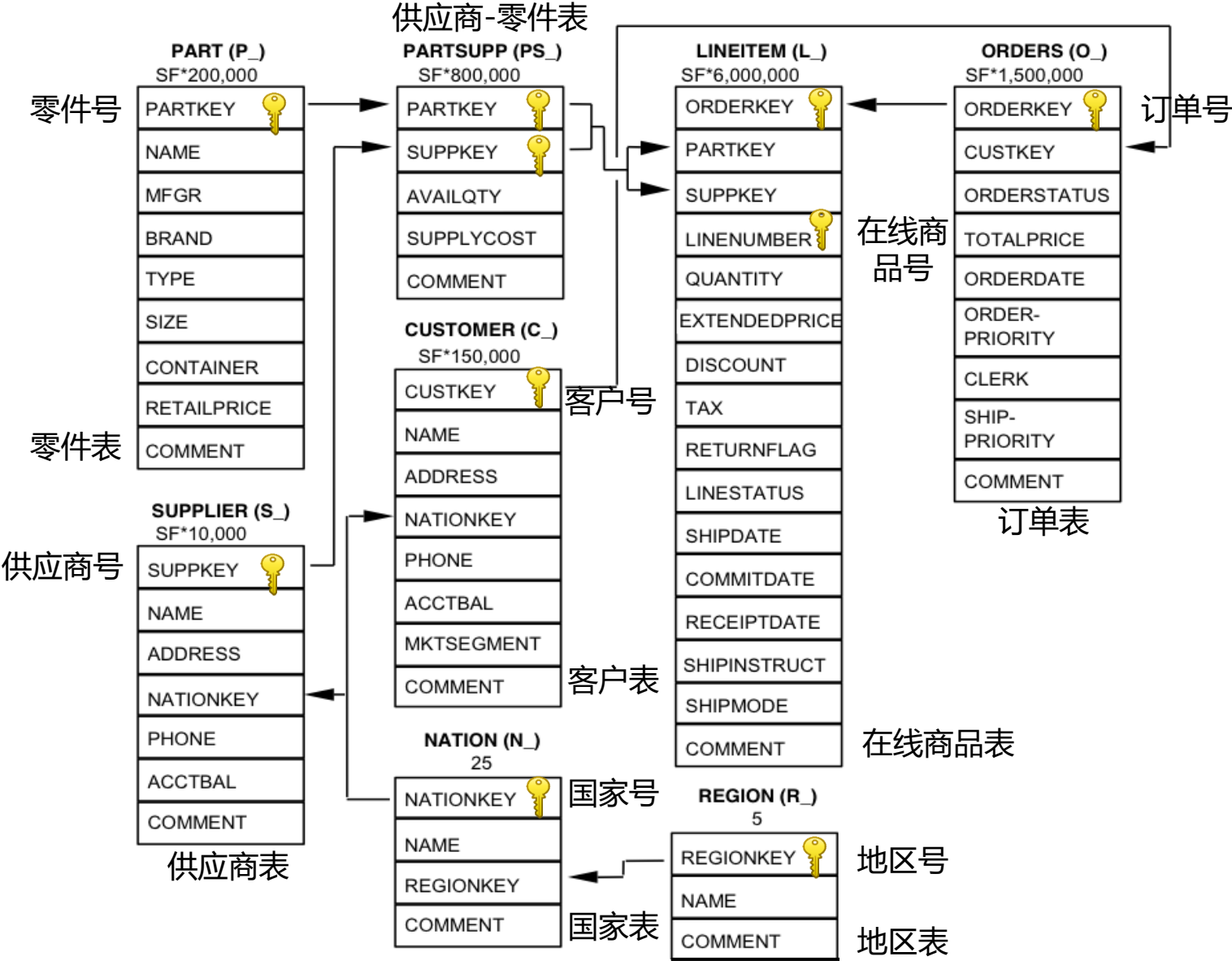
表A的主键与表B的部分主键关联，A称为主表，B称为子表。

订单表的主键订单号与在线商品表的一个主键订单号关联。订单表是主表，在线商品表是子表。

连接关系

主子表是一对多的关系，只有内连接和左连接，不会有全连接。

背景知识-TPC-H表结构



事实表

订单表和在线商品表的数据随时间不断增长，是事实表。

订单表和在线商品表是一对多关系，在线商品表是每个订单包含商品的在线商品。

维表

其他表数据量相对固定，不会随着时间不断增长，是维表。

说明

课堂练习中各表的数据量和图中不一定一致。

也可能会增加一些表来加强练习的效果。

第三章 连接

3.2 外键表

知识点-全内存外键预关联

按照客户城市分组汇总订单金额

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE
1	10262	[RATT,Learn the ...	1996-07-22	14487.0
2	10263	[ERN,Resources ...	1996-07-23	43818.0
3	10264	[FOLKO,Wuzhou tr...	2007-12-18	1101.0
4	10265	[BLONP,The hao,D...	1996-07-25	5528.0
5	10266	[WARTH,Upgrade t...	1996-07-26	7719.0
6	10267	[FRANK,Trust frien...	1996-07-29	20858.0
7	10268	[GROSR,Light far tr...	1996-07-30	19887.0
8	10269	[WHITC,Chair day ...	1996-07-31	456.0

订单表

Index	C_CUSTKEY	C_NAME	C_CITY
1	BLONP	The hao	Dalian
2	CACTU	Wayair freight co. LTD	Dalian
3	CENTC	Three jie industrial	Dalian
4	HUNGC	Hardware mechanical	Dalian
5	MEREP	huake	Dalian
6	ALFKI	Sanchuan industrial ...	Tianjin

客户表

预先关联外键

若两个表都可以装入内存，可在内存中预先遍历订单表，找到客户号在客户表中对应的记录，将客户号替换为指向客户表记录的指针。

外键属性化

预关联之后，对订单表遍历分组时，即可用客户号指针直接访问客户表字段，就像访问订单表的字段（属性）一样。这样就把外键转化为本表的字段（属性）了。

外键属性化优点

预关联只需要计算一次，再对两表做连接计算时就不必做哈希连接，复用预关联结果，大幅度提高性能。
预关联可以在初始化执行或者定时执行。

知识点-全内存外键预关联

按照客户城市分组汇总订单金额

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE	Index	C_CUSTKEY	C_NAME	C_CITY
1	10262	[RATT C, Learn the ...	1996-07-22	14487.0	1	BLONP	The hao	Dalian
2	10263	[ERN SH, Resources ...	1996-07-23	43818.0	2	CACTU	Wayair freight co. LTD	Dalian
3	10264	[FOLKO, Wuzhou tr...	2007-12-18	1101.0	3	CENTC	Three jie industrial	Dalian
4	10265	[BLONP, The hao, D...	1996-07-25	5528.0	4	HUNGC	Hardware mechanical	Dalian
5	10266	[WARTH, Upgrade t...	1996-07-26	7719.0	5	MEREP	huake	Dalian
6	10267	[FRANK, Trust frien...	1996-07-29	20858.0	6	ALFKI	Sanchuan industrial ...	Tianjin
7	10268	[GROSR, Light far tr...	1996-07-30	19887.0				
8	10269	[WHITC, Chair day ...	1996-07-31	456.0				

订单表

客户表

外键属性化的条件

订单表的客户号，必须在客户表中都能找到。如果出现找不到的情况，订单表客户号就会被替换成空值，客户号也会丢失。

代码示例

预计算

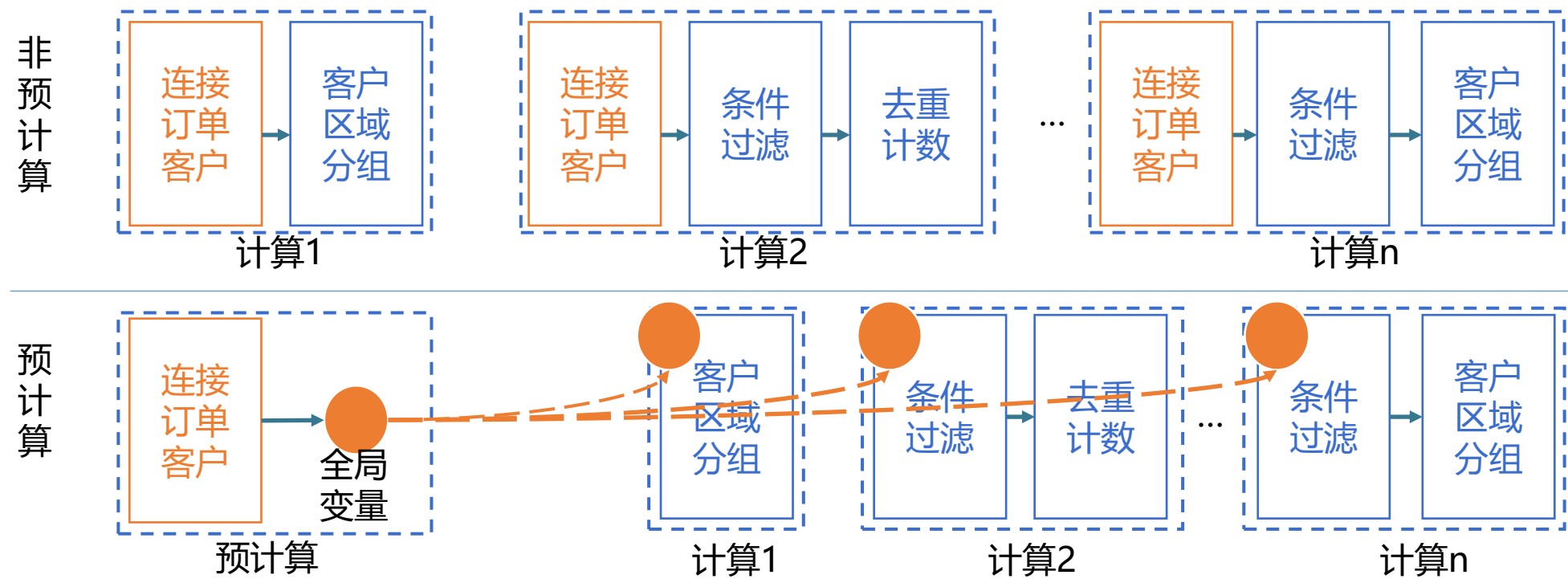
	A	B
1	=ORDERS.switch(O_CUSTKEY,CUSTOMER:C_CUSTKEY)	/外键预关联
2	=env(orderCust,A1)	/存全局变量

分组汇总

	A
1	=orderCust.groups(O_CUSTKEY.C_CITY:C_CITY;sum(O_TOTALPRICE):AMOUT)

知识点-预计算时关联

预计算



什么是预计算

将可以复用结果的计算提取出来，预先计算好结果。省去重复计算的时间，提高性能。

预计算的时间选择

根据实际需求，预计算可以发生在：系统初始化、定时执行、数据变动、ETL过程汇总。

课堂练习p2.1-全内存外键预关联

练习1：执行p2.1.dfx，在线商品表LINEITEM部分数据和零件表PART全内存关联，记录关联汇总时间。

预计算要求：

1、LINEITEM字段L_SHIPDATE，仅1992年1季度数据，与零件表都装入内存。

计算要求：

- 1、PART的P_TYPE的字段长度大于2；
- 2、汇总每月零件订单总收入sum(L_EXTENDEDPRICE*(1-L_DISCOUNT))。

练习2：改写p2.1.dfx，把预关联和分组汇总分开，分别记录时间。

思考：预计算应在什么时候执行，为什么？

补充阅读 关系数据库的连接运算不假定外键指向记录的唯一性，不能使用外键属性化方法，每次关联时都要计算 HASH 值并比对。

	执行时间（毫秒）
关联汇总	
预关联	
预关联后 分组汇总	

知识点-全内存多外键预关联

按照客户区域、雇员姓名分组汇总订单金额

Index	O_ORDERKEY	O_CUSTKEY	O_EMPID	O_ORDERDATE	O_TOTALPRICE
1	10262	[RATTC, Lear...	[8, LiuYinjiu, co...	1996-07-22	14487.0
2	10263	[ERNSH, Reso...	[9, ZhangXue...	1996-07-23	43818.0
3	10264	[FOLKO, Wuzh...	[6, SunLin, sales]	2007-12-18	1101.0
4	10265	[BLONP, The ...]	[2, WangWei, Vi...	1996-07-25	5528.0
5	10266	[WARTH, Upg...	[3, LiFang, sales]	1996-07-26	7719.0
6	10267	[FRANK, Trust...	[4, ZhenJianjie...	1996-07-29	20858.0
7	10268	[GROSR, Light...	[8, LiuYinjiu, co...	1996-07-30	19887.0
8	10269	[WHITC, Chair...	[5, ZhaoJun, m...	1996-07-31	456.0

订单表

Index	C_CUSTKEY	C_NAME	C_REGION
1	BLONP	The hao	North East
2	CACTU	Wayair freight co. LTD	North East
3	CENTC	Three jie industrial	North East
4	HUNGC	Hardware mechanical	North East
5	MEREP	huake	North East
6	ALFKI	Sanchuan industrial co...	North China

客户表

Index	E_ID	E_NAME	E_POSITION
1		ZhangYinling	sales
2		WangWei	Vice President
3		LiFang	sales
4		ZhenJianjie	sales
5		ZhaoJun	manager
6		SunLin	sales

雇员表

预先关联多个外键

订单表通过两个字段关联两个外键表，全内存时可对订单表一次遍历，将两个字段分别替换为两个外键表记录的指针。

代码示例

	A	B
1	=ORDERS.switch(O_CUSTKEY,CUSTOMER:C_CUSTKEY;O_EMPID,EMPLOYEE:E_ID)	/预计算：多外键预关联
2	=A1.groups(O_CUSTKEY.C_REGION:C_REGION,O_EMPID.E_NAME:E_NAME;sum(O_TOTALPRICE):AMOUNT)	/按客户区域和雇员姓名汇总金额

说明：代码省略了预计算、结果存入全局变量，应用时要注意这部分。以下代码不再说明。

课堂练习p2.2-全内存多外键预关联

练习1：执行p2.2.dfx，在线商品表LINEITEM和零件表PART关联、供货商SUPPLIER关联，记录关联汇总时间。

预计算要求：

1、LINEITEM字段L_SHIPDATE，日期是1992年1季度，与维度表都装入内存。

计算要求：

- 1、PART的P_TYPE的字段长度大于2；
- 2、SUPPLIER的S_SUPPKEY大于0；
- 3、汇总每月零件订单总收入sum (L_EXTENDEDPRICE* (1-L_DISCOUNT))

练习2：改写p2.2.dfx，把预关联和分组汇总分开，分别记录时间。

补充阅读 哈希连接算法每次只能解析一个连接，有N个字段连接，要计算N遍。每次连接后需要保持中间结果供下一轮使用，计算过程复杂得多，数据也会被遍历多次。

	执行时间（毫秒）
关联汇总	
预关联	
预关联后 分组汇总	

知识点-全内存复制外键属性

客户地区是中国北部的订单总金额

订单表

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE
1	10262	[RATTC, Learn the ker...	1996-07-22	14487.0
2	10263	(null)	1996-07-23	43818.0
3	10264	[FOLKO, Wuzhou trust...	2007-12-18	1101.0
4	10265	(null)	1996-07-25	5528.0
5	10266	[WARTH, Upgrade the ...	1996-07-26	7719.0
6	10267	[FRANK, Trust friend Jo...	1996-07-29	20858.0
7	10268	[GROSR, Light far trade...	1996-07-30	19887.0
8	10269	[WHITC, Chair day cult...	1996-07-31	456.0

订单表

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE	CUSTOMER_fk
1	10262	RATTC	1996-07-22	14487.0	RATTC
2	10263	ERNSH	1996-07-23	43818.0	(null)
3	10264	FOLKO	2007-12-18	1101.0	FOLKO
4	10265	BLONP	1996-07-25	5528.0	(null)
5	10266	WARTH	1996-07-26	7719.0	WARTH
6	10267	FRANK	1996-07-29	20858.0	FRANK
7	10268	GROSR	1996-07-30	19887.0	GROSR
8	10269	WHITC	1996-07-31	456.0	WHITC

复制

代码示例

	A	B
1	=CUSTOMER.keys(C_CUSTKEY)	/预计算：设置客户表主键
2	=ORDERS.join(O_CUSTKEY,A1,~:CUSTOMER_fk)	/预计算：建立预关联
3	=A2.select(CUSTOMER_fk.C_REGION=="North China").sum(O_TOTALPRICE)	/客户地区是中国北部的总金额

错误做法：直接预关联

实际应用中，订单表有些客户号在客户表中可能找不到记录，也就是左连接的情况。如果直接用客户号关联，找不到的客户号会变为空值，原来的客户号会丢失。

正确做法：预关联结果存为新字段

用订单表的外键（客户号）关联客户表，结果存为一个新字段。这样做，即使因为找不到客户表记录将新字段设置为空值，也不影响原来的客户号。

课堂练习p2.3-全内存复制外键属性

练习1：执行p2.3.dfx，订单表ORDERS中的O_CUSTKEY和客户表CUSTOMER的主键C_CUSTKEY关联。

预计算要求：

1、ORDERS字段O_ORDERDATE，仅1992年1季度数据，与客户表都装入内存。

计算要求：

- 1、CUSTOMER的C_NATIONKEY等于1并且O_TOTALPRICE小于5000；
- 2、满足条件的记录，O_TOTALPRICE求和。

练习2：改写p2.3.dfx。

要求：假设订单表中，有些客户信息在客户表中找不到。

提示：用"复制外键属性"的办法来进行数据关联。

知识点-全内存复制多个外键

客户和雇员两个外键

Index	O_ORDERKEY	O_CUSTKEY	O_EMPID	O_ORDERDA...	O_TOTALPRI...
1	10262	[RATTC, Lea...	[8, LiuYinjiu...	1996-07-22	14487.0
2	10263	(null)	[9, ZhangXu...	1996-07-23	43818.0
3	10264	[FOLKO, Wuz...	[6, SunLin, sal...	2007-12-18	1101.0
4	10265	(null)	[2, WangWei,...	1996-07-25	5528.0
5	10266	[WARTH, Up...	[3, LiFang, sal...	1996-07-26	7719.0
6	10267	[FRANK, Tru...	[4, ZhenJianji...	1996-07-29	20858.0
7	10268	[GROSR, Ligh...	[8, LiuYinjiu...	1996-07-30	19887.0
8	10269	[WHITC, Chal...	(null)	1996-07-31	456.0

Index	O_ORDER...	O_CUSTKEY	O_EMPID	O_ORDER...	O_TOTAL...	CUSTOMER...	EMPLOYEE...
1	10262	RATTC	8	1996-07...	14487.0	RATTC	8
2	10263	ERNSH	9	1996-07...	43818.0	(null)	9
3	10264	FOLKO	6	2007-12...	1101.0	FOLKO	6
4	10265	BLONP	2	1996-07...	5528.0	(null)	2
5	10266	WARTH	3	1996-07...	7719.0	WARTH	3
6	10267	FRANK	4	1996-07...	20858.0	FRANK	4
7	10268	GROSR	8	1996-07...	19887.0	GROSR	8
8	10269	WHITC	5	1996-07...	456.0	WHITC	(null)

复制

代码示例

	A	B
1	=CUSTOMER.keys(C_CUSTKEY),EMPLOYEE.keys(E_ID)	/预计算：设置主键客户ID、雇员ID
2	=ORDERS.join(O_CUSTKEY,CUSTOMER,~:CUSTOMER_fk;O_EMPID,EMPLOYEE,~:EMPLOYEE_fk)	/预计算：建立预关联
3	=A2.select(CUSTOMER_fk.C_REGION=="North China" && EMPLOYEE_fk.E_POSITION=="sales").sum(O_TOTALPRICE)	/汇总客户地区是中国北方，雇员职位是销售代表的订单金额

复制多个外键

用订单表的外键（客户号）关联客户表，结果存为一个新字段；外键（客户号）关联雇员表，结果存为第二个新字段。

同样，一次遍历即可复制多个外键。

课堂练习p2.4-全内存复制多个外键

练习1： 打开p2.4.dfx，LINEITEM表中的字段（L_PARTKEY、L_SUPPKEY）和PART表、SUPPLIER表的主键（P_PARTKEY、S_SUPPKEY）关联。

预计算要求：

1、LINEITEM的L_SHIPDATE字段，日期是1992年1季度，与维表都装入内存。

计算要求：

1、PART的P_SIZE等于7

2、SUPPLIER的S_NATIONKEY等于5；

3、汇总订单总收入（sum(L_EXTENDEDPRICE)）

练习2： 改写p2.4.dfx。

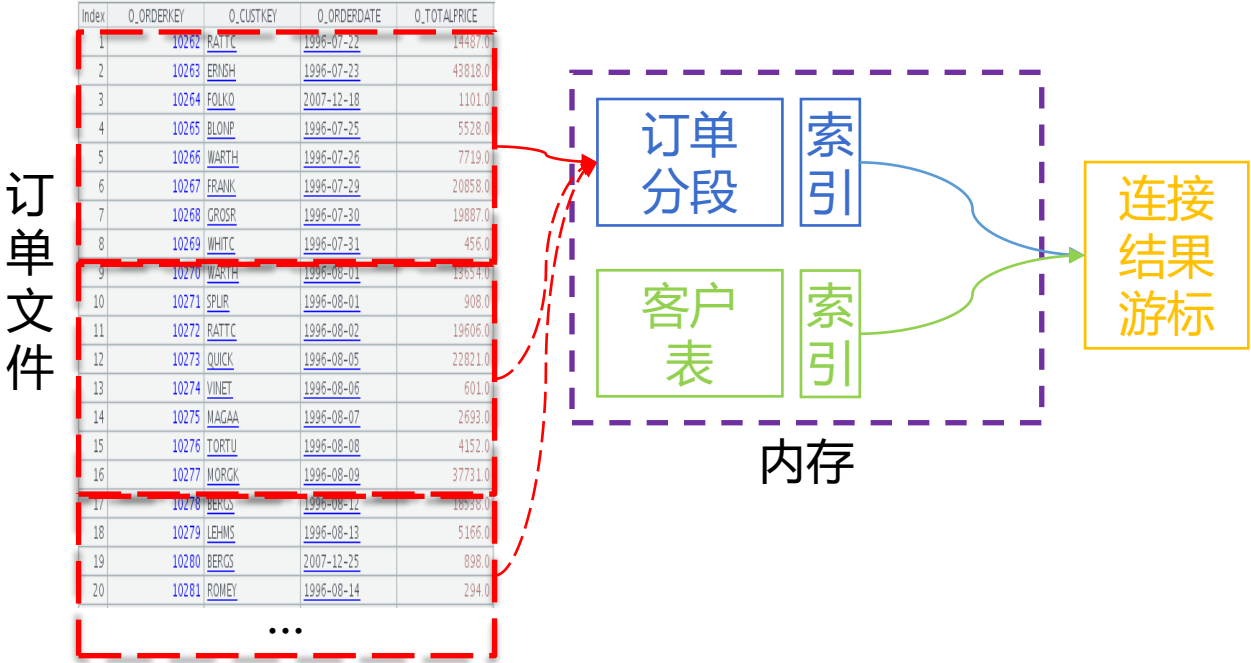
要求： 假设在线上商品表中，有些数据在PART表和SUPPLIER表中找不到。

提示： 用"复制多个外键"的办法来进行数据关联。

知识点-仅维表全内存-临时指向

按客户所在城市汇总订单金额

订单数据量大，文件存放；客户数据量小，全内存



临时指向计算步骤

先将客户表读入内存，建立索引。再分批读入订单文件。

读入一批，在内存中建立索引，与客户表做全内存外键属性化，结果输出到连接结果游标，继续下一步计算。

临时指向计算性能分析

临时指向要计算哈希索引并比对是否同值，性能不如全内存外键属性化。

好处包括：

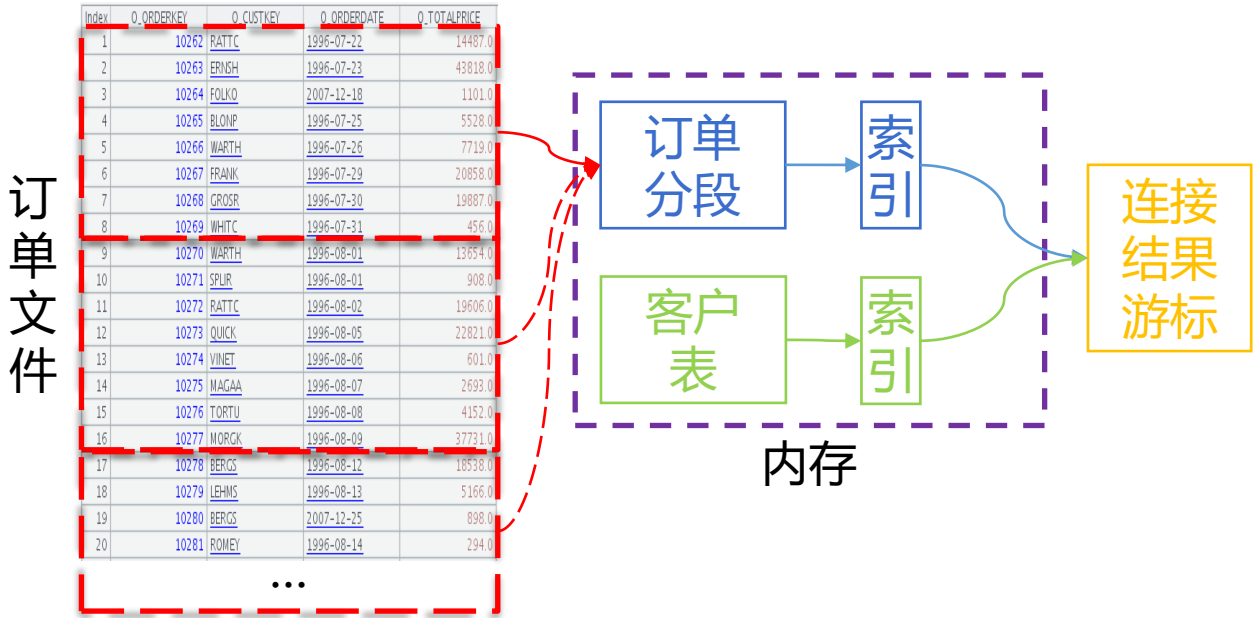
- 客户表（维表）的索引可以重复使用；
- 一次遍历关联所有订单数据（事实表）；
- 易于分段并行。

因此，相比哈希连接，临时指向性能优势还是很大。

知识点-仅维表全内存-临时指向

按客户所在城市汇总订单金额

订单数据量大，文件存放；客户数据量小，全内存



临时指向保持原数据顺序

订单文件是顺序分批读入的，所以输出结果游标保持原文件顺序，后续计算如果需要有序，可避免大排序。

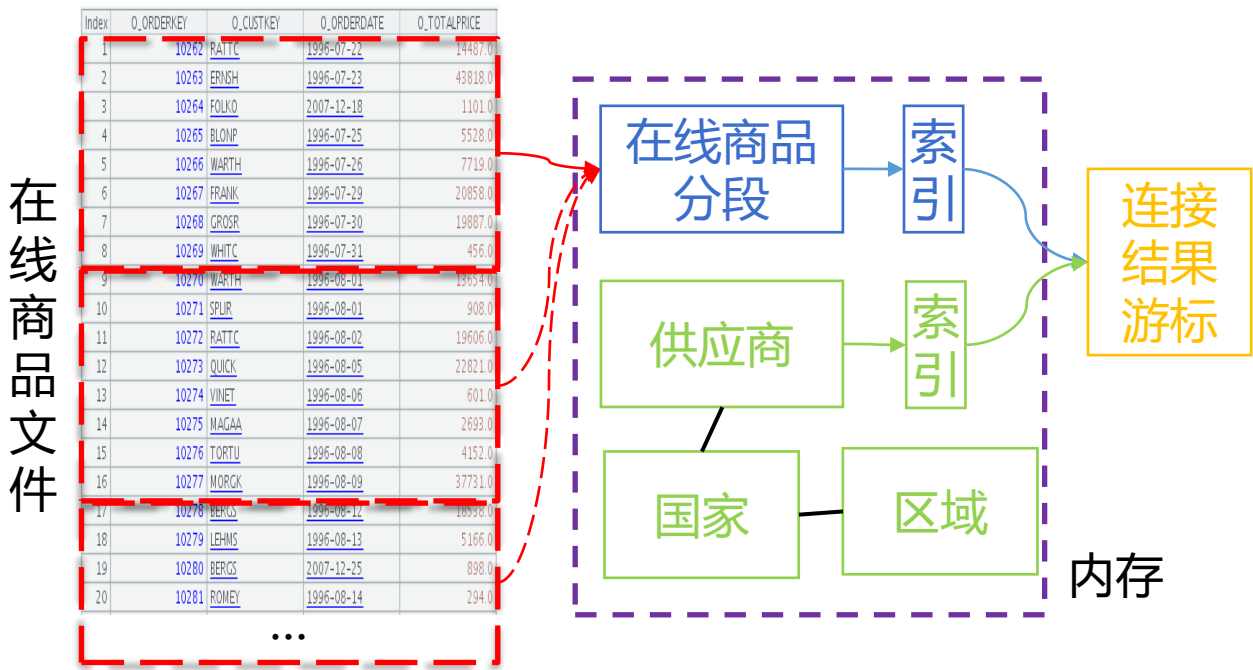
哈希连接很难保持原顺序。

代码示例		A	B
1		=file("ORDERS.btx").cursor@b()	/建立订单表记录游标，分批读入数据
2		=A1.switch(O_CUSTKEY,CUSTOMER:C_CUSTKEY)	/将订单表的客户号属性化为客户表记录的引用
3		=A1.groups(O_CUSTKEY.C_CITY:CITY;sum(O_TOTALPRICE):A MOUNT)	/按照客户所在城市汇总订单的金额

知识点-仅维表全内存-临时指向

按供应商区域名分组统计在线商品记录个数

在线商品数据量大，文件存放；维表数据量小，全内存



代码示例

	A	B
1	>supplier.switch(S_NATIONKEY,nation), nation.switch(N_REGIONKEY,region)	/三个维表预关联
2	=lineitemCursor.switch(L_SUPPKEY,supplier:S_SUPPKEY)	
3	=A1.groups(L_SUPPKEY.S_NATIONKEY.N_REGIONKEY.R_NAME:NAME;count(L_LINENUMBER):C)	

多个维表预关联后临时指向

供应商关联国家，国家关联区域等等。在内存中，多个维表预先关联好，分批读入在线商品表时，再用临时指向的手段完成计算。

这样做，维表数量增加，对临时指向的关联计算性能影响很小。

课堂练习p2.5-仅维表全内存-临时指向

练习1：打开p2.5.dfx，在线商品表部分数据（LINEITEM）和零件表（PART）关联，全部装入内存。

要求：

- 1、LINEITEM的L_SHIPDATE字段，仅1992年1季度数据；
- 2、PART的P_TYPE的字段长度大于2；
- 3、汇总每月零件订单总收入sum（L_EXTENDEDPRICE*（1-L_DISCOUNT））。

练习2：改写p2.5.dfx，对LINEITEM表不再限时时间，而是计算全部数据。

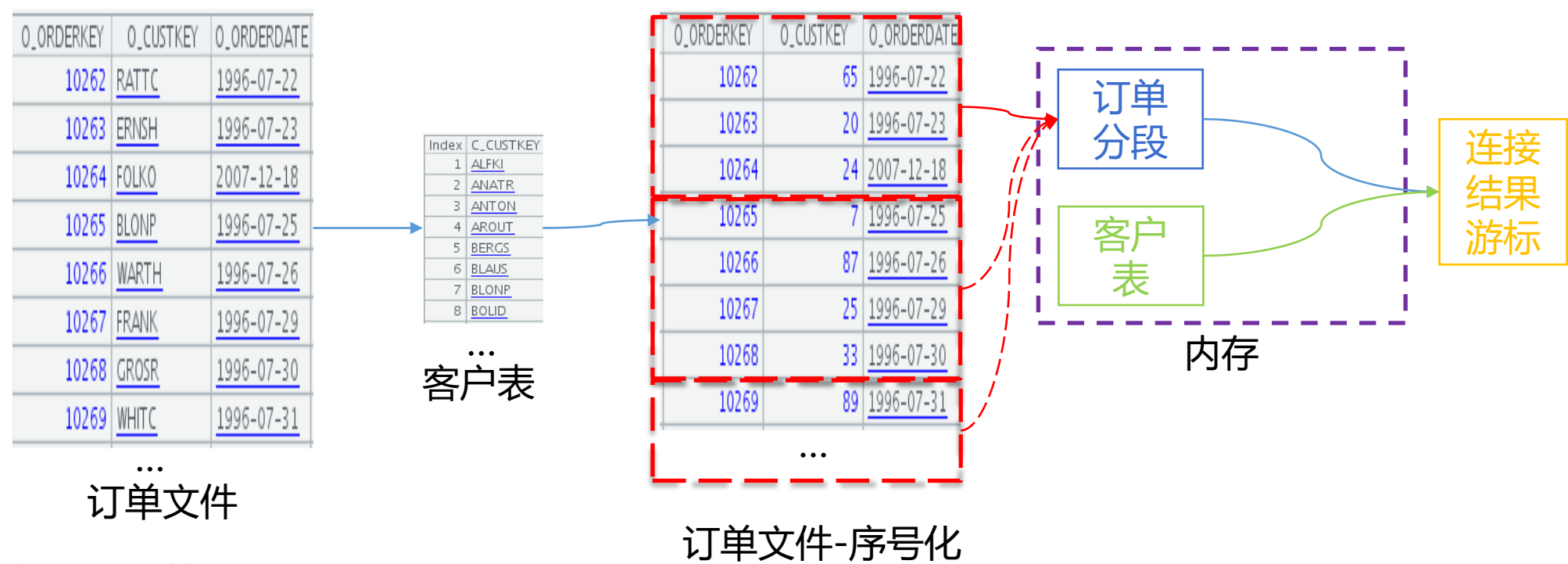
提示：全量的LINEITEM表为大事实表，内存无法直接装下，用游标方式计算。PART表为维表，数据量小，可以全内存。

补充阅读 关系数据库用在关联表个数很多的时候，会出现数据库引擎自动优化困难的问题，导致性能下降很严重。

知识点-仅维表全内存-外键序号化

按客户所在城市汇总订单金额

订单数据量大，文件存放；客户数据量小，全内存



序号化计算步骤

先将订单文件中的客户号改写为客户表中对应客户号的位置，完成订单文件的序号化。再用序号化订单文件和全内存的客户表连接，用位置直接定位维表记录。

知识点-仅维表全内存-外键序号化

序号化优点

用序号化订单文件和全内存的客户表连接时，可用位置直接定位维表记录，不需要生成索引、索引查找和比对同值。

序号化的本质

外键序号化本质上相当于在外存实现外键属性化，也可以像在内存一样，复用序号化的结果。

序号化的注意事项

维表数据变动较大时，需同步整理事实表的外键字段，否则可能对应错位。不过，维表不太会频繁变动，而且大多数是追加和修改，而非删除，需要重整事实表的情况并不多。

代码示例

	A	B
1	=file("ORDERS.btx").cursor@b()	/建立订单表记录游标，分批读入
2	=A1.switch(O_CUSTKEY,CUSTOMER:#)	/利用序号建立外键关联
3	=A1.groups(O_CUSTKEY.C_NAME:C_NAME;sum(O_TOTALPRICE):AMOUNT)	/分组汇总

说明：如果客户号不是序号（自然数），而是字符串，需要预计算成序号。订单表也要预计算，将外键客户号转换为对应的序号。

课堂练习p2.6-仅维表全内存-外键序号化

练习1：执行p2.6.dfx，LINEITEM表和零件表
PART、供货商表SUPPLIER关联。记录非序号话执行时间。
预计算要求：零件表和供货商表预先读入内存。

计算要求：

- 1、PART的P_SIZE大于0;
- 2、SUPPLIER的S_ACCTBAL小于999999;
- 3、按年分组统计订单总收入sum (L_EXTENDEDPRICE* (1-L_DISCOUNT)) 。

练习2： p2.6_XH.dfx完成了零件表的序号化，要求完善p2.6_XH.dfx，补全供货商表的序号化，并执行。

练习3： 改写p2.6.dfx，用外键序号化方法。记录序号化执行时间并比较。

补充阅读 关系数据库采用无序集合的概念，即使事先把外键序号化了，数据库也难以利用这个特点，仍然会去计算哈希值并且等值比对。

	执行时间（毫秒）
非序号化	
序号化	

知识点-仅维表全内存-排号键连接

按纳税人所在区域分组统计税额

id	idcard	tax	...
1	11010519730609816	1500	
2	54211019840812023	560	
3	64072219630218415	300	
...			

税收数据文件

idcard	name	area	...
11010519730609816	张三	23	
54211019840812023	李四	44	
64072219630218415	王五	53	
.....			

纳税人-内存

外键序号化存在的问题

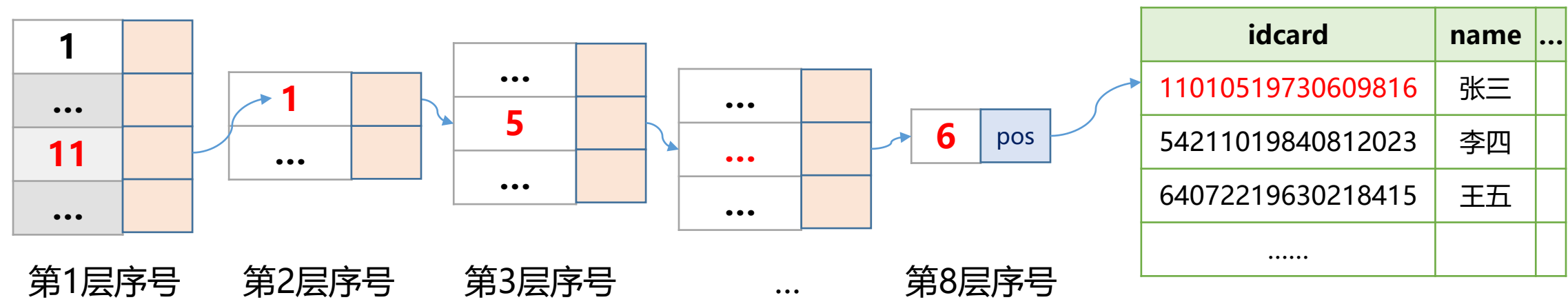
身份证18位，去掉校验位还有17位。
身份证号是不连续的，直接序号定位需要补齐成连续的序号。也就是说，需要10的17次方个Long型序号，占用内存过多。

外键多层序号化

将身份证转换为多层序号，比如第1、2位省级编码为第一层序号；3、4位市级编码为第二层序号等等。
身份证号很稀疏，很多序号没有下级节点，因此树形的多层序号占用的内存比单层序号少很多。

知识点-仅维表全内存-排号键连接

用多层序号关联身份证号11010519730609816



代码示例-仅维表全内存-排号键连接

排号键索引实现多层序号定位的步骤



代码示例

	A
1	=k(11,1,5...)
2	/转换为排号键

	A	
1	=taxpayer.keys@s(idcard)	/预计算：排号键主键
2	=tax_cursor.swtich(cardid,A1:cardid)	/预计算：预关联
3	=A2.groups(cardid.area:area;sum(tax):total)	/分组统计

排号键索引的一种物理实现

在物理实现上，集算器排号键采用Long型（8个字节）实现最多8层序号。利用排号键建立的树形索引，实现多层序号定位。

相比集算器排号键索引，用Java的多层嵌套数组也可以实现多层序号，但复杂数组对象成本很高，会抵消序号定位的性能提升。

课堂练习p2.7-仅维表全内存-排号键

	执行时间（毫秒）
字符串	
排号键	

练习1： 执行p2.7.dfx，税收记录taxNormal.btx和纳税人cardNormal.btx，通过字符串字段身份证cardNo外键关联，并记录字符串方式执行时间。
要求： 预计算全内存加载纳税人。按纳税人所在区域分组统计总税额。

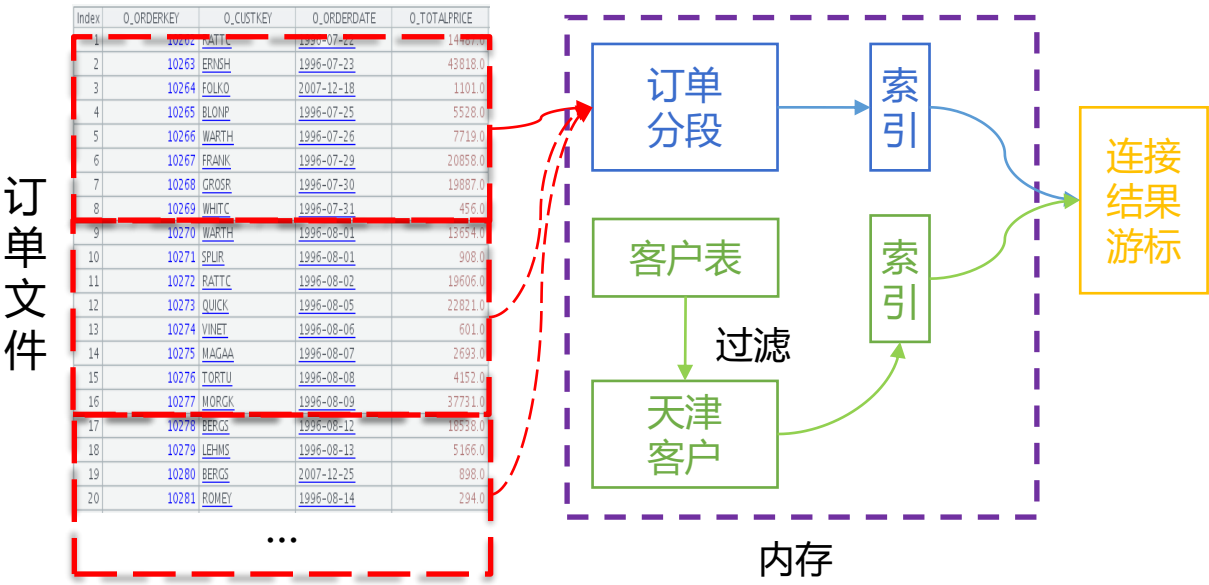
练习2： p2.7Cardk.dfx完成了纳税人表身份证字段的排号键转换，仿照这个dfx写出p2.7Taxk.dfx，生成taxK.btx，用排号键代替taxNormal.btx中的字符串字段身份证号。

练习3： 改写p2.7.dfx，用taxK.btx和cardK.btx，通过排号键字段cardNo外键关联，记录排号键方式执行时间。
要求： 预计算全内存加载纳税人。按纳税人所在区域分组统计总税额。
提示： key@s, @s创建排号索引。

知识点-维表过滤-利用已建索引

按订单日期汇总"天津"客户的订单金额

订单数据量大，文件存放；客户数据量小，全内存



复用维表的索引

维表客户表装入内存并建好索引，如果针对过滤后的维表做关联，需要重建维表索引，维表较大时建也很耗时。

可复用维表已有索引做关联，不用重新计算哈希值。

复用维表索引的注意事项

过滤后的维表记录顺序有可能会被打乱。

代码示例

	A	B
1	=CUSTOMER.select@i(C_CITY=="Tianjin")	/过滤客户表，并复用有索引
2	=orderCursor.switch@i(O_CUSTKEY,A1:C_CUSTKEY)	/外键属性化并删除关联不上的记录
3	=A1.groups(O_ORDERDATE;sum(O_TOTALPRICE):O_TOTALPRICE)	/按照订购日期汇总订单的销售额

知识点-维表过滤-对位序列

汇总“州编号为23”客户的订单总金额

订单数据量大，文件存放；客户数据量小，全内存

Index	C_CUSTKEY	C_NATIONKEY
10	10	5
11	11	23
12	12	13
13	13	3
14	14	1
15	15	23
16	16	10
17	17	2
18	18	6
19	19	18

客户表

Index	Member
10	false
11	true
12	false
13	false
14	false
15	true
16	false
17	false
18	false
19	false

对位序列

生成对位序列

先将客户数据和订单中的客户号序号化。

维表客户装入内存后，再生成一个等长的、一一对应的序列。

客户表中，州编号为23的记录，序列中的对应成员设置为true，如图红框所示。其他成员设置成false。

这个序列中值为true的成员编号，对应客户数据中的序号，就是符合条件的客户记录，称为对位序列。

代码示例

	A	B
1	=CUSTOMER.(C_NATIONKEY==23)	/按照过滤条件生成客户的对位序号

知识点-维表过滤-对位序列

汇总“州编号为23” 客户的订单总金额

订单数据量大，文件存放；客户数据量小，全内存

Index	O_ORDERDATE	O_CUSTKEY	O_TOTALPRICE
54	1994-03-15	13	161835.66
55	1996-12-07	11	251492.1
56	1994-11-03	8	184908.92
57	1996-04-17	11	83996.01
58	1996-01-03	4	46227.94
59	1993-06-27	5	324835.83
60	1994-03-26	13	159823.91
61	1996-08-11	4	8332.33
62	1995-10-01	10	52546.37
63	1998-02-18	7	31698.35
64	1993-02-01	13	78761.73
65	1994-06-10	4	40947.40

订单文件

Index	Member
10	false
11	true
12	false
13	false
14	false
15	true
16	false
17	false
18	false
19	false
20	false

对位序列

用对位序列计算维表过滤

在内存中分段遍历订单数据，用客户序号，在对位序列中取对应的成员。

例如：客户序号是13，对位序列第13个成员的值是false，则本条记录不满足条件。

再如：客户序号是11，对位序列第11个成员的值是true，则本条订单记录满足条件。

对位序列性能优势

过滤计算只在维表上进行。事实表只需一次遍历，省掉了关联计算，不用建索引、也不用计算哈希值。

代码示例

	A	B
1	=CUSTOMER.(C_NATIONKEY==23)	/按照过滤条件生成客户的对位序号
2	=ordersCursor().select(A1(C_NATIONKEY)).total(sum(O_TOTALPRICE))	/用对位序列完成维表过滤

课堂练习p2.8-维表过滤

练习1：执行p2.8.dfx，该dfx为在线商品表LINEITEM和零件表PART关联，记录关联后过滤的执行时间。

要求：

- 1、PART的P_SIZE等于4；
- 2、PART的P_NAME以"bis"开头；
- 3、求LINEITEM的L_EXTENDEDPRICE字段的合计值sum(L_EXTENDEDPRICE)

	执行时间（毫秒）
关联后过滤	
复用索引	
序号对位	

练习2：改写p2.8.dfx，改为先过滤维表后关联，记录并比较执行时间。

提示：过滤维表时用"复用维表索引"。

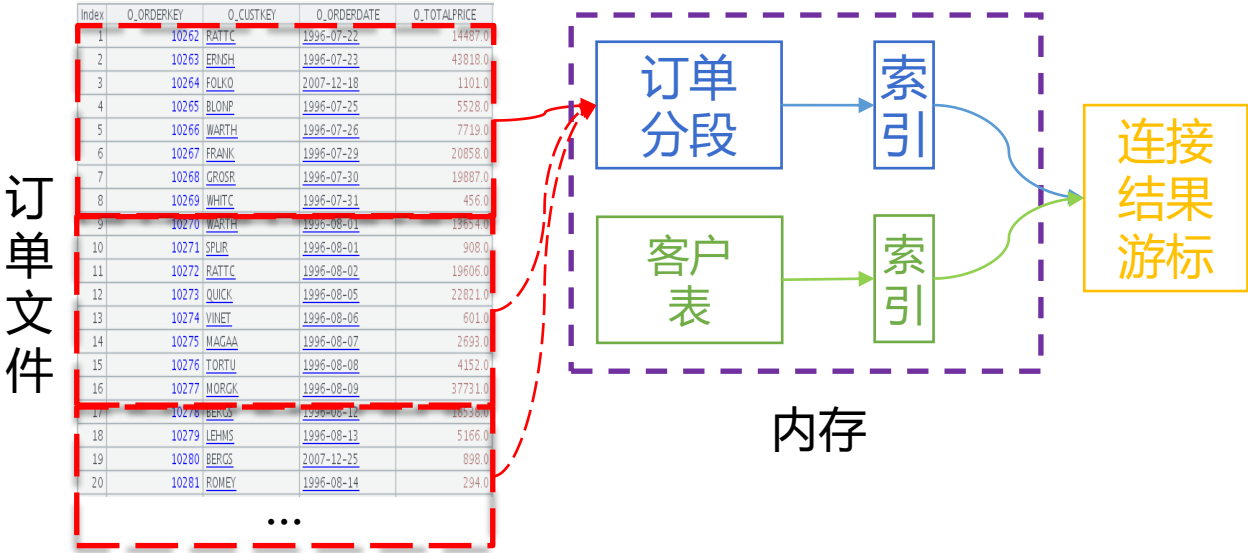
练习3：改写p2.8.dfx，改为序号对位方式，记录并比较执行时间。

提示：A.()函数和select()函数。

知识点-内连接-维表字段仅用于过滤

去掉客户表没有记录的订单

订单数据量大，文件存放；客户数据量小，全内存



代码示例

	A	B
1	=file("ORDERS.ctx").open().cursor()	/建立订单表记录游标，分批读入数据
2	=A1.join@i(O_CUSTKEY,CUSTOMER:C_CUSTKEY)	/关联订单表和顾客表，丢弃关联不上的记录
3	=A2.groups(O_ORDERDATE;sum(O_TOTALPRICE):O_TOTALPRICE)	/按照订购日期汇总订单的销售额

维表字段仅用于过滤

事实表和维表做内连接，维表仅用于过滤，在后续的计算中不使用维表的字段。

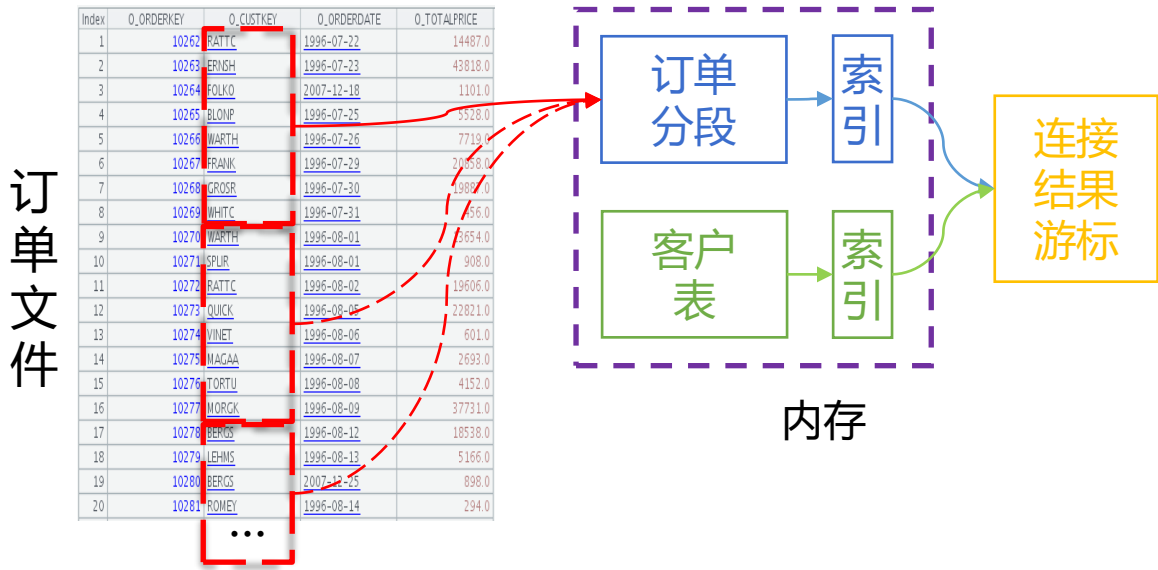
读入内存后过滤

分批读入事实表数据，和维表关联，丢弃关联不上的记录。

知识点-内连接-游标读取时关联过滤

去掉客户表没有记录的订单

订单数据量大，文件存放；客户数据量小，全内存



代码示例

	A	B
1	<code>=orderfile.cursor(;CUSTOMER.find(C_CUSTKEY))</code>	/读入数据时先读入客户号，关联客户表，能关联上则继续读入其它字段，反之则丢弃当前记录
2	<code>=A1.groups(O_ORDERDATE;sum(O_TOTALPRICE):O_TOTALPRICE)</code>	/按照订购日期汇总订单的销售额

维表字段仅用于过滤

事实表和维表做内连接，维表仅用于过滤，在后续的计算中不使用维表的字段。

读入内存时过滤

游标读入时，先仅仅读入关联字段，关联并过滤。关联不上则舍弃本条记录，不再读出其它字段。

过滤掉较多记录时可以明显减少硬盘读取数据量，从而提升性能。

课堂练习p2.9-游标读取时关联过滤

练习1：执行p2.9.dfx，订单表ORDERS，通过客户号O_CUSTKEY和主键为C_CUSTKEY的客户表CUSTOMER关联，客户表全内存。记录全字段执行时间。

要求：

- 1、客户表过滤条件C_NATIONKEY==15，丢弃关联不上的记录；
- 2、按订单表订购日期O_ORDERDATE汇总日销售额sum(O_TOTALPRICE)。

提示：用join@i，对于匹配不上的外键删除整条记录。

	执行时间（毫秒）
全字段	
仅关联字段	

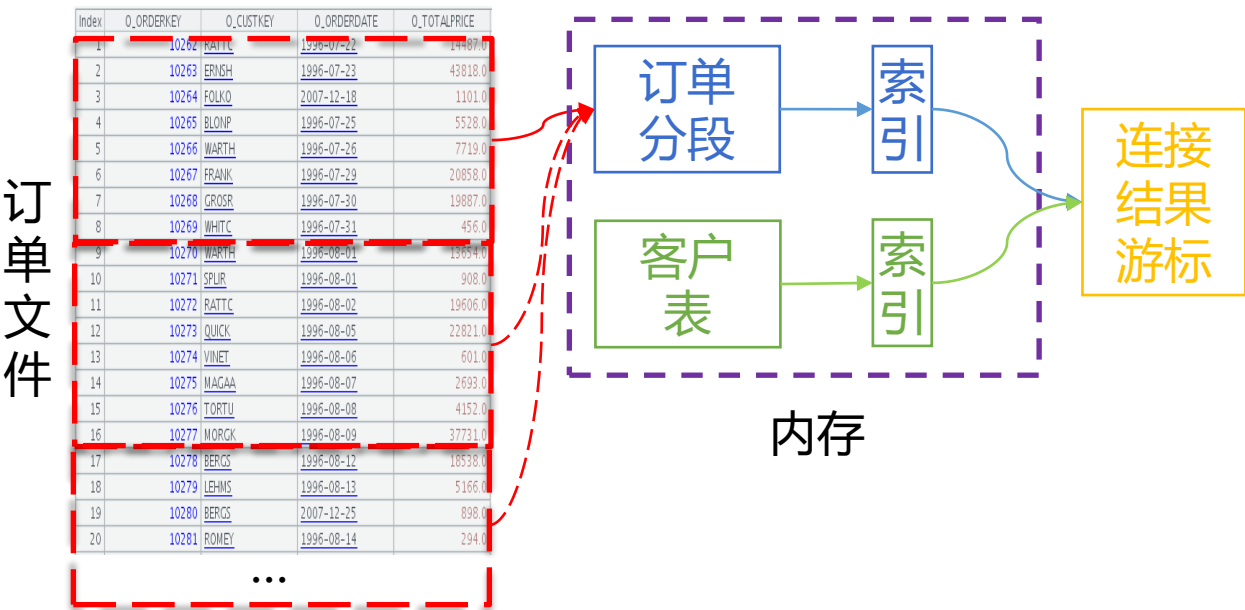
练习2：改写p2.9.dfx，用游标读取时关联过滤方法，记录仅关联字段执行时间。

提示： 用T.cursor(x:C,...;wi...;k:n)，在过滤条件wi中，用Ti.find(K)过滤。

知识点-内连接-关联过滤的同时属性化

去掉客户表没有记录的订单

订单数据量大，文件存放；客户数据量小，全内存



维表字段过滤后还要参与计算

事实表和维表做内连接，维表不仅用于过滤，在后续的计算中还要使用维表的字段。

读入内存后过滤

分批读入事实表数据，和维表关联，丢弃关联不上的记录。

代码示例

	A	B
1	=orderCursor.switch@i(O_CUSTKEY,CUSTOMER:C_CUSTKEY)	/分批读取数据，将订单表中的客户号和客户表做外键属性化，并删除关联不上的记录
2	=A1.groups(O_CUSTKEY.C_NAME:C_NAME;sum(O_TOTALPRICE):O_TOTALPRICE)	/按照客户公司名称汇总订单的销售额

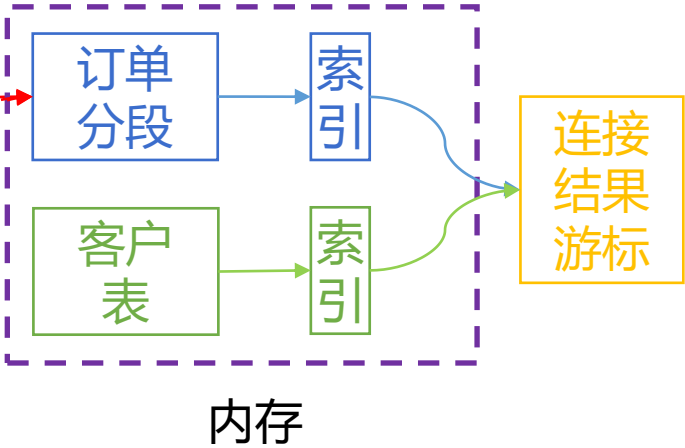
知识点-内连接-游标读取时关联过滤并属性化

去掉客户表没有记录的订单

订单数据量大，文件存放；客户数据量小，全内存

订单文件

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE
1	10262	BATTC	1996-07-22	14487.0
2	10263	ERNSH	1996-07-23	43818.0
3	10264	FOLKO	2007-12-18	1101.0
4	10265	BLONP	1996-07-25	5528.0
5	10266	WARTH	1996-07-26	7719.0
6	10267	FRANK	1996-07-29	20958.0
7	10268	GROSR	1996-07-30	19887.0
8	10269	WHITC	1996-07-31	156.0
9	10270	WARTH	1996-08-01	1654.0
10	10271	SPUR	1996-08-01	908.0
11	10272	BATTC	1996-08-02	19606.0
12	10273	QUICK	1996-08-05	12821.0
13	10274	WNET	1996-08-06	601.0
14	10275	MACAA	1996-08-07	2693.0
15	10276	ORTU	1996-08-08	4152.0
16	10277	MORGK	1996-08-09	37731.0
17	10278	BERGS	1996-08-12	18538.0
18	10279	EHMS	1996-08-13	5166.0
19	10280	BERGS	2007-12-25	898.0
20	10281	ROMEY	1996-08-14	294.0
...				



维表字段过滤后还要参与计算

事实表和维表做内连接，维表不仅用于过滤，在后续的计算中还要使用维表的字段。

读入内存时过滤

游标读入时，仅仅读入关联字段，关联并过滤。关联不上则舍弃本条记录，不再读出其它字段。
关联上的记录要外键属性化。
过滤掉较多记录时可以明显减少硬盘读取数据量，从而提升性能。

代码示例

	A	B
1	=orderfile.cursor(;O_CUSTKEY:CUSTOMER)	/分批读数据时先读入客户号，关联客户表，能关联上则继续读入其它字段，反之则丢弃当前记录
2	=A1.groups(O_CUSTKEY.C_NAME:C_NAME;sum(O_TOTALPRICE):O_TOTALPRICE)	/按照客户公司名称汇总订单的销售额

课堂练习p2.10-游标读取时关联过滤并属性化

练习1：执行p2.10.dfx，订单表ORDERS的客户号O_CUSTKEY和客户表CUSTOMER的主键C_CUSTKEY关联。客户表全内存。记录全字段方式执行时间。

要求：

- 1、客户表过滤条件C_NATIONKEY==15，删除关联不上的记录；
- 2、按照客户公司名称（C_NAME）汇总订单的销售（sum(O_TOTALPRICE)) 。

提示：用cs.switch@i过滤。

练习2：改写p2.10.dfx，用游标读取时关联过滤并属性化的方法，记录仅关联字段方式的执行时间。

提示：用T.cursor(x:C,...;wi...;k:n)，在过滤条件wi中，用K:Ti过滤。

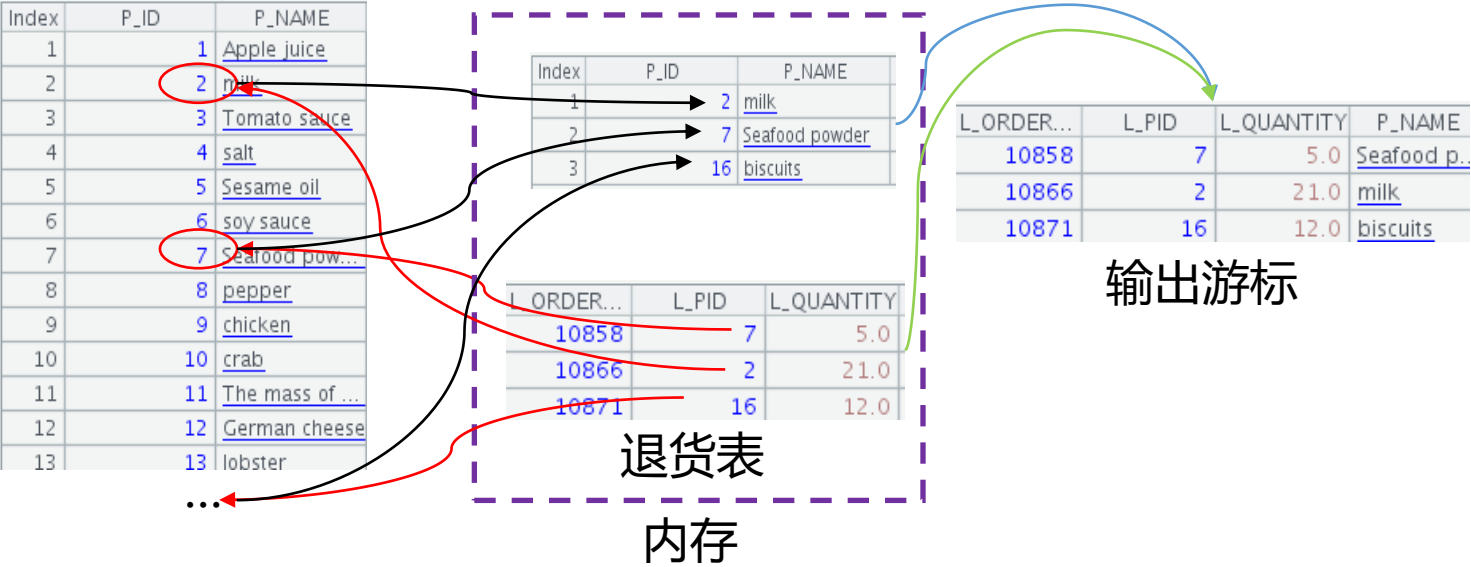
	执行时间（毫秒）
全字段	
仅关联字段	

知识点-大维表、小事实表

查询退货产品的信息

产品表是大维表，文件存放；退货表是小事实表，全内存

产品数据文件



优化思路

退货表小，其中包含的产品号就少。产品号是产品表的主键，少量主键查找速度要比产品表全表遍历快很多。

计算步骤

- 1、红线部分。退货表全内存，用产品号批量查找产品表。
- 2、黑线部分。找到的少量产品数据装入内存。
- 3、蓝线和绿线。少量产品数据与退货表关联输出。

代码示例

	A	
1	=file("RETURN.ctx").open().cursor()	/退货表读入内存
2	=file("PRODUCT.ctx").open()	/产品表文件
3	=A1.joinx@q(L_PID,A2:P_ID,P_NAME,P_TYPENAME,P_PRICE;500000).fetch()	/先查找、再关联，计算结果。n为50万

注意事项

n为50万：如果退货表不超过50万，只查找产品表一次；如果超过，就分几次查找。

课堂练习p2.11-大维表、小事实表

练习1：执行p2.11.dfx，账户表ACCOUNT（1千万条）。
交易表TRADE，每次只计算一天数据，几万条。
账户表的帐户号ACCOUNTID是交易表中OUTID和
RECEIVEID字段的外键，都是一对多的关系。记录一般关联执行时间。
要求：查询各州（账户表STATE字段）之间转账总金额sum(amount)。

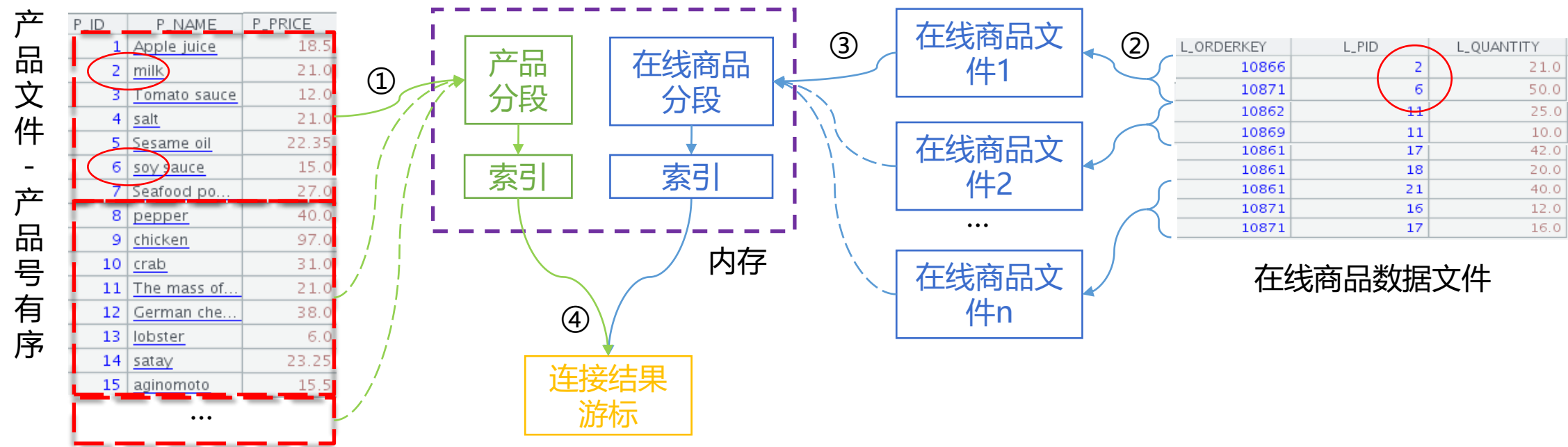
	执行时间（毫秒）
一般关联	
大维表 小事实表	

练习2：改写p2.11.dfx，用大维表小实时表的方法计算，记录大维表小事实表方式执行时间。
提示：cs.joinx，选项@q，当cs数据量不大或是序列时，有序匹配加快速度。
思考：不用@q时，n是交易表游标每次取出的记录数；
用@q的时候，n是什么含义？

知识点-维表事实表都很大-单边哈希手段

按产品分组汇总订单价格

维表产品和在线商品都很大，无法全内存



单边哈希手段计算步骤

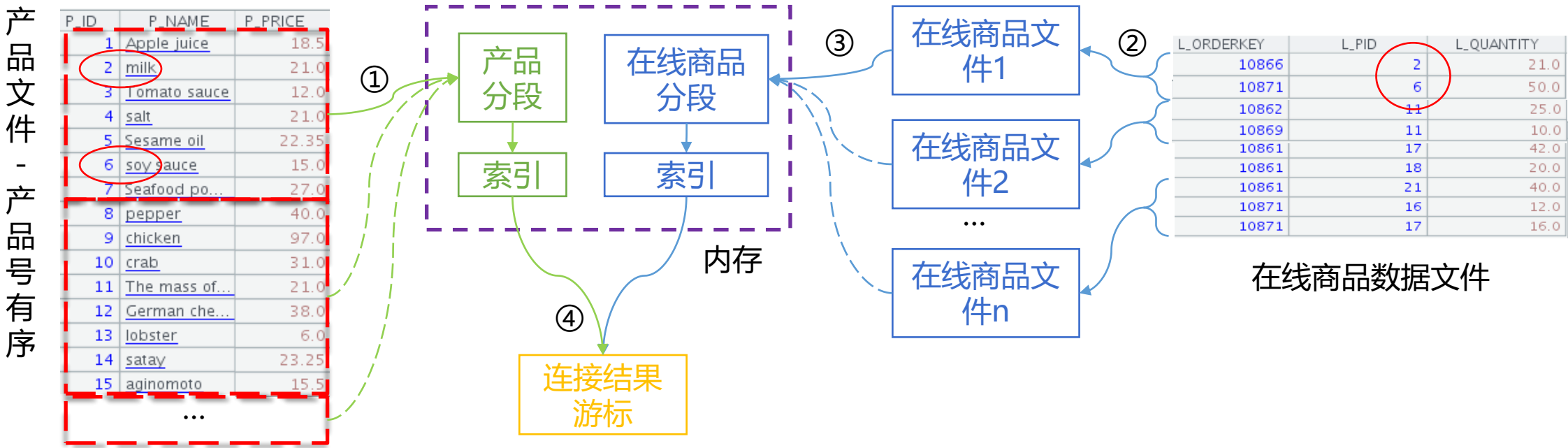
- ①产品文件按主键有序平均分段，形成n个维分段。
- ②按照维的分段，从在线商品文件中找产品号对应的记录，另存成n个临时文件。
- ③分段将产品数据读入内存和对应的在线商品临时文件，在内存中连接。
- ④边连接边输出结果游标，供下一步计算。

单边哈希手段优势

产品文件无需哈希划分，且平均分段。不会出现哈希划分不均匀，二次哈希的情况。

代码示例-维表事实表都很大-单边哈希手段

按产品分组汇总订单价格 维表产品和在线商品都很大，无法全内存



代码示例

	A	B
1	=file("PRODUCT.ctx").open()	/在线商品游标
2	=file("LINEITEM.ctx").open().cursor(P_ID,P_NAME,P_PRICE)	/产品数据文件，已经按主键排序
3	=A1.joinx@u(L_PID,A1:P_ID,P_NAME,P_PRICE)	/单边哈希手段连接
4	=A3.groups(P_NAME;sum(P_PRICE*L_QUANTITY):AMOUNT)	/分组汇总

课堂练习p2.12-单边哈希手段

练习1：执行p2.12.dfx，账户表ACCOUNT（1千万条）。
交易表TRADE，计算全部数据（1千万条）。
账户表的帐户号ACCOUNTID是交易表中OUTID字段的外键，
是一对多的关系。记录一般关联执行时间。

要求：查询各州（账户表STATE字段）之间转出总金额sum(amount）。

练习2：改写p2.12.dfx，用单边哈希手段计算，比较执行时间。
提示：cs.joinx，选项@u。

补充阅读 关系数据采用的外存哈希连接，需要对维表做哈希划分，难以保证平均。如果内存装不下，还要做二次哈希划分。

	执行时间（毫秒）
一般关联	
单边哈希	

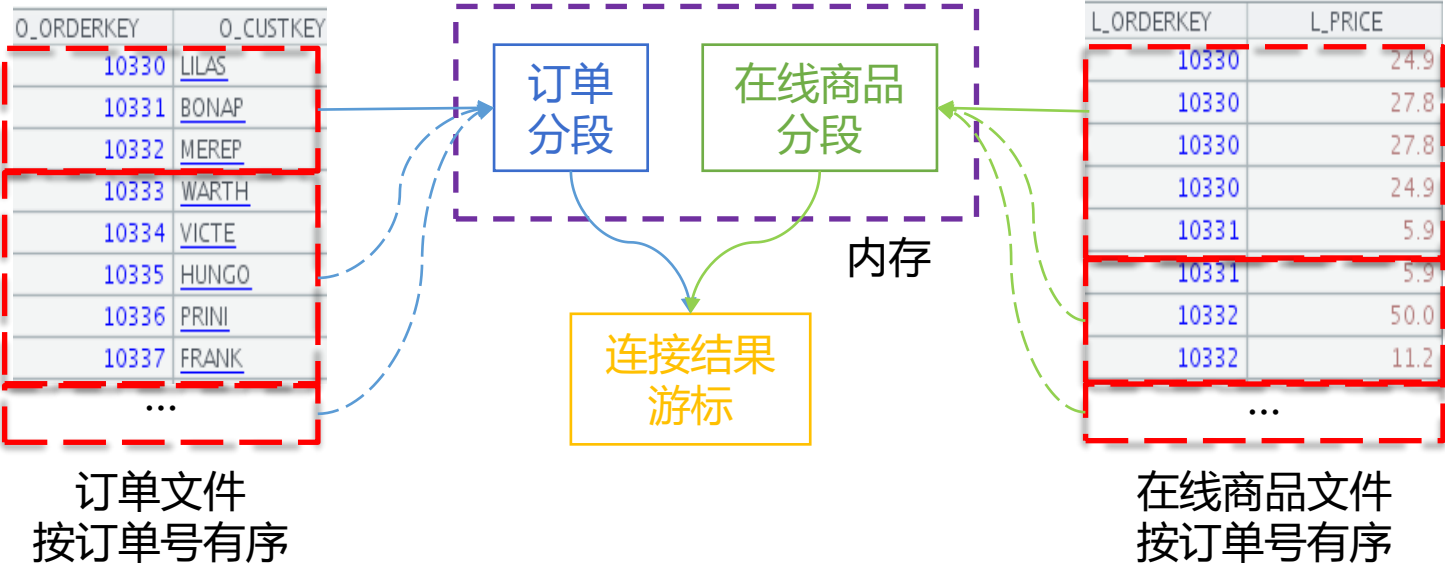
第三章 连接

3.3 主子与同维表

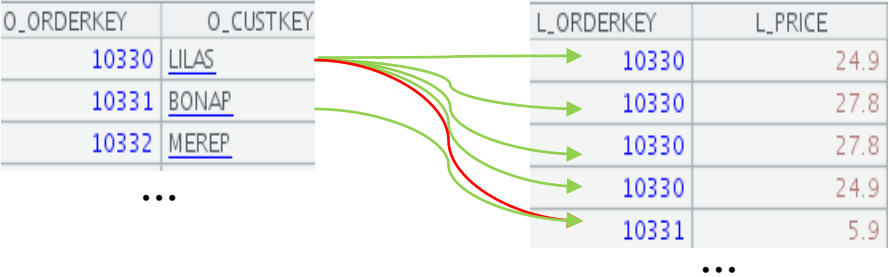
知识点-同维或主子表有序归并

按照客户号分组汇总订单金额

订单（N条记录）和在线商品表（M条记录）都很大，不能全内存



内存中的有序归并



有序归并计算步骤

分批读入订单和在线商品，在内存中有序归并，并输出连接结果游标，供下一步计算。

内存中的有序归并

订单表第一条记录，在线商品能找到等值订单号，做记录后，子表指针下移，直到遇到不匹配的在线商品记录，表示订单表第一条记录的连接完成。

此时在线商品指针不动，订单表指针下移一位，指向第二条记录，以此类推。

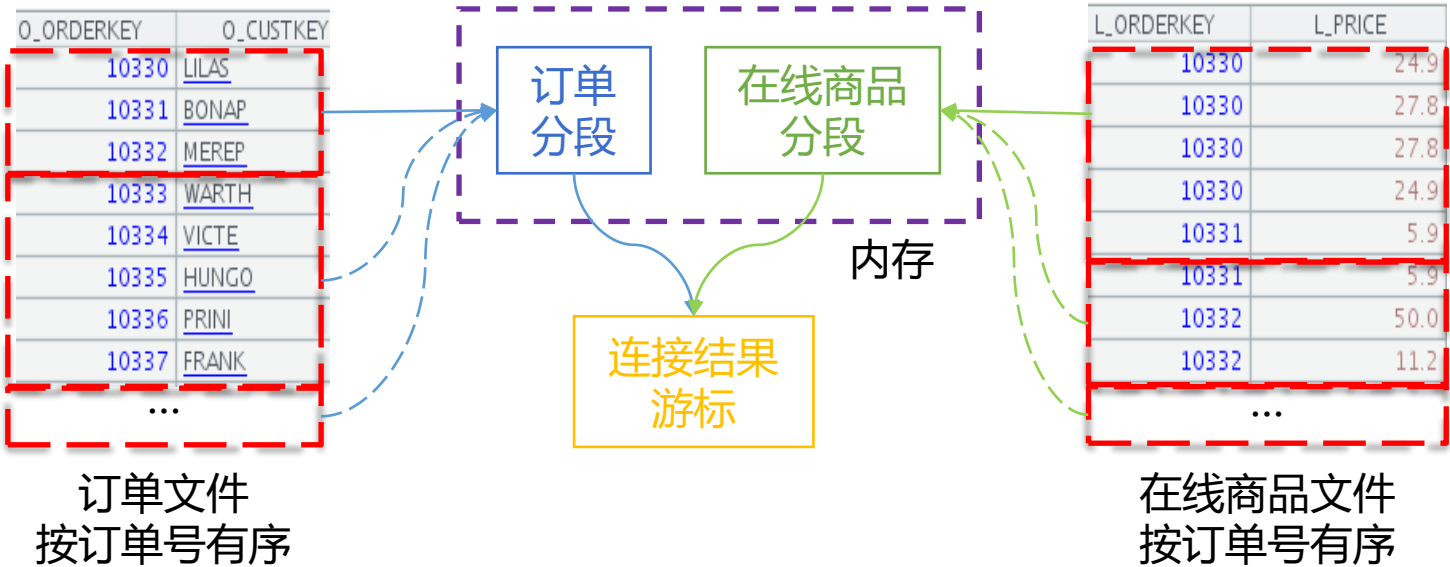
有序归并比较次数

笛卡尔积比较次数 $N \times M$ ；哈希连接比较次数 $\sum(N_i \times M_i)$ ；有序归并比较次数 $N + M$ 。

知识点-同维或主子表有序归并

按照客户号分组汇总订单金额

订单（N条记录）和在线商品表（M条记录）都很大，不能全内存



代码示例

	A	B
1	=joinx(oCursor:O,O_ORDERKEY;lCursor:L,L_ORDERKEY)	/有序归并连接
2	=A1.groups(O.O_CUSTKEY:CUST;sum(L.L_PRICE*L.L_QUANTITY):AMOUNT)	/分组汇总

有序归并与哈希连接比较

在内存中，有序归并不需要计算哈希值，可以节省时间。

在外存中，哈希连接需要将两个文件按照哈希值拆分成相应的临时文件，再做连接。

临时文件需要读写硬盘，非常耗时。

有序归并不需要读写临时文件，性能提升很明显。

课堂练习p3.1-同维或主子表有序归并

练习1：执行p3.1.dfx，在线商品表LINEITEM和订单表ORDERS，两表通过L_ORDERKEY、O_ORDERKEY关联。两表都是外存。记录有序关联执行时间。

要求：

统计每年订单总收入sum(L_EXTENDEDPRICE*(1-detail.L_DISCOUNT))和总数量sum(L_QUANTITY)。

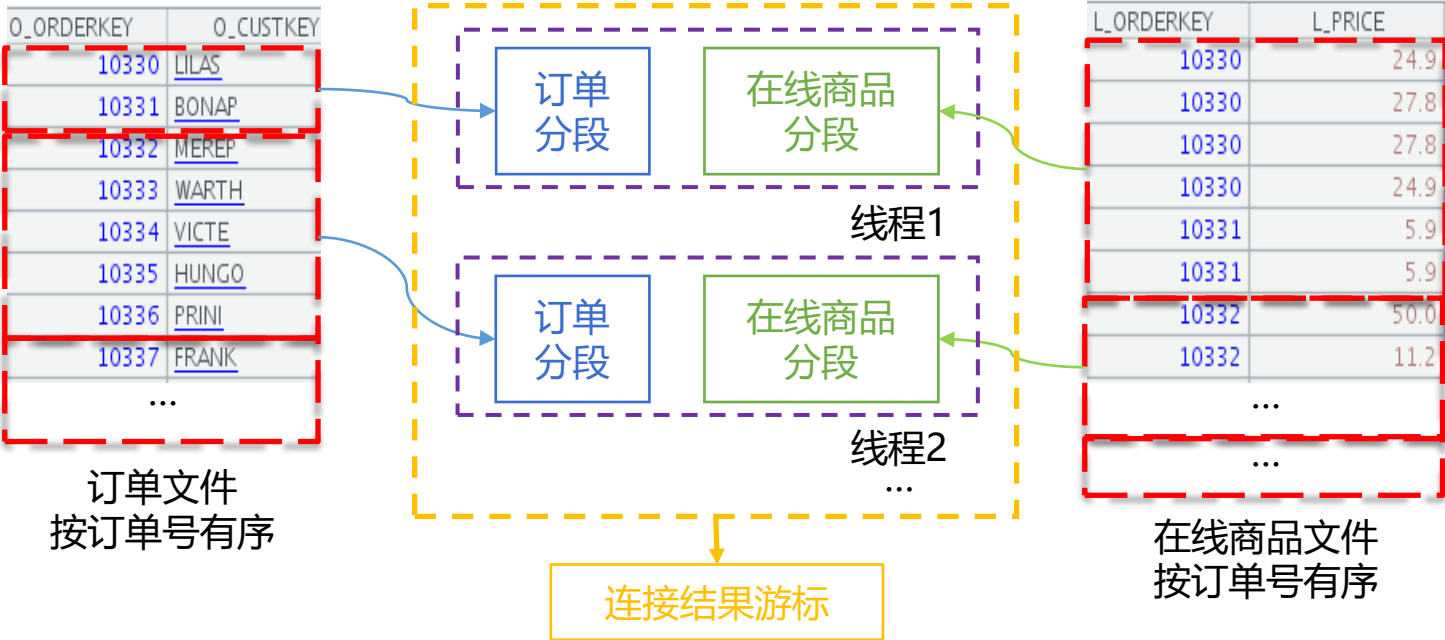
提示：在线商品表对L_ORDERKEY有序；订单表对O_ORDERKEY有序。

	执行时间（毫秒）
有序关联	

知识点-并行有序归并

按照客户号分组汇总订单金额

订单（N条记录）和在线商品表（M条记录）都很大，不能全内存



代码示例

	A	B
1	<code>=file("ORDERS.ctx").open().cursor@m(;;4)</code>	<code>=file("LINEITEM.ctx").open().cursor(;;A1)</code>
2	<code>=joinx(A1:O,O_ORDERKEY;B1:L,L_ORDERKEY)</code>	<code>=A2.groups(O.O_CUSTKEY:CUSTOMER;sum(L.L_PRICE*L.L_QUANTITY):AMOUNT)</code>

A1：订单多路游标，游标路数为4。B1：根据A1，定义在线商品多路游标，路数也是4。A2：并行有序归并。

并行有序归并计算步骤

订单文件按照订单号有序，可以多线程分段读入内存。

每个线程读取之后，根据订单文件中的订单号，在线商品文件中用二分法快速定位数据对应的分段点，读入内存。

每个线程在内存中做有序归并。

所有线程，按照订单号有序归并，输出结果游标。

代码示例-并行有序归并

数据准备：生成组表数据保证主子对齐

以订单表为基准表，在生成数据时对齐，分段时可保证多个表同步，归并计算时不会发生记录错位。

代码示例

	A	B
1	=file("ORDERS.txt").cursor@t(O_ORDERKEY,O_CUSTKEY,O_ORDERDATE)	/分批读入订单表.txt
2	=A1.sortx(O_ORDERKEY)	/按订单号排序
3	=file("ORDERS.ctx").create(#O_ORDERKEY,O_CUSTKEY,O_ORDERDATE)	/创建、打开组表
4	=A3.append(A2)	/将游标中的记录追加写入到组表中
5	=file("LINEITEM.txt").cursor@t(L_ORDERKEY,L_PRICE,L_QUANTITY)	/分批读入在线商品.txt
6	=A5.sortx(L_ORDERKEY)	/按订单号排序
7	=file("LINEITEM.ctx").create(#L_ORDERKEY,L_PRICE,L_QUANTITY;L_ORDERKEY)	/创建、打开组表，按订单号字段分段，不会把订单号同值的记录分到两段中
8	=A7.append(A6)	/将游标中的记录追加写入到组表中

课堂练习p3.2-并行有序归并

练习1： 改写p3.1.dfx，用并行有序归并的方式。
记录并行执行时间，和上个练习执行时间比较。

要求：
并行数为4。

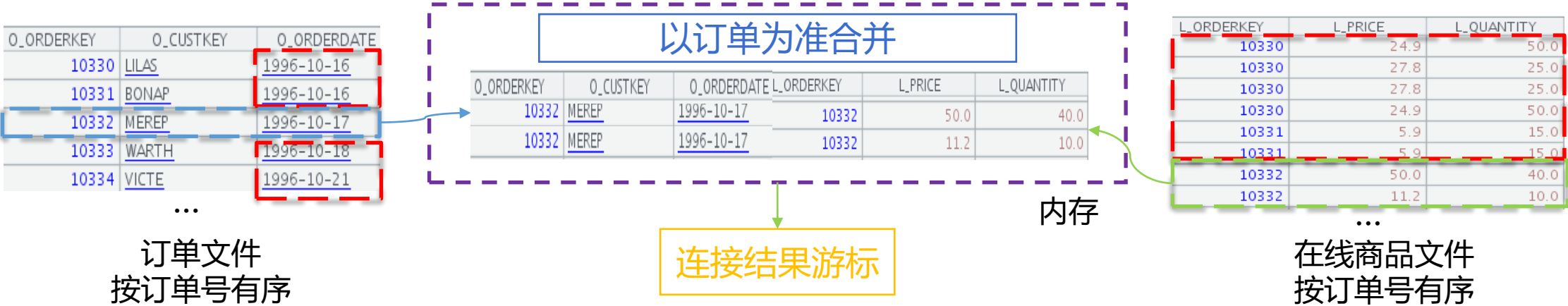
	执行时间（毫秒）
有序关联	
并行 有序关联	

补充阅读 关系数据采用的传统哈希连接很难并行，并行哈希分段时需要同时向某个分段写出数据，造成共享资源冲突；而计算某一段又会几乎耗光所有内存，使其它并行任务无法进行。

知识点-用主表过滤子表

日期等于1996-10-17的订单，按客户分组汇总订单金额

订单和在线商品表都很大，不能全内存



适用场景

过滤后的订单记录较少。而前面讲的有序归并，不能跳过在线商品文件中的红框部分，还是要全部遍历。
此时要用主表过滤子表。

计算步骤

订单文件分批读入内存，先读入日期，不满足条件的舍弃，不必读入其他字段。
满足条件的，在线商品文件中用主键订单号高速定位，连接后输出游标。

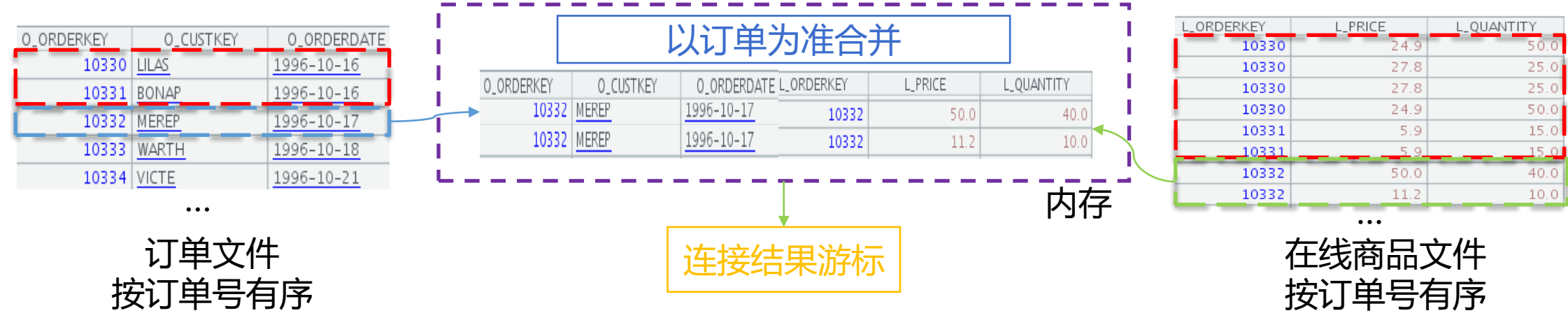
性能优势

订单文件，红色框中的订单不满足条件，直接舍弃。在线商品表中红色框也不会被遍历。当过滤后的订单较少时，可减少硬盘的读取，性能提升。

代码示例-用主表过滤子表

日期等于1996-10-17的订单，按客户分组汇总订单金额

订单和在线商品表都很大，不能全内存



代码示例

	A	B
1	1996-10-17	
2	=file("ORDERS.ctx").open().cursor(;O_ORDERDATE==A1)	/订单表游标，游标前过滤
3	=file("LINEITEM.ctx").open().news(A2,L_PRICE,L_QUANTITY,O_CUSTKEY)	/在线商品游标，引用订单表客户号字段
4	=A3.groups(O_CUSTKEY;sum(L_PRICE*L_QUANTITY):AMOUNT)	/分组汇总客户的订单额

课堂练习p3.3-用主表过滤子表

练习1：执行p3.3.dfx，该dfx为常规有序归并。
主表是订单表ORDERS，子表是在线商品表
LINEITEM，两表中记录分别按O_ORDERKEY、
L_ORDERKEY升序排列，记录有序归并执行时间。

要求：

- 1、主表ORDERS的O_TOTALPRICE小于2000；
- 2、按年分组汇总sum(L_EXTENDEDPRICE*(1-detail.L_DISCOUNT))。

练习2：改写p3.3.dfx，改为用主表过滤子表的方式。记录执行时间。
提示：用cs.news()。

	执行时间（毫秒）
有序归并	
主表过滤子表	

知识点-用子表过滤主表

数量小于20的在线商品，按年分组汇总订单金额

订单和在线商品表都很大，不能全内存

O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
10330	LILAS	1996-10-16
10331	BONAP	1996-10-16
10332	MEREP	1996-10-17
10333	WARTH	1996-10-18
10334	VICTE	1996-10-21

订单文件
按订单号有序



L_ORDERKEY	L_PRICE	L_QUANTITY
10330	24.9	50.0
10330	27.8	25.0
10330	27.8	25.0
10330	24.9	50.0
10331	5.9	15.0
10331	5.9	15.0
10332	50.0	40.0
10332	11.2	10.0

在线商品文件
按订单号有序

适用场景

过滤后的在线商品较少。而前面讲的有序归并，不能跳过订单文件中的红框部分，还是要全部遍历。
此时要用子表过滤主表。

计算步骤

在线商品文件分批读入内存，先读入数量，不满足条件的舍弃，不必读入其他字段。
满足条件的，先汇总求和，再在订单文件中用主键订单号高速定位，连接后输出游标。

性能优势

在线商品文件，红色框中的记录不满足条件，直接舍弃。所以订单表中红色框也不会被遍历，当过滤后的记录较少时，可减少硬盘的读取，性能得到提升。

代码示例-用子表过滤主表

按照年分组汇总订单金额

订单和在线商品表都很大，不能全内存

O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
10330	LILAS	1996-10-16
10331	BONAP	1996-10-16
10332	MEREP	1996-10-17
10333	WARTH	1996-10-18
10334	VICTE	1996-10-21

订单文件
按订单号有序



L_ORDERKEY	L_PRICE	L_QUANTITY
10330	24.9	50.0
10330	27.8	25.0
10330	27.8	25.0
10330	24.9	50.0
10331	5.9	15.0
10331	5.9	15.0
10332	50.0	40.0
10332	11.2	10.0

在线商品文件
按订单号有序

代码示例

	A	
1	=file("LINEITEM.ctx").open().cursor@m(L_QUANTITY<20;4)	/在线商品游标，游标前过滤
2	=file("ORDERS.ctx").open().new(A1,O_ORDERKEY,O_ORDERDATE,sum(L_PRICE*L_QUANTITY):v)	/订单游标，引用在线商品字段，并汇总
3	=A2.groups(year(O_ORDERDATE):year;sum(v):AMOUNT)	/按年分组汇总

课堂练习p3.4-用子表过滤主表

练习1：执行p3.4.dfx，主表是订单表ORDERS，子表是在线商品表LINEITEM，两表中记录分别按O_ORDERKEY、L_ORDERKEY升序排列，记录有序归并执行时间。

要求：

- 1、LINEITEM的L_QUANTITY小于50；
- 2、按年分组汇总sum(L_EXTENDEDPRICE*(1-detail.L_DISCOUNT))。

练习：改写p3.4.dfx，改为用子表过滤主表的方式。记录执行时间并比较。
提示：用cs.new()。

	执行时间（毫秒）
有序归并	
子表过滤主表	

知识点-主子表一体化存储

按照客户分组汇总订单金额

订单和在线商品表都很大，不能全内存

订单文件

O_ID	C_ID	O_DATE
10248	VINET	2018-03-02
10249	TOMSP	2018-03-03
...
...

L_ID	L_SUBID	PRICE	NUMS
10248	1024801	14.00	12
10248	1024802	9.00	10
10249	1024901	18.00	9
...

在线商品文件

组合存储

O_ID	L_SUBID	PRICE	NUMS	C_ID	O_DATE
10248				VINET	2018-03-02
	1024801	14.00	12		
	1024802	9.00	10		
10249				TOMSP	2018-03-03
	1024901	18.00	9		
...	

订单-在线商品组合文件

主子表一体化存储

主子表采用层次存储还可以进一步提升计算性能。

将主子表组合固化在存储格式中，使用时无需再次关联从而获得更高性能。

性能提升效果

订单1亿条，每条对应大约10条在线商品，测试结果：

耗时单位（秒）

主子表joinx	主子合一	主子合一(4线程)
781	602	368

代码示例-主子表一体化存储

按照客户分组汇总订单金额

订单和在线商品表都很大，不能全内存

代码示例

存储准备

	A	B
1	=db.cursor("select * from ORDERS order by O_ID")	
2	=db.cursor("select * from LINEITEM order by L_ID")	
3	=file("MULTIPLE.ctx").create(#O_ID,C_ID,O_DATE)	
4	=A3.append(A1)	=A3.attach(LINEITEM,#L_SUBID,PRICE,NUMS)
5	=B4.append(A2)	

分组计算

	A	B
1	=file("MULTIPLE.ctx").open().attach(LINEITEM)	/打开附表在线商品
2	=A1.cursor@m(O_ID,C_ID,PRICE,NUMS;;4)	/建立附表多路游标,路数4
3	=A2.groups(C_ID:CUSTOMER;sum(PRICE*NUMS):AMOUNT)	/按客户分组汇总销售额

课堂练习p3.5-主子表一体化存储

练习1：执行p3.5.dfx，LINEITEM和ORDERS，两表通过L_ORDERKEY、O_ORDERKEY关联，记录有序归并执行时间。

要求：
统计每年订单总收入sum(L_EXTENDEDPRICE*(1-L_DISCOUNT))和总数量sum(L_QUANTITY)。

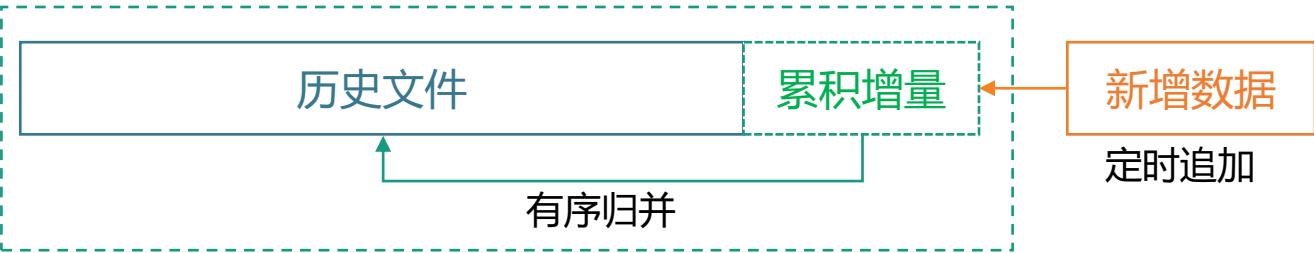
练习2：写出p3.5_attach.dfx,从orders.ctx和lineitem.ctx中读取数据来生成附表文件orders_lineitem.ctx。

练习3：改写p3.5.dfx，利用带有附表的组表文件orders_lineitem.ctx来完成相同的计算，记录附表执行时间并比较。

	执行时间（毫秒）
有序归并	
附表	

知识点-有序和数据更新

订单表更新数据



历史文件

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
1	10248	VINET	2011-10-04
2	10249	TOMSP	1996-07-05
3	10250	HANAR	1996-07-08
4	10251	VICTE	1996-07-09
5	10253	HANAR	1996-07-10
6	10254	CHOPS	1996-07-11

增量

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
1	10252	SUPRD	1996-07-09
2	10255	RICSU	1996-07-12

有序归并

Index	O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
1	10248	VINET	2011-10-04
2	10249	TOMSP	1996-07-05
3	10250	HANAR	1996-07-08
4	10251	VICTE	1996-07-09
5	10252	SUPRD	1996-07-09
6	10253	HANAR	1996-07-10
7	10254	CHOPS	1996-07-11
8	10255	RICSU	1996-07-12

归并后文件

历史文件并非按照日期有序

历史文件如果按照日期有序，则可以直接将新增数据追加到文件末尾。

历史数据按照主键排序存储，则需要将新增数据也按照主键排序后，有序归并入历史文件。

增量数据更新

新增数据有序后，更新的过程就是有序归并。

把新增数据单独排序后和已有序的历史数据归并，而不必把历史、新增合并后重新排序，有效减少数据更新时间。

代码示例

	A	B
1	=file("ORDERS.ctx").open()	/打开订单表
2	=db.query@x("select * from ORDERS where O_ORDERDATE>=' 1996-07-09' order by O_ORDERKEY")	/数据库排序取数
3	=A1.append@m(A2.cursor())	/有序归并

课堂练习p3.6-有序和数据更新

练习1：执行p3.6_prepare.dfx。

要求：将订单表ORDERS随机拆分为两份数据。组表文件（orders_main.ctx）是主文件有140万行；集文件（orders_new.btx）是待追加的新数据有10万行。两文件都对O_ORDERKEY有序。

练习2：写p3.6.dfx。

要求：将待合并新增数据集文件（orders_new.btx），以有序归并方式加入主文件（orders_main.ctx），完成数据更新。

第三章 连接

3.4 子查询转换

知识点-子查询转换成连接总体说明



讲解什么情况下的子查询可以转为连接，
转为连接后，可用前面的优化方法提性能。



本节例子中涉及到的表，都按主键有序存储。

知识点-事实表只包含维表部分主键

保险、理财分别按业务号和比例号分组汇总

business_id	money
1	19391
...	...

保险主表
对应维表分类001

business_id	money
1	86597
...	...

理财主表
对应维表分类002

Category	business_id	rate
001	1	0.3
002	1	0.5
...

业务维表

代码示例 保险主表按业务号和比例号分组汇总

```
select d.business_id, d.rate, sum(t.money) as money from transaction_001 as t, business_dim as d
where d. category = '001' and t. business_id = d.business_id
group by d.business_id, d.rate
```

数据结构说明

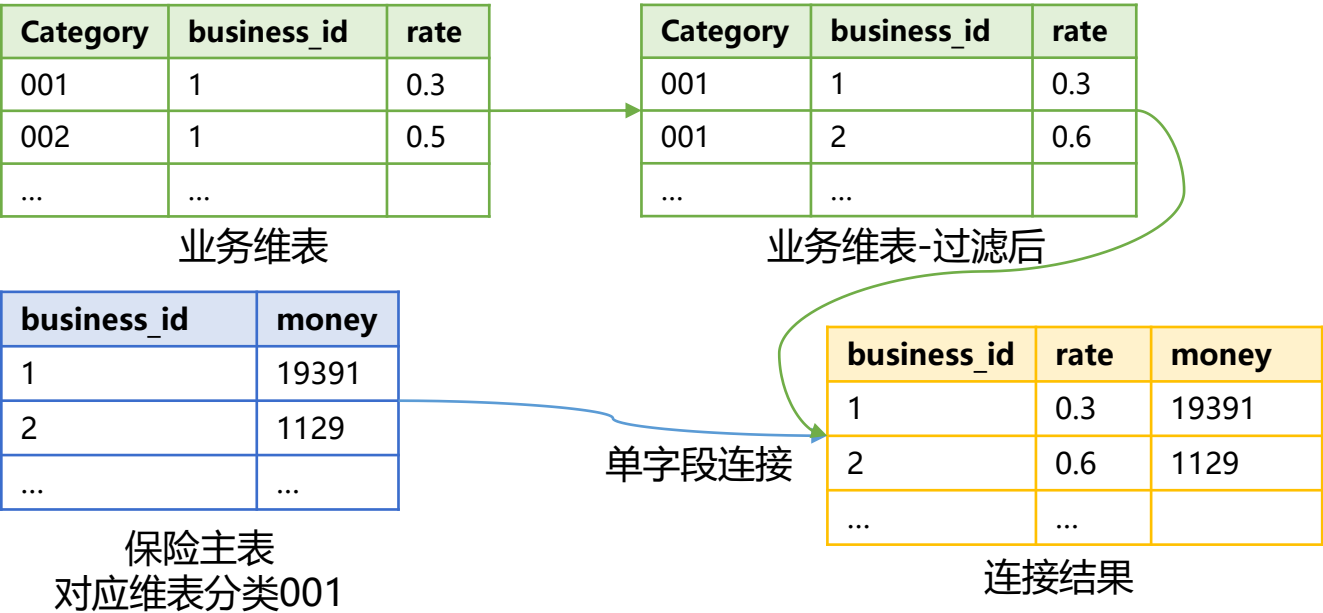
保险主表、理财主表是事实表，包含外键字段业务号。关联同一个维表"业务维表"。

业务维表的主键是分类号和业务号。保险主表对应业务分类号是001，理财主表对应002。

保险主表与分类号是001的记录关联，理财主表与分类号是002的记录关联。

知识点-事实表只包含维表部分主键

优化思路1：先过滤再连接



性能分析

对业务维表过滤得到以业务号为主键的新维表，再与主表关联。

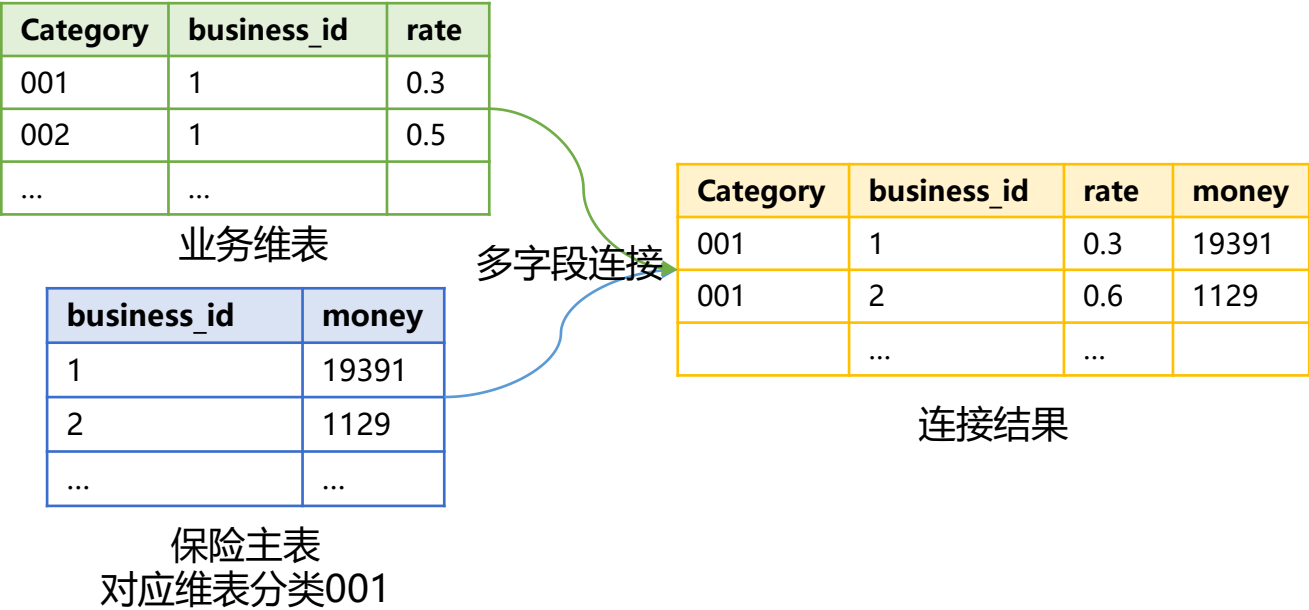
维表过滤之后再关联的时候，不能再利用原有的两个字段的索引，要按业务号建索引，会增加计算时间。

代码示例

	A
1	=file("business_dim.btx").import@b()
2	=A1.keys@i(category, business_id)
3	=A2.select(category=="001")
4	=file("transaction_001.btx").cursor@b(business_id,money)
5	=A4.join(business_id, A3:business_id, rate)
6	=A5.groups(business_id, rate; sum(money):money)

知识点-事实表只包含维表部分主键

优化思路2：多字段连接



代码示例

```
A
1 =file("business_dim.btx").import@b()
2 =A1.keys@i(category, business_id)
3 =file("transaction_001.btx").cursor@b(business_id,money)
4 =A3.join("001": business_id, A1, rate)
5 =A4.groups(business_id, rate; sum(money):money)
```

性能分析

思路二将001看作主表的常数字段。主表就变成了多字段主键，和维表做多字段关联。这样可以直接利用原两字段索引，不必重建索引。比思路一减少了过滤和重建索引计算。但思路二利用索引关联时，就需要计算两个字段做关联。而思路一，只需要计算一个字段。

适用场景

思路一适合维表较小，事实表较大的情况。
思路二适合维表较大，事实表较小的情况。

课堂练习p4.1-事实表只包含维表部分主键

练习1：打开p4.1.dfx，其中业务维表(10*1万行)
(business_dim.btx) 主键是分类号category
和业务号business_id;

保险主表 (transaction_001.btx) 100万行，对应业务分类号
是001，业务号为business_id，两表关联。

要求：按业务号business_id、比例rate分组，汇总金额sum(money)。

	执行时间（毫秒）
先过滤	
常数字段	

练习2：改写p4.1.dfx，由于已知分类号为1，可将1看作主表的常数字段。主表就变成了多字
段主键，和维表做多字段关联。

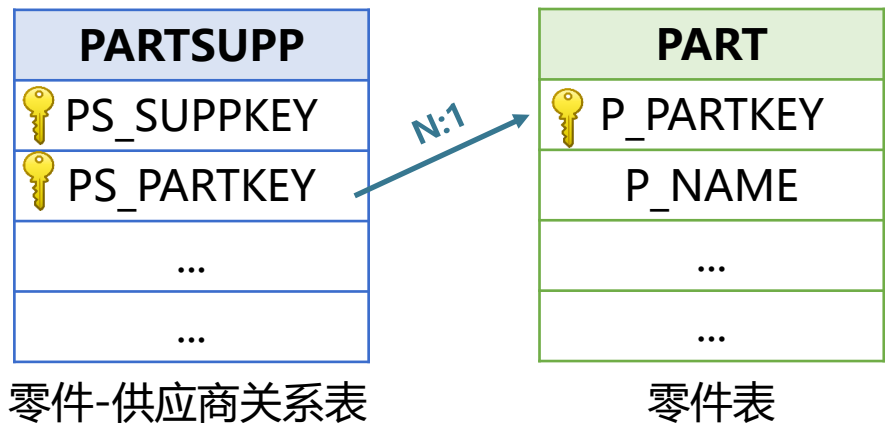
思考：改写create/createBT.dfx，加大业务维表为10*3千万行，会观察到常数字段计算方法
更快，为什么？

提示：需要16G以上内存服务器测试。

知识点-外键表的IN、EXISTS

供应商满足零件名称条件的零件个数

零件表过滤结果较小，可全内存。



数据结构说明

零件表是零件-供应商关系表的外键表，通过零件号关联。

计算要求

在零件表中找，名称包含"bisque%%"。以此为淮在零件-供应商关系表中找对应的记录，并按照供应商分组汇总记录数。

代码示例

```
SELECT PS_SUPPKEY, COUNT(1) AS S_COUNT FROM PARTSUPP
WHERE PS_PARTKEY IN (SELECT P_PARTKEY FROM PART WHERE P_NAME LIKE 'bisque%%')
GROUP BY PS_SUPPKEY
```

SQL

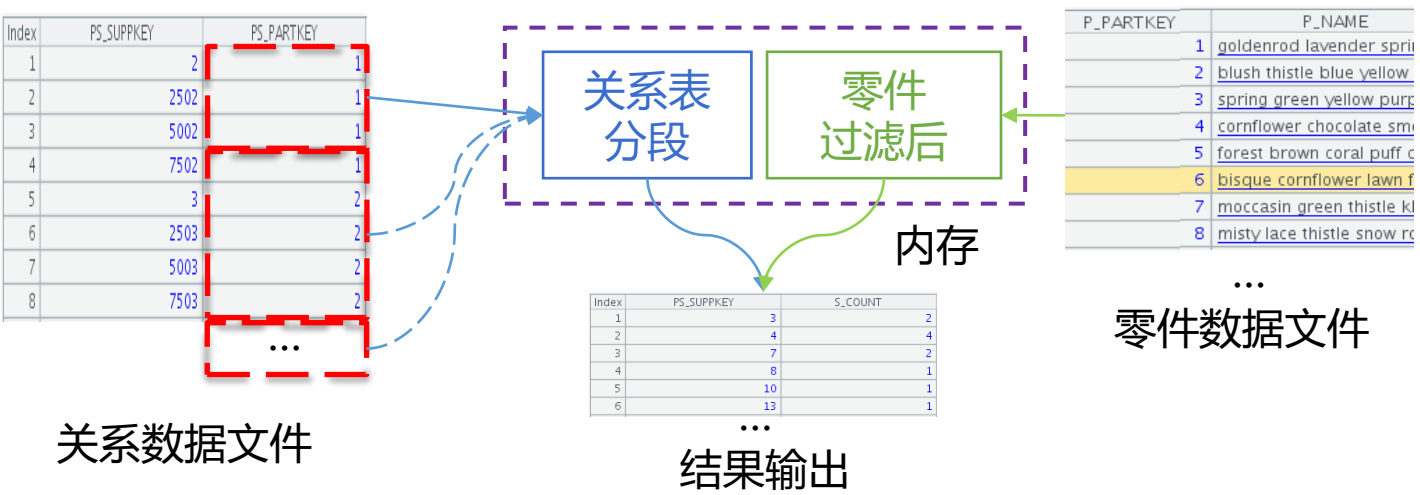
```
SELECT PS_SUPPKEY, COUNT(1) AS S_COUNT FROM PARTSUPP
WHERE EXISTS (SELECT * FROM PART WHERE P_PARTKEY = PS_PARTKEY
              AND P_NAME LIKE 'bisque%%')
GROUP BY PS_SUPPKEY
```

SQL

知识点-外键表的IN、EXISTS

优化思路：外键表内存过滤后关联

零件表过滤结果较小，可全内存。



代码示例

	A
1	=file("PART.ctx").open().cursor(P_PARTKEY,P_NAME; like(P_NAME, "bisque*"))
2	=A1.fetch().index()
3	=file("PARTSUPP.ctx").open().cursor(PS_SUPPKEY, PS_PARTKEY; A2.find(PS_PARTKEY))
4	=A3.groups(PS_SUPPKEY; count(1):S_COUNT)

计算步骤

零件表按照条件过滤，读入内存后建立索引。

分批读取关系表，先读入零件编号，如果能关联上，再度去其他字段。否则直接舍弃。

关联上的数据直接在内存中做小分组汇总。

性能分析

关联不上则不再读出其它字段，过滤掉较多记录时可以明显减少IO操作从而提升性能。

课堂练习p4.2-外键表的IN、 EXISTS

练习1： 执行p4.2.dfx， 记录全字段执行时间。

要求： 在零件表PART中找名称P_NAME

以"saddle"开头的。 以此为准在在线商品表

LINEITEM中找对应的记录， 并按照

供应商L_SUPPKEY分组汇总记录数。

提示： 用join@i(f,T)要取全部记录后过滤。

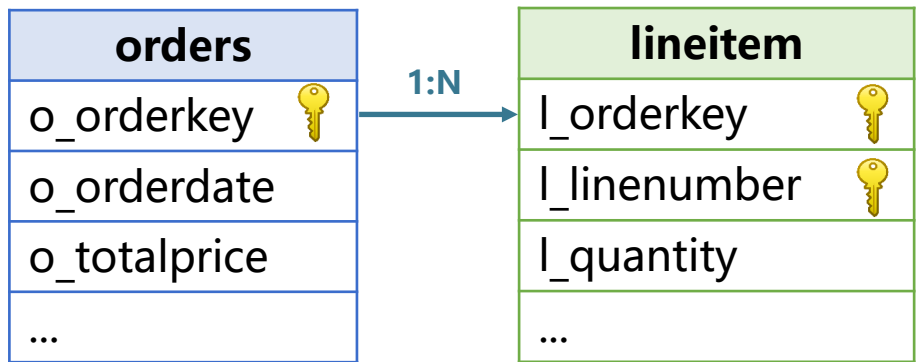
	执行时间（毫秒）
全字段	
仅关联字段	

练习2： 改写p4.2， 用外键表内存过滤后关联方式计算， 记录仅关联字段执行时间。

提示： 用cursor(;T.find(f))。

知识点-主子表的IN、EXISTS

订单和在线商品表



计算要求

在线商品表按照L_COMMITDATE < L_RECEIPTDATE过滤。订单表以在线商品表的过滤结果为准，再以日期条件过滤后，按照O_ORDERPRIORITY分组汇总。

关联字段只是在线商品表主键的一部分。

代码示例

```
SELECT O_ORDERPRIORITY, COUNT(*) AS O_COUNT FROM ORDERS
WHERE O_ORDERDATE >= DATE '1995-10-01'
      AND O_ORDERKEY IN (SELECT L_ORDERKEY FROM LINEITEM WHERE L_COMMITDATE< L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
```

SQL

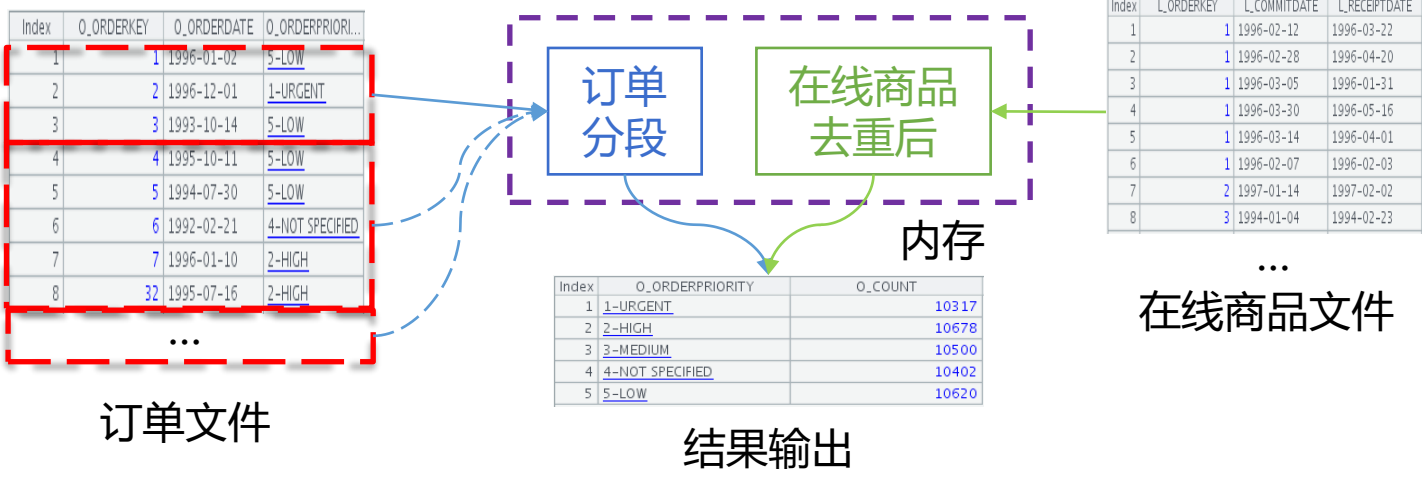
```
SELECT O_ORDERPRIORITY, COUNT(*) AS O_COUNT FROM ORDERS
WHERE O_ORDERDATE >= DATE '1995-10-01'
      AND EXISTS (SELECT * FROM LINEITEM WHERE L_ORDERKEY = O_ORDERKEY
                  AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
```

SQL

知识点-主子表的IN、EXISTS

优化思路：子表分组去重后关联

在线商品过滤结果较小，可全内存。



计算步骤

在线商品表过滤后，按照订单号分组去掉重复，订单号就是唯一的，相当于逻辑主键。数据量较小可全内存。

订单文件分批读入，与全内存的在线商品数据内连接。

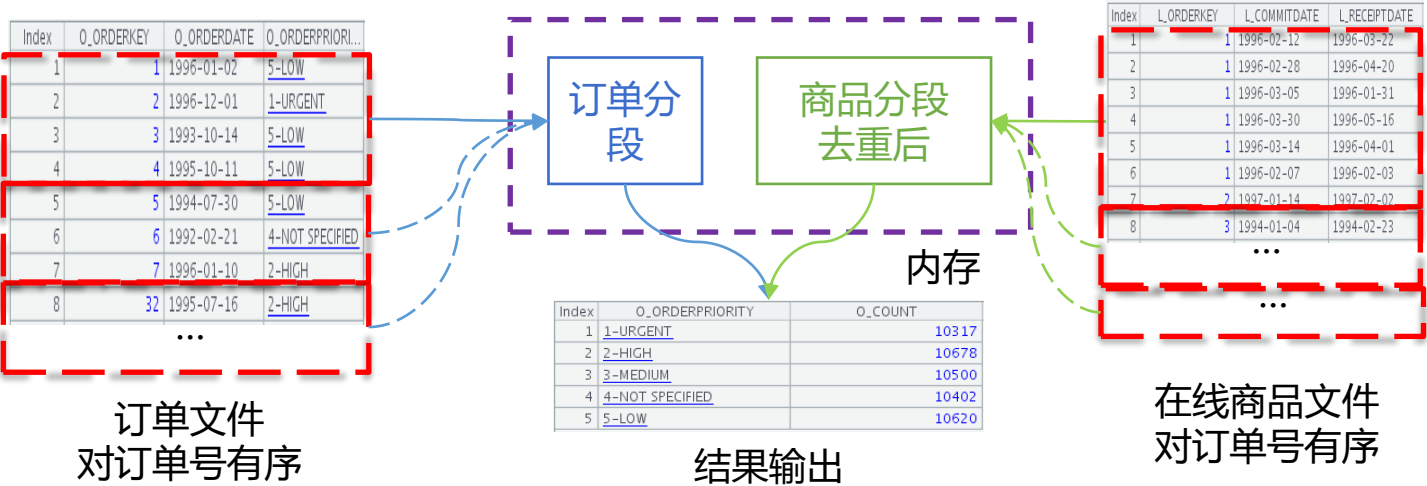
代码示例

	A
1	1995-10-01
2	=file("LINEITEM.btx").cursor@b(L_ORDERKEY,L_COMMITDATE,L_RECEIPTDATE)
3	=A2.select(L_COMMITDATE < L_RECEIPTDATE)
4	=A3.groups(L_ORDERKEY)
5	=file("ORDERS.btx").cursor@b(O_ORDERKEY,O_ORDERDATE,O_ORDERPRIORITY)
6	=A5.select(O_ORDERDATE >= A1)
7	=A6.join@i(O_ORDERKEY, A4:L_ORDERKEY)
8	=A7.groups(O_ORDERPRIORITY;count(1):O_COUNT)

知识点-主子表的IN、EXISTS

进一步优化：有序归并

在线商品过滤结果较大，无法全内存。



计算步骤

在线商品表分批读入内存，过滤后，按照订单号分组去掉重复，订单号就是唯一的，相当于逻辑主键。

订单文件分批读入，与内存中的在线商品数据内连接，有序归并。

代码示例

	A
1	1995-10-01
2	=file("LINEITEM.btx").cursor@b(L_ORDERKEY,L_COMMITDATE,L_RECEIPTDATE)
3	=A2.select(L_COMMITDATE < L_RECEIPTDATE)
4	=A3.group@1(L_ORDERKEY)
5	=file("ORDERS.btx").cursor@b(O_ORDERKEY,O_ORDERDATE,O_ORDERPRIORITY)
6	=A5.select(O_ORDERDATE >= A1)
7	=joinx(A6:O,O_ORDERKEY; A4:L,L_ORDERKEY)
8	=A7.groups(O.O_ORDERPRIORITY;count(1):O_COUNT)

课堂练习p4.3-主子表的IN、 EXISTS

练习1： 执行p4.3.dfx， 订单表ORDERS以在线商品表LINEITEM的过滤结果为准， 按照O_ORDERPRIORITY分组汇总记录数。

要求：

- 1、 LINEITEM的L_COMMITDATE小于L_RECEIPTDATE;
- 2、 LINEITEM的L_SHIPDATE小于1993-01-01;
- 3、 ORDERS的O_ORDERDATE小于1993-01-01。

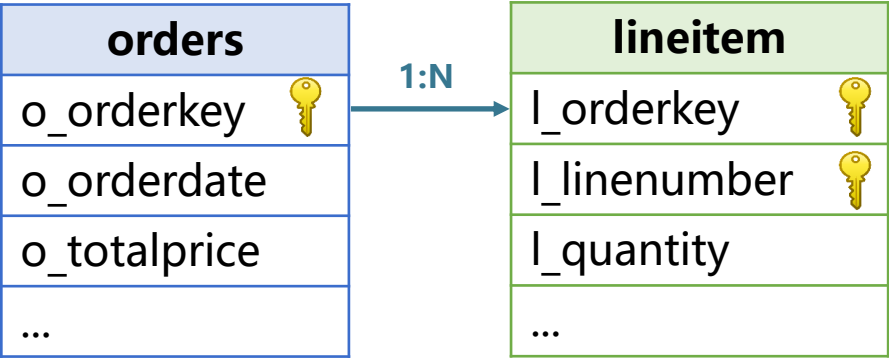
提示： 因为有L_SHIPDATE的时间限定， L_ORDERKEY分组去重结果可放入内存， 用cs.join@i()。

练习2： 改写p4.3.dfx， 不限定LINEITEM的L_SHIPDATE小于1993-01-01 其他要求不变。

提示： 因为取LINEITEM全量数据， L_ORDERKEY分组去重结果无法装入内存， 用joinx()。

知识点-主子表,关联字段过滤成逻辑主键

订单和在线商品主子表



计算要求

在线商品表按照L_COMMITDATE < L_RECEIPTDATE并且L_LINENUMBER = 1 过滤。订单表以在线商品表的过滤结果为准，再以日期条件过滤后，按照 O_ORDERPRIORITY分组汇总。

L_LINENUMBER = 1 条件过滤之后，在线商品的订单号已经唯一，是逻辑主键了。

代码示例

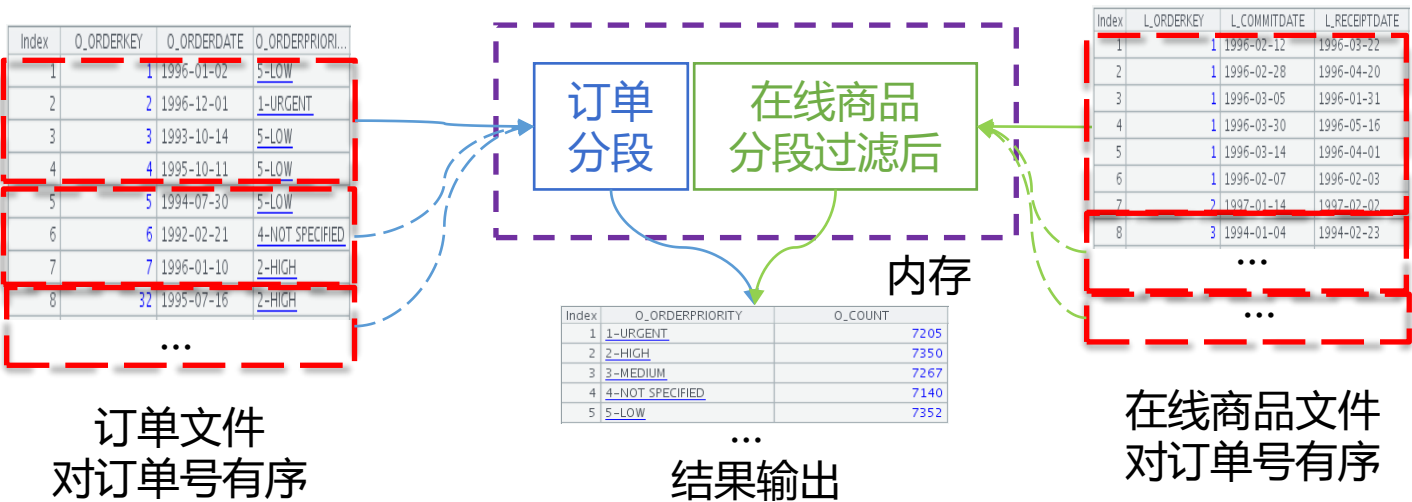
```
SELECT O_ORDERPRIORITY, COUNT(*) AS O_COUNT FROM ORDERS
WHERE O_ORDERDATE >= DATE '1995-10-01'
      AND O_ORDERKEY IN (SELECT L_ORDERKEY FROM LINEITEM
                        WHERE L_LINENUMBER = 1 and L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
```

SQL

知识点-主子表,关联字段过滤成逻辑主键

优化思路：有序归并

在线商品表过滤结果较大，无法全内存。



代码示例

	A
1	1995-10-01
2	=file("LINEITEM.btx").cursor@b(L_ORDERKEY,L_LINENUMBER,L_COMMITDATE,L_RECEIPTDATE)
3	=A2.select(L_LINENUMBER == 1 && L_COMMITDATE < L_RECEIPTDATE)
4	=file("ORDERS.btx").cursor@b(O_ORDERKEY,O_ORDERDATE,O_ORDERPRIORITY)
5	=A4.select(O_ORDERDATE>=A1)
6	=joinx(A5:O,O_ORDERKEY; A3:L,L_ORDERKEY)
7	=A6.groups(O.O_ORDERPRIORITY;count(1):O_COUNT)

计算步骤

在线商品表分批读入内存，过滤后，按照订单号分组去掉重复，订单号就是唯一的，相当于逻辑主键。

订单文件分批读入，与内存中的在线商品数据内连接，有序归并。

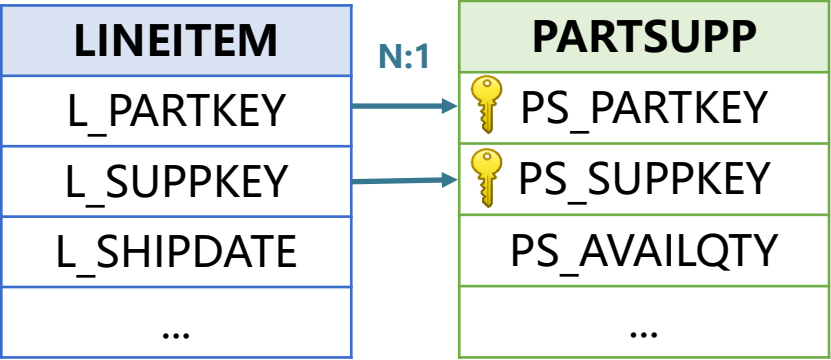
课堂练习p4.4-主子表,关联字段过滤成逻辑主键

练习1：继续改写p4.3.dfx。增加条件：LINEITEM的L_LINENUMBER等于1，其他要求不变。

提示：L_LINENUMBER=1过滤后，在线商品的订单号已经唯一，是逻辑主键了。

知识点-WHERE子查询转换为连接

在线商品和零件供应商关系表



计算要求

在线商品按照日期过滤，并计算主键分组汇总值。

零件供应商表的过滤条件是字段值 PS_AVAILQTY 大于在线商品的汇总值。关联字段是零件号和供应商号。

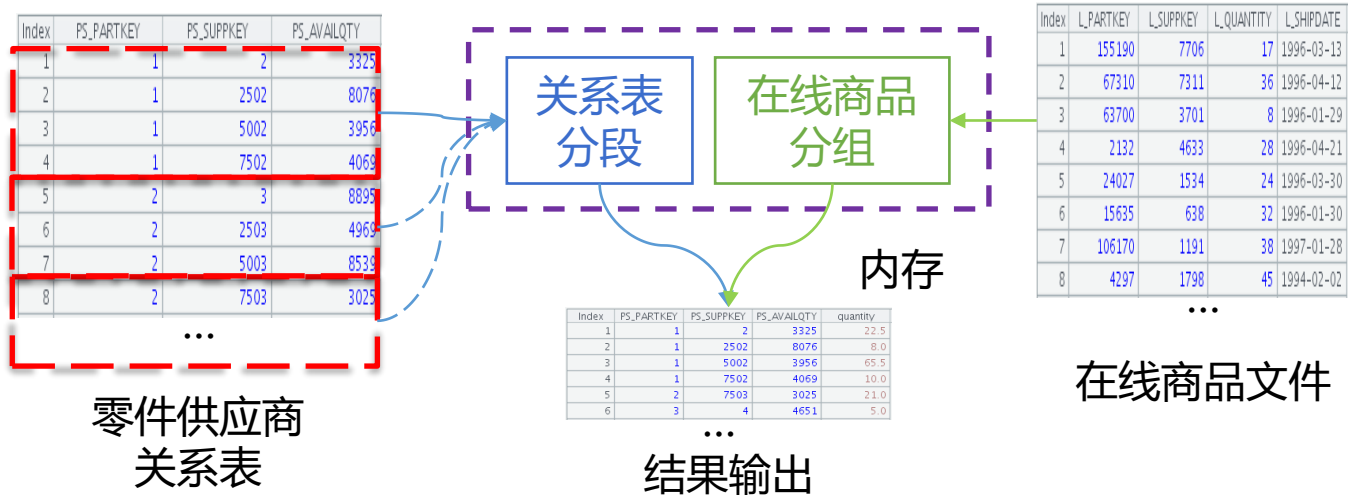
代码示例

```
SELECT PS_SUPPKEY FROM PARTSUPP
WHERE PS_AVAILQTY > (SELECT 0.5 * SUM(L_QUANTITY) FROM LINEITEM
                     WHERE L_PARTKEY = PS_PARTKEY AND L_SUPPKEY = PS_SUPPKEY
                     AND L_SHIPDATE >= DATE '1995-04-01' )
```

SQL

知识点-WHERE子查询转换为连接

优化思路：计算临时维表



代码示例

	A	B
1	1995-04-01	/获取参数日期的下一年
2	=file("LINEITEM.btx").cursor@b(L_PARTKEY,L_SUPPKEY,L_QUANTITY,L_SHIPDATE)	/在在线商品集文件上定义游标
3	=A2.select(L_SHIPDATE >= A1)	/对游标附加过滤操作
4	=A3.groups@u(L_PARTKEY,L_SUPPKEY;sum(L_QUANTITY) * 0.5:quantity)	/分组汇总,@u为结果不按分组字段排序
5	=file("PARTSUPP.btx").cursor@b(PS_PARTKEY,PS_SUPPKEY,PS_AVAILQTY)	/对游标附加过滤操作
6	=A5.join@i(PS_PARTKEY:PS_SUPPKEY,A4:L_PARTKEY:L_SUPPKEY,quantity)	/对PARTSUPP连接过滤,@i表示内连接
7	=A6.select(PS_AVAILQTY>quantity).fetch()	/对游标过滤得到最终结果

课堂练习p4.5-WHERE子查询转换为连接

练习：写出p4.5.dfx，实现下面SQL的子查询。

SQL:

```
SELECT PS_SUPPKEY FROM PARTSUPP
WHERE PS_AVAILQTY > (SELECT SUM(L_QUANTITY) FROM LINEITEM
                     WHERE L_PARTKEY = PS_PARTKEY AND L_SUPPKEY = PS_SUPPKEY
                     AND L_SHIPDATE < DATE '1993-01-01' )
```

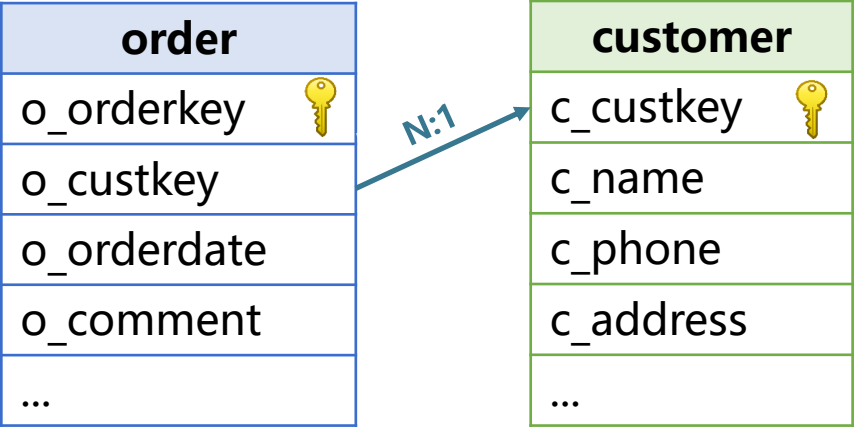
提示：用计算临时维表的方式实现。

分组汇总时可以用groups@u提速,@u为结果不按分组字段排序。

关联则使用cs.join@i()。

知识点-集合运算-差集

查询没有订单的客户个数



计算要求

在客户表中有记录，在订单表中没有下单记录的客户。

代码示例

```
SELECT COUNT(1) FROM CUSTOMER
WHERE NOT EXISTS (SELECT * FROM ORDERS WHERE O_CUSTKEY = C_CUSTKEY)
```

SQL

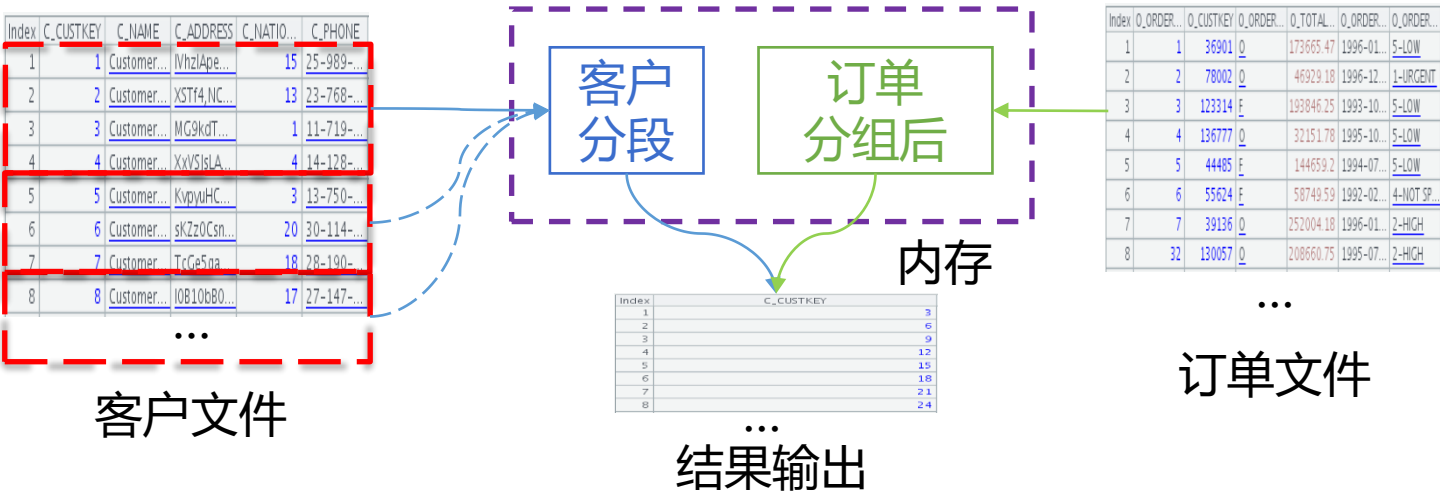
知识点-集合运算-差集

优化思路1：子查询结果较小，可全内存

计算步骤

订单表按照客户号分组去重，结果集比较小可全内存。

分批读入客户文件，分段和内存中的客户号连接并求差集。

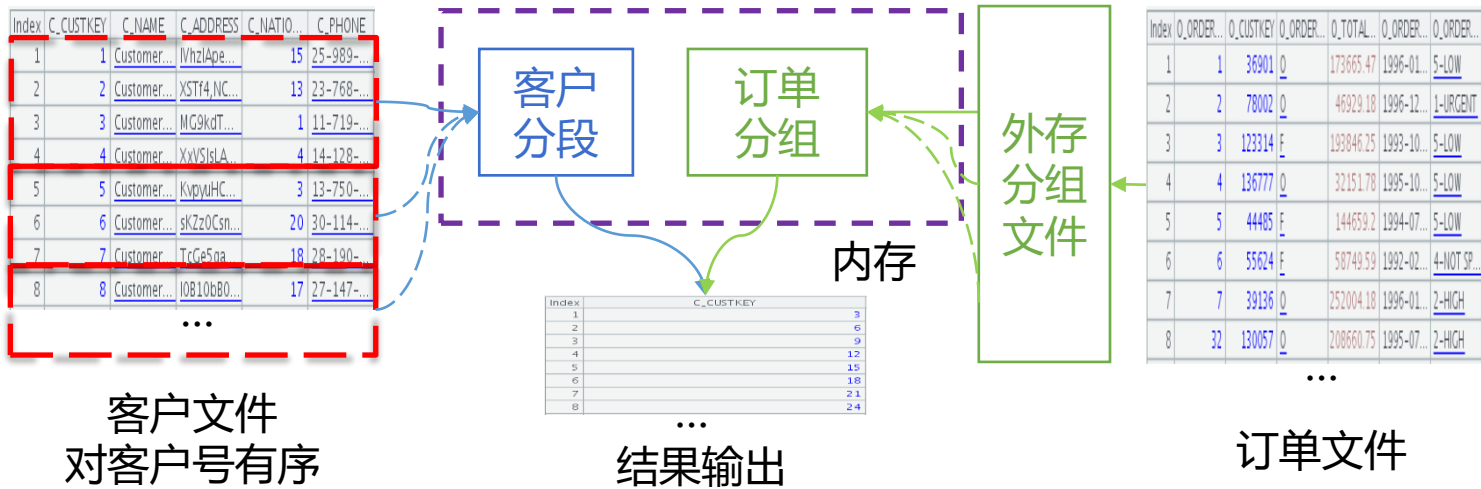


代码示例

	A	B
1	=file("ORDERS.btx").cursor@b(O_CUSTKEY)	/定义游标
2	=A1.groups(O_CUSTKEY)	/O_CUSTKEY去重
3	=file("CUSTOMER.btx").cursor@b(C_CUSTKEY)	/定义游标
4	=A3.join@d(C_CUSTKEY,A2:O_CUSTKEY)	/关联, @d为取差集
5	=A4.total(count(1))	/对游标汇总记录数

知识点-集合运算-差集

优化思路2：子查询结果较大，无法全内存



代码示例

	A	B
1	=file("ORDERS.btx").cursor@b(O_CUSTKEY)	/定义游标
2	=A1.groupx(O_CUSTKEY:C_CUSTKEY)	/分组去重
3	=file("CUSTOMER.btx").cursor@b(C_CUSTKEY)	/定义游标
4	=A3,A2].mergex@d(C_CUSTKEY)	/归并, @d为取差集
5	=A4.total(count(1))	/对游标汇总记录数

课堂练习p4.6-集合运算-差集

练习1： 执行p4.6.dfx，统计1996-01-01没有订单的客户个数。

要求： 根据订单表ORDERS的客户号 (O_CUSTKEY) 和客户表CUSTOMER的主键 C_CUSTKEY

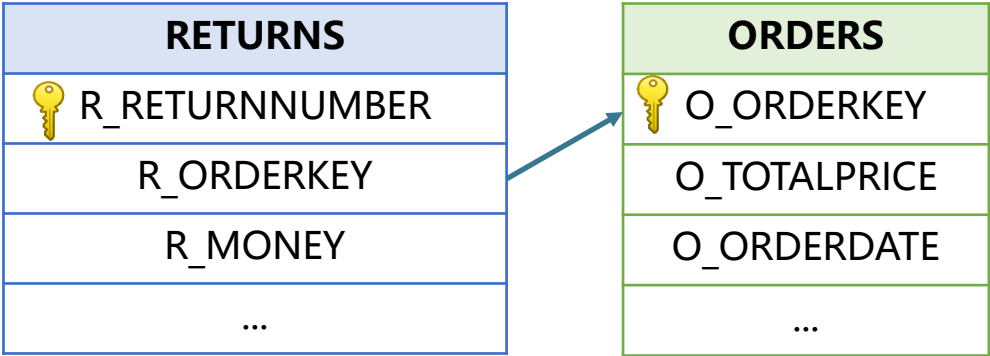
提示： O_CUSTKEY去重后结果较小，可放入内存用groups，差集用join@d()。

练习2： 改写p4.6.dfx，统计所有日期，都没有订单的客户个数。

提示： O_CUSTKEY去重后结果较大，不可放入内存用groupx，差集用mergex@d()。

知识点-集合运算-交集

订单金额超过一万，且回款小于五千的订单



计算要求

在订单表中找出金额超过一万的订单号；
在回款单中找出回款小于五千的订单号。
两者求交集。

代码示例

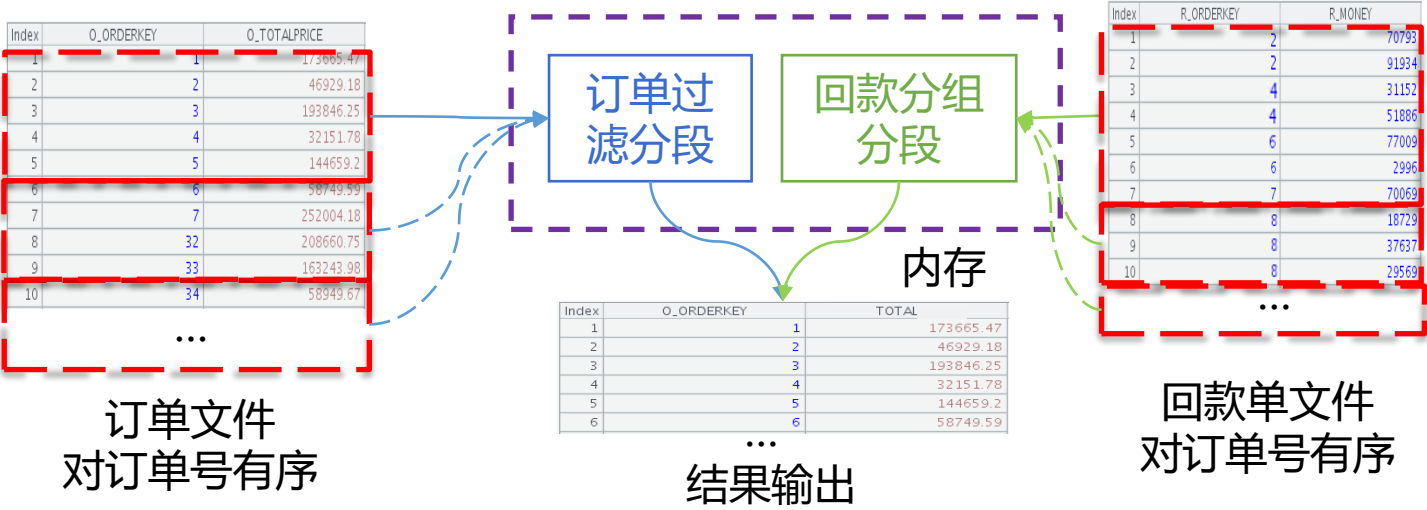
```
SELECT O_ORDERKEY FROM ORDERS
      WHERE O_TOTALPRICE > 10000
INTERSECT
SELECT R_ORDERKEY FROM RETURNS
      GROUP BY R_ORDERKEY HAVING SUM(R_MONEY) < 5000
```

SQL

知识点-集合运算-交集

优化思路：有序归并

订单和回款的过滤结果较大，无法全内存。



计算步骤

订单表和回款单分批读入内存，订单表过滤后与回款单分组后的结果，按照订单号有序归并，求交集。

代码示例

	A	B
1	=file("RETURNS.btx").cursor@b(R_ORDERKEY,R_MONEY)	/定义游标
2	=A1.group(R_ORDERKEY:O_ORDERKEY;sum(R_MONEY):TOTAL).select(TOTAL < 5000)	/分组去重
3	=file("ORDERS.btx").cursor@b(O_ORDERKEY,O_TOTALPRICE:TOTAL).select(TOTAL > 10000)	/定义游标
4	=[A3,A2].mergex@i(O_ORDERKEY)	/归并, @i取交集
5	=A4.total(count(1))	/汇总记录数

课堂练习p4.7-集合运算-交集

练习：写出p4.7.dfx，实现下面的SQL。



SQL:

```
SELECT O_ORDERKEY FROM ORDERS
      WHERE O_TOTALPRICE > 10000
INTERSECT
SELECT R_ORDERKEY FROM RETURNS
      GROUP BY R_ORDERKEY HAVING SUM(R_MONEY) < 2000
```

提示：订单表和退款表数据量较大，无法全内存，用有序归并（mergex@i）完成交集运算。

知识点-同表关联,EXISTS非等值条件

多供应商，仅有一供应商超时交货的订单

LINEITEM	
	L_ORDERKEY
	L_LINENUMBER
	L_SUPPKEY
	L_COMMITDATE
	...

计算要求

先要找出超时的在线商品记录，在其中找出有多个供应商的订单。
再排除掉其中有其他供应商超时交货的情况。
找到符合条件的订单后，按照供应商分组计数。

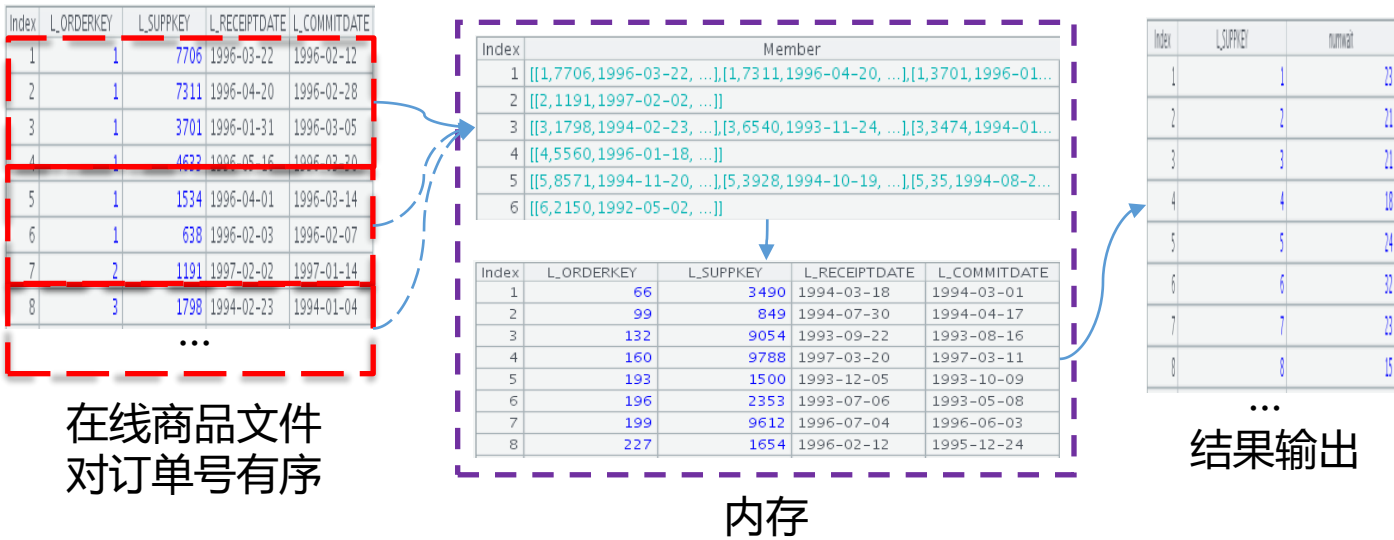
代码示例

```
SELECT L_SUPPKEY, COUNT(*) AS NUMWAIT FROM LINEITEM L1,
WHERE L1.L_RECEIPTDATE > L1.L_COMMITDATE
  AND EXISTS (SELECT * FROM LINEITEM L2 WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
              AND L2.L_SUPPKEY <> L1.L_SUPPKEY)
  AND NOT EXISTS (SELECT * FROM LINEITEM L3 WHERE L3.L_ORDERKEY = L1.L_ORDERKEY
                  AND L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)
GROUP BY L_SUPPKEY
```

SQL

知识点-同表关联,EXISTS非等值条件

优化思路：先分组，再组内过滤



计算步骤

在线商品数据按订单顺序存放，分批取出，按订单号有序分组但并不汇总，保留每组的在线商品。

循环每组订单判断是否有未按时交货的订单，是否有多个供货商，且是否只有一个供应商没有按时交货。

条件都满足的在内存中把分组重新合并成在线商品数据，再按照供应商号，小分组汇总，输出结果。

代码示例

	A	B
1	=file("LINEITEM.btx").cursor@b(L_ORDERKEY,L_SUPPKEY,L_RECEIPTDATE,L_COMMITDATE)	/在LINEITEM表所对应的集文件上定义游标,选出需要的列
2	=A1.group(L_ORDERKEY)	/对有序游标定义分组计算
3	=A2.conj((t=~.select(L_RECEIPTDATE>L_COMMITDATE),if(t.len())>0&&t.select@1(t(1).L_SUPPKEY!=L_SUPPKEY)=null&&~.select@1(t(1).L_SUPPKEY!=L_SUPPKEY)!=null,t,null)))	/选出每一组中未按时发货的订单给临时变量t, 如果t长度大于0且t中的供应商只有一个且此组中供应商有多个则返回t, 否则返回null, conj相当于group的逆操作
4	=A3.groups@u(L_SUPPKEY;count(1):numwait)	/对游标计算分组得到最终结果

课堂练习p4.8-同表关联, EXISTS非等值条件

练习：写出p4.8.dfx，实现下面SQL的计算。

SQL:

```
SELECT L_SUPPKEY, COUNT(*) AS NUMWAIT FROM LINEITEM L1,  
WHERE L1.L_RECEIPTDATE > L1.L_COMMITDATE  
      AND EXISTS (SELECT * FROM LINEITEM L2 WHERE L2.L_ORDERKEY = L1.L_ORDERKEY  
                  AND L2.L_SUPPKEY <> L1.L_SUPPKEY)  
      AND NOT EXISTS (SELECT * FROM LINEITEM L3 WHERE L3.L_ORDERKEY = L1.L_ORDERKEY  
                     AND L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)  
GROUP BY L_SUPPKEY
```

提示：

- 1、找出超时的记录 (L_RECEIPTDATE>L_COMMITDATE) ；
- 2、在1的条件下，找出有多个供应商的订单；
- 3、再基于2的条件下，排除掉其中有其他供应商超时交货的情况，最终找到符合条件的订单；
- 4、将这些订单，按照供应商分组计数。

知识点-子查询转换成连接-小结

子查询的特征

用子查询描述的 IN 都可以改用 EXISTS，等值 EXISTS 本质上是做连接，对于 `select * from A where exists (select * from B where ...)` 类型的问题，优化时要判断下面三个特征：



关联字段是否是各表的主键或者逻辑主键？



A、B表的规模,执行其它过滤条件后是否能载入内存？



如果都不能装入内存则要考察两个表是否按关联字段有序？

优化思路

子查询要按关联字段值唯一，如果不是逻辑主键则要先去重变为逻辑主键。

代码示例：用 `A.groups()` 去重。

如果有一个表能载入内存则可用内存连接方法。

代码示例：`cs.switch()`、`cs.join()`，选项 `@i`、`@d` 分别对应 `exists` 和 `not exists`。

如果两表都大不能载入内存则要考察两个表是否按关联字段有序，如果无序要先排序。

代码示例：`cs.sortx()` 排序，有序的两表可以用 `joinx()` 做连接。

第四章 多维分析

4.1 聚合

4.2 切片

4.3 数据路由

课前准备1-硬件和主目录

硬件 内存：8G以上 硬盘：空余30G以上 CPU：主频1G以上

主目录 在硬盘适当位置解压缩 “olap.zip”文件夹，并将集算器主目录设置成 “olap”文件夹
设置方法：集算器-菜单-工具-选项-主目录

课前准备2-生成数据文件

要求

上课前执行并读懂生成数据文件的脚本，回答三个问题：

A，数据文件数据量是多少？有哪些字段？

B，有哪些字段，生成后占用硬盘有多大？

C，有什么特征：有序、列存、行存、分段等等。

执行脚本

“主目录\dfx\orders.dfx”，生成订单集文件“主目录
\data\btx\orders.btx”和订单组表文件“主目录\data\ctx\orders.ctx”。

第四章 多维分析

4.1 聚合

知识点-全量预汇总

多维分析传统做法：全量预汇总

```
SELECT D1,..., SUM(M), ... FROM T GROUP BY D1,...
```

维度组合	count	sum	avg	max	min	...
D1,D2..Dn	✓	✓	✓	✓	✓	...
D2,D3..Dn	✓	✓	✓	✓	✓	...
D3,D4..Dn	✓	✓	✓	✓	✓	...
D4,D5..Dn	✓	✓	✓	✓	✓	...
...
Dn	✓	✓	✓	✓	✓	...

结论：

当n<=10时，且中间CUBE不太大的时候，才可以做全量预汇总

聚合运算转换为查找

预计算各种维度组合的汇总值后，预先保存，访问时直接用结果。
聚合运算变成查找运算，可利用索引迅速完成。

全量预计算占用空间太大

一个CUBE（数据立方体）有n个独立维度，维度组合有 2^n 种。
假设一个中间CUBE占1K空间（实际情况会远大于占1K），n=50时需要空间：
 $1K \times 2^{50} = 1048576 \text{ T}$ （超过1百万TB）
假定每次使用不超过20个维度（聚合、切片）：
 $1K \times (C(50,1)+C(50,2)+...+C(50,20))$ ，仍是天文数字。C(50,1)表示50个维度选1个。

知识点-全量预汇总

全量预汇总功能盲区

非常规聚合

如唯一计数、中位数、方差这些非常规聚合。都是常见需求，难以预测，且无法用其它聚合值计算。

条件测度

统计金额大于1000的订单的总金额。
统计销量大于50的订单的总金额。

组合聚合

组合太多，无法预测，如：

月平均销售额的TOP3 = $\text{top}(3, \text{按月求和}(\text{销售额})/\text{count}())$

月销售额中位数的MAX = $\text{max}(\text{按月排序}(\text{销售额})(\text{count}() + 1/2))$

时间段统计

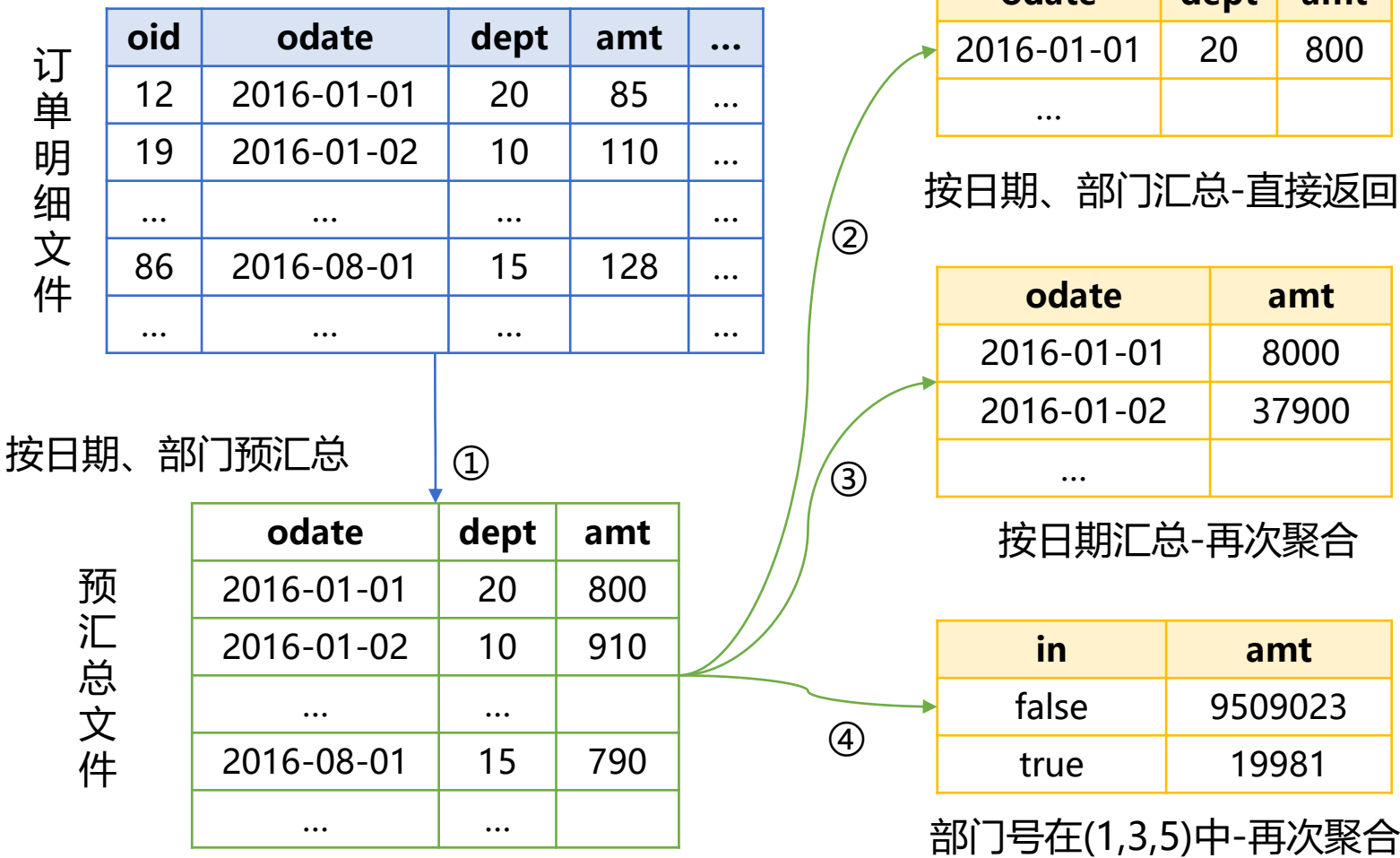
统计2018-6-10到2018-6-20期间的销售总额。

维度D在(d,...)范围内

(d,...)的组合太多，无法全部预汇总。

知识点-部分预汇总

部分预汇总



部分维预汇总常规聚合

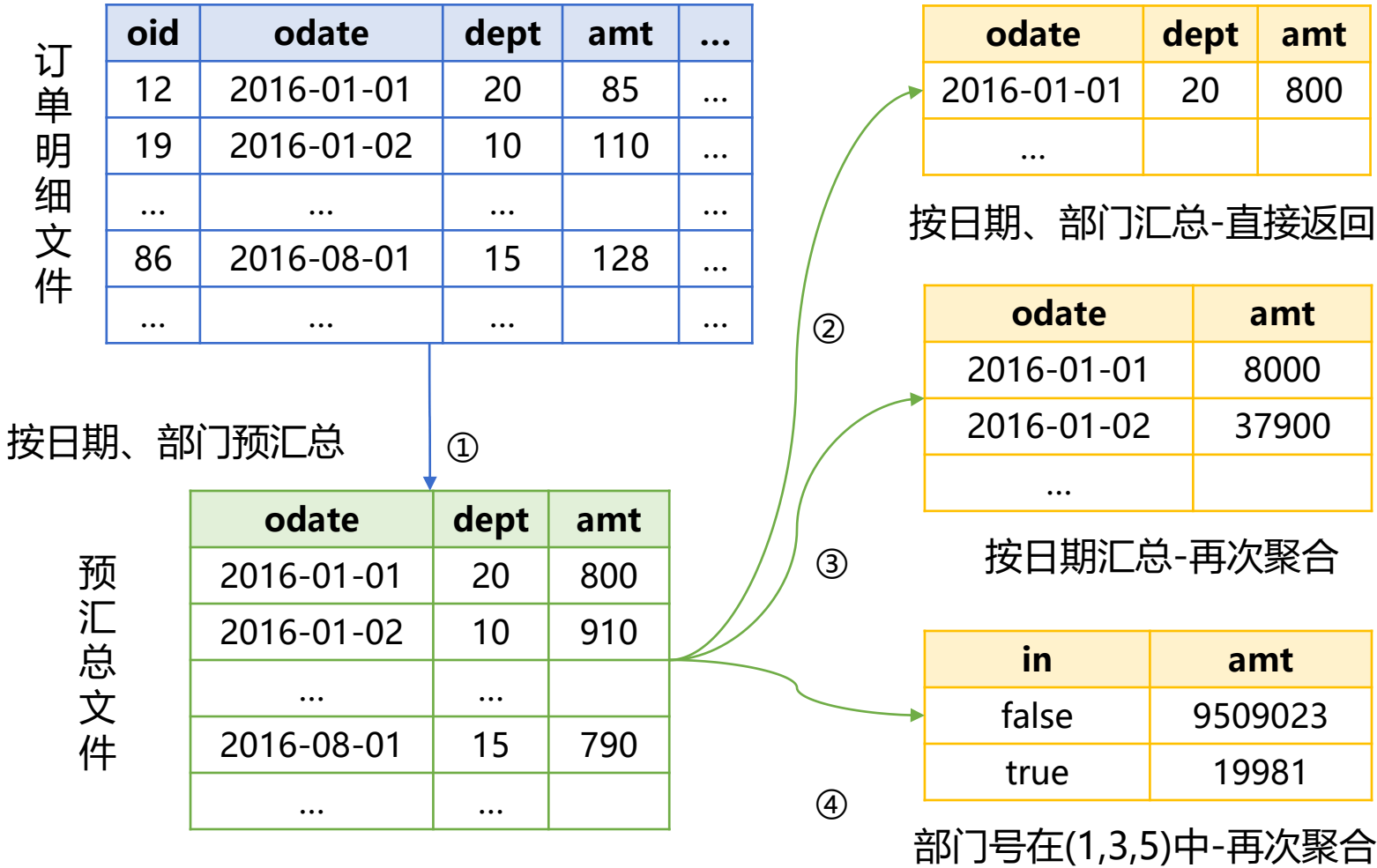
对部分常用维度及组合计算汇总并事先保存，以后查询时可以直接返回结果。

再次聚合汇总

部分常用维度及组合计算的汇总值如果不能直接得到需要的结果，可基于这些保存过的中间结果再聚合汇总。

代码示例-部分预汇总

部分预汇总



代码示例

- ①
- | A | |
|---|--|
| 1 | =ordersFile.open() |
| 2 | =A1.cuboid(cu_1,odate,dept;sum(amt):amt) |
- ②
- | A | |
|---|--------------------------------------|
| 1 | =ordersFile.open() |
| 2 | =A1.cgroups(odate,dept;sum(amt):amt) |
- ③
- | A | |
|---|---------------------------------|
| 1 | =ordersFile.open() |
| 2 | =A1.cgroups(odate;sum(amt):amt) |
- ④
- | A | |
|---|--|
| 1 | =ordersFile.open() |
| 2 | =A1.cgroups([1,3,5].contain(dept):in;sum(amt):amt) |

课堂练习p1.1-部分预汇总

练习：打开p1.1.dfx，观察三个分组汇总，记录执行时间。

单位：秒	基于明细计算	部分预汇总
两字段分组		
单字段分组		
包含分组		

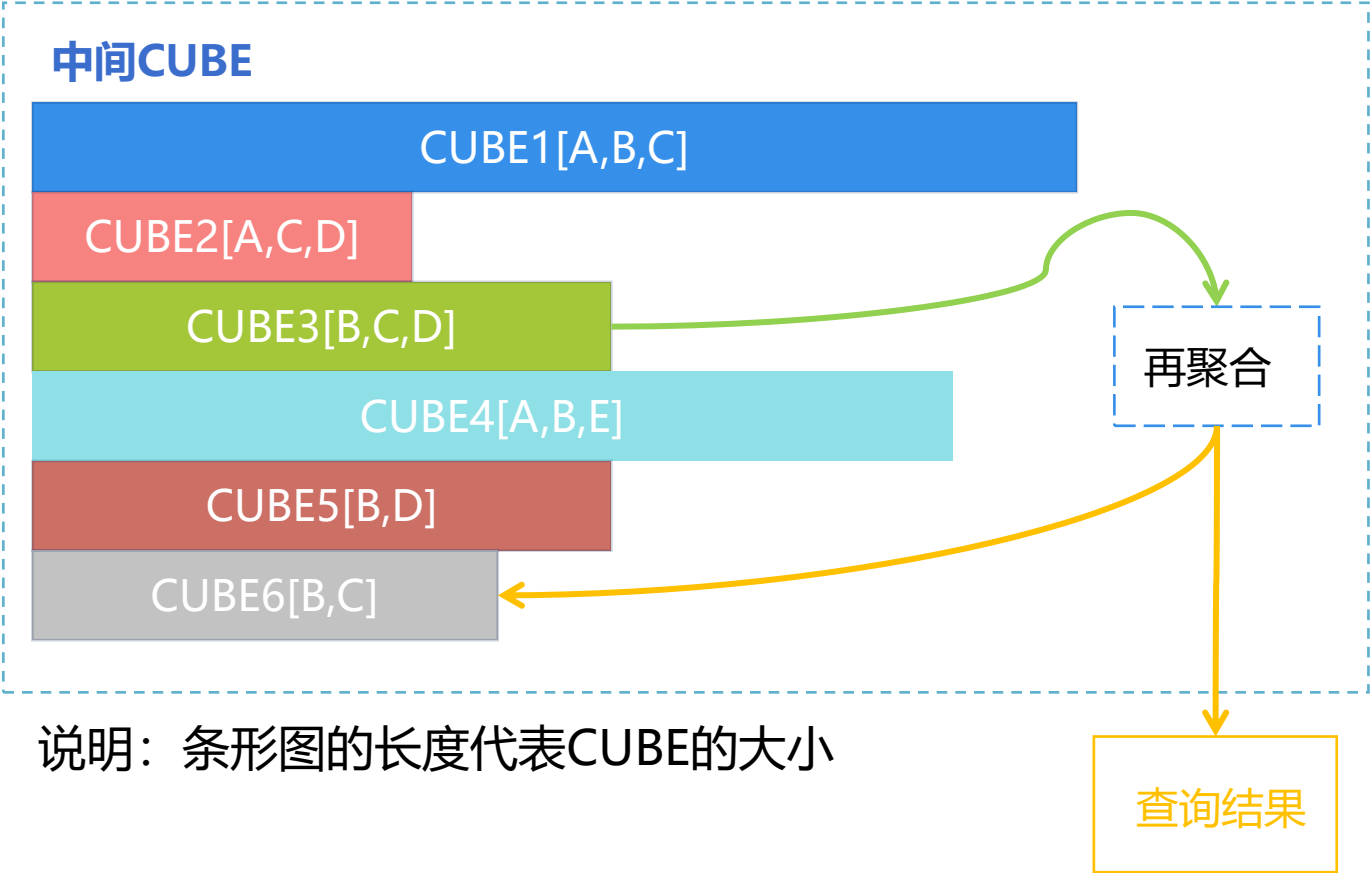
练习：改写p1.1.dfx，改成按照orderdate，custid部分预汇总再分组汇总，记录执行时间。

补充阅读 现有的多维分析后台，多采用全部预汇总的方式。如果汇总的维度较多，就会占用大量的硬盘。所以，实际应用中只能选择尽可能少的维度，往往无法满足客户需求。

知识点-部分预汇总

部分预汇总多个中间结果选择

需求：对维度[B,C]进行聚合



多个中间结果再汇总

再次聚合汇总时，若中间结果集有多个，则选择数据量最小的。
例如：要对维度[B,C]进行聚合，可以利用的预汇总立方体有2个：
CUBE1的汇总维度是[A,B,C]
CUBE3的汇总维度是[B,C,D]
两个都可以对[B,C]再聚合，要选择更小的CUBE3。
集算器自动实现中间结果的选择。

输出结果同时保存

输出结果如果以后还要用，可保存到硬盘，命名为CUBE6。
集算器无法判断是否需要保存，需要写代码手工实现。

知识点-时间段预汇总

时间维的聚合计算

计算时间段的销售总金额

odate	amt	...
.....		
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	
.....		

按日期汇总文件

预汇总

汇总结果

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

按月预汇总文件

需求说明

按日期汇总文件是2018年每天的销售额。要计算6月9日到11月11日的总金额。

时间段预汇总计算步骤

预先按照月份汇总，建中间立方体-按月预汇总文件。
将日期段划分为3段：
6月19日-6月30日，12条日金额数据。
7月-10月，4条月金额数据。
11月1日-11月11日，11条日金额数据。
3段汇总数据共27条，而直接汇总明细需要计算145条。
计算量减少5倍多。

代码示例-时间段预汇总

时间维的聚合计算

计算时间段的销售总金额

odate	amt	...
.....		
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	
.....		

按日期汇总文件

预汇总

汇总结果

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

按月预汇总文件

代码示例

部分预汇总

	A
1	=ordersFile.open()
2	=A1.cuboid(month(odate);sum(amt))

时间段汇总

	A
1	=ordersFile.open()
2	=A1.cgroups(;sum(amt); odate>=date("2018-6-19") && odate<=date("2018-11-11"))

知识点-时间段预汇总

使用中间CUBE进行时间段预汇总的原则

时间段再聚合的条件

查询条件是时间段，不一定能再聚合。如[2018-9-15,2018-10-18]时间段内不存在整年或整月，无法再聚合。

多层级中间数据

可利用的中间CUBE可能有多个层级。如[2016-11-11,2018-2-14]时间段，可利用2017整年的数据CUBE；也可利用2016年12月和2018年1月的整月数据的CUBE。

日期再汇总

中间CUBE的汇总维度是[年月日]的，可以再汇总为[年月]或[年]。

数据更新

原表数据更新的时候，也要同步更新中间CUBE的汇总结果，保证数据同步。
集算器可自动实现数据追加的同步更新。删、改和插入需要手工重建CUBE。

课堂练习p1.2-时间段预汇总

练习：打开p1.2.dfx，观察某个时间段内的订单金额汇总，记录执行时间。

单位：秒	执行时间
按明细分组	
预汇总分组	

练习：改写p1.2.dfx，改成按照orderdate部分预汇总再分组汇总，记录执行时间。

第四章 多维分析

4.2 切片

知识点-布尔维序列解决切块

按照客户年龄段切块

年龄段在30-35岁或者35-40岁...

序号	custid	name	ages	...
1	011	James	AGE12	
2	012	Luke	AGE10	
3	013	Tom	AGE11	
...		

客户数据文件

序号	code	value
1	AGE10	25-30岁
2	AGE11	30-35岁
3	AGE12	35-40岁
...

年龄段代码表
agesCode

切块的本质是IN计算

在年龄段代码表，可以查到满足条件的年龄段代码。

切块问题最终是要计算ages in (AGE11,AGE12...).

IN计算的性能问题

IN计算要在客户数据文件中找n个ages值。如果客户数据文件中的ages无序，那么查找需要遍历整个文件，性能较差。如果ages有序，用二分法会提速，但是要查找多个值，也还是很慢。

而且，计算的耗时和n有关系，n较大时性能会很差。

知识点-布尔维序列解决切块

按照客户年龄段切块

ages in ("AGE11", "AGE12"...)

序号	custid	name	ages	...
1	011	James	3	
2	012	Luke	1	
3	013	Tom	2	
...		

客户数据文件

年龄段代码表agesCode

序号	code	value
1	AGE10	25-30岁
2	AGE11	30-35岁
3	AGE12	35-40岁
...

序号	成员
1	False
2	True
3	True
...	...

布尔维序列

代码示例

	A	B
1	=agesCode(["AGE11","AGE12"...].contain(code))	/用IN条件和年龄段代码表生成布尔序列
2	=customerFile().cursor(;A1(ages))	/用布尔序列完成切片计算

布尔维序列用于IN计算

预处理：将客户文件中的年龄段ages字段，转换成年龄段代码表的序号。

查询时：根据IN条件和年龄段代码表生成布尔维序列。

分段遍历客户文件。第1条记录，用ages字段的值3，找到布尔维序列中第3个成员，值为true，因此第1条客户记录满足条件。

如果值为false，则不满足条件。其他记录以此类推。

布尔维序列性能优势

布尔维序列将值比较转换为序号引用，有效的减少了计算时间。

用于IN计算，计算时间和IN枚举值的多少无关，不会随着IN枚举值的增加而增加。

课堂练习p2.1-布尔维序列解决切块

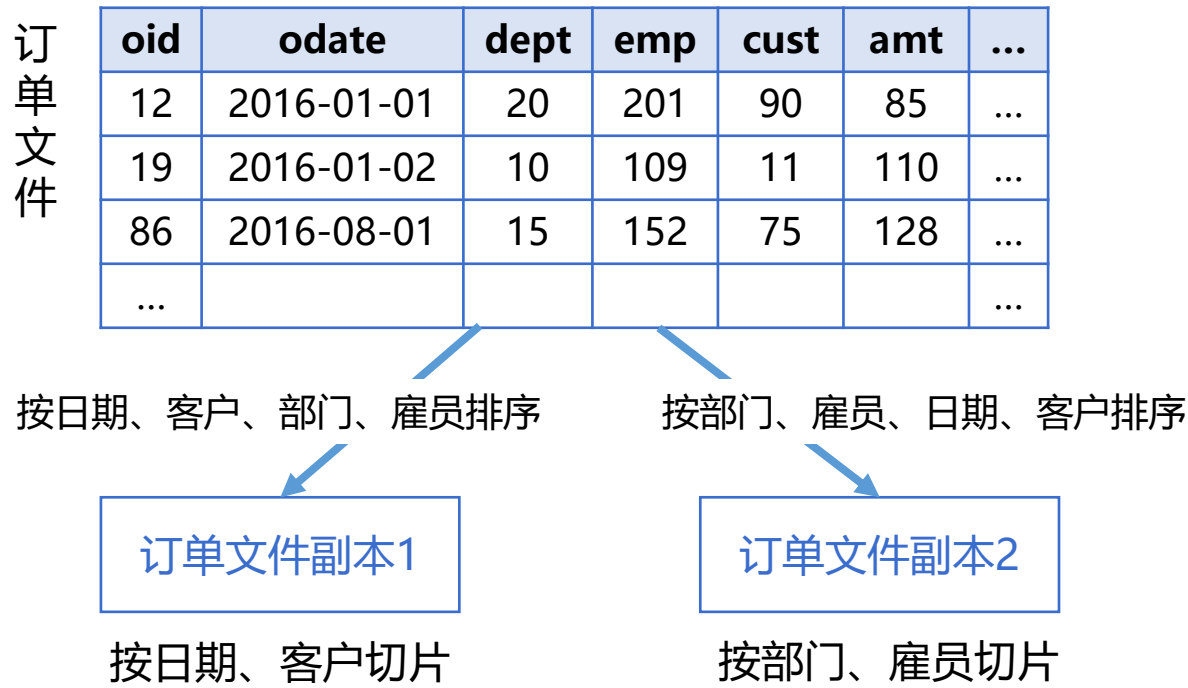
练习：打开p2.1.dfx，用普通方法计算订单按照客户切块的结果，记录执行时间

	执行时间（毫秒）
普通方法	
布尔维序列	

练习：续写p2.1.dfx，增加用布尔维序列的方式计算切块的代码，记录执行时间

知识点-冗余排序

按照订单日期、客户切片和按照部门、雇员切片



冗余排序

预处理：将订单文件分别按照图中的两种方式排序，保存为订单文件的两个副本。

查询时：根据切片的不同要求，选择副本1或者副本2计算。

冗余排序可以让有序数据尽量连续，减少遍历时的数据跳跃，从而达到提高切片计算速度的目的。

代码示例

	A	B
1	if sliceType==1	=orderFile1.cursor(;odate==... && cust==...)
2	else sliceType==2	=orderFile2.cursor(;dept==... && emp==...)

知识点-冗余排序-索引

按照客户切片和按照雇员切片

订单文件

oid	odate	dept	emp	cust	amt	...
12	2016-01-01	20	201	90	85	...
19	2016-01-01	21	109	11	110	...
86	2016-01-05	15	152	75	128	...
...						...

按日期、部门、雇员、客户排序

索引

订单文件按照日期、部门、雇员、客户排序后，按照日期是完全有序的，切片最快。后面的字段是不完全有序，按照后面的字段切片会变慢。

如果希望获得更高的性能，可以用索引。

排序字段越多，后面字段的索引效果越好。

代码示例

建索引

	A
1	=orderFile.open()
2	=A1.index(id_2,dept)
3	=A1.index(id_8,emp)

查找

	A
1	=orderFile.open()
2	=A1.icursor(;dept>100,id_2)
3	=A1.icursor(;emp==599,id_9)

课堂练习p2.2-冗余排序

练习：打开p2.2.dfx，两种切片计算都用一个订单组文件执行，记录执行时间

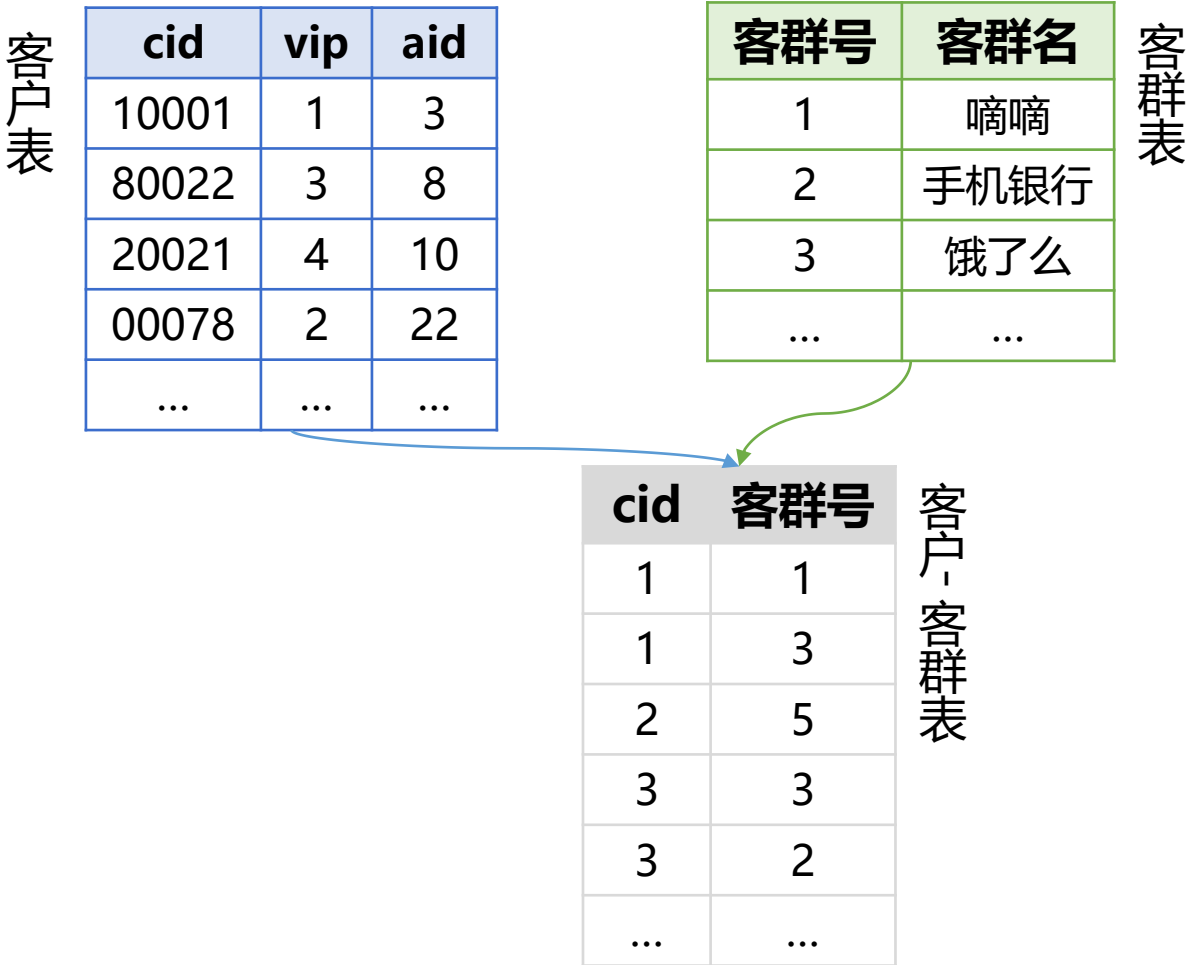
	执行时间（毫秒）
普通方法	
冗余排序	

练习：改写p2.2.dfx，按照按日期、客户切片采用orders1.ctx（排序方式是：按日期、客户、部门、雇员排序）；按照部门、员工切片采用orders2.ctx（排序方式是：按日期、客户、部门、雇员排序）。记录执行时间

提示：用for循环+if else 判断。

知识点-二值维度（是否型）

客群求交集并按条件过滤-传统做法
计算多个客群交集的客户，并按维度过滤



传统数据结构和计算
客户和客群的多对多关系，用三个表描述。
计算时将客户表和客户-客群表关联计算。

性能分析
客户表可以有几千万，甚至上亿客户。
每个客群可以拥有几百万上千万客户。几千个客群的总客户数量非常庞大。
客户表和客户-客群表都比较大，两个大表关联，性能较差。

知识点-二值维度（是否型）

客群求交集并按条件过滤-按位存储

计算多个客群交集的客户，并按维度过滤数

客户表

cid	vip	aid
10001	1	3
80022	3	8
20021	4	10
00078	2	22
...

客群表

客群号	客群名
1	滴滴
2	手机银行
3	饿了么
...	...

cid	vip	aid	1to8
10001	1	3	AAh
80022	3	8	55h
20021	4	10	AAh
00078	2	22	55h
...	

1	2	3	4	5	6	7	8
1	0	1	0	1	0	1	0

客户-客群-按位存储

用整数存储位数据

假设共有8个客群，用8位二进制数表示为10101010，表示客户属于第1、3、5、7个客群。

在字段1to8中，位数据用整数存储，例如10101010，存储为十六进制整数AAh。

用位数据计算

要统计第1、3客群共同的客户，可用10100000（A0h）和1to8字段做按位与计算，结果还是A0h的，满足条件。

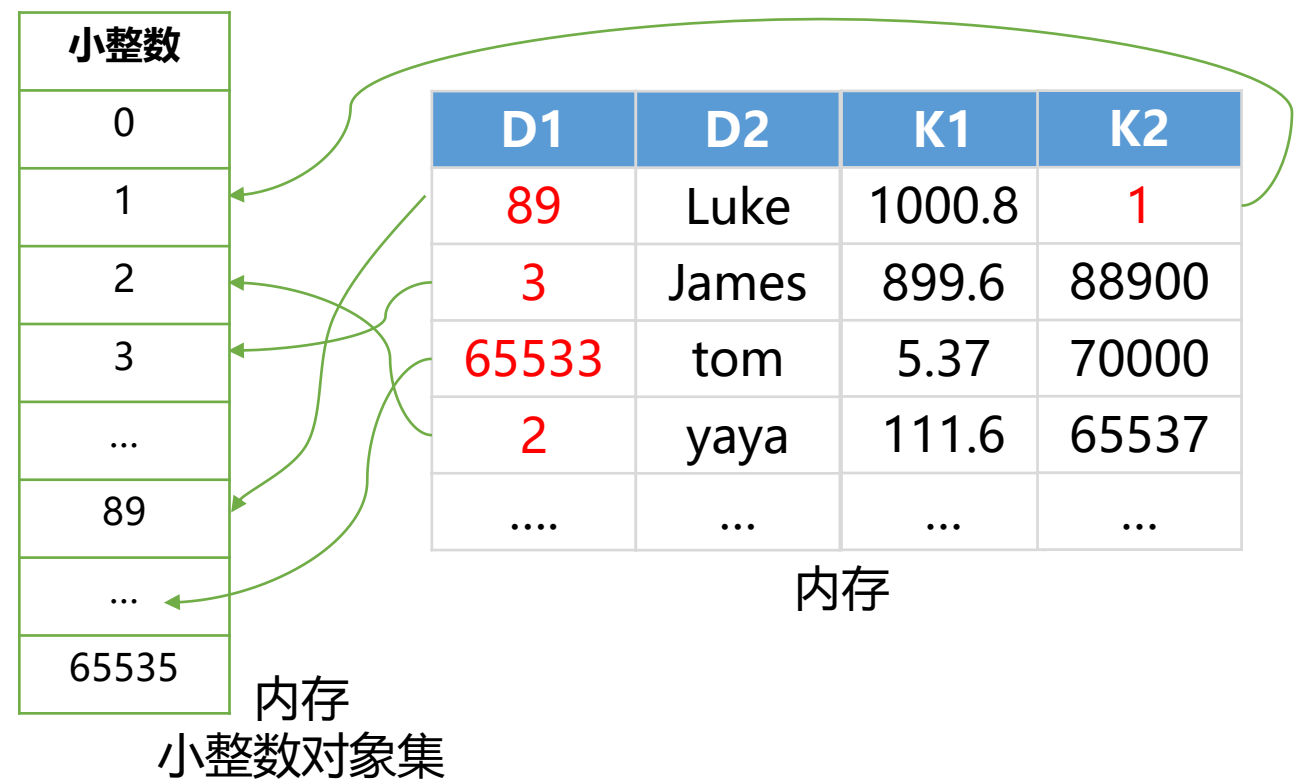
再结合前述的维度序号化方式，计算维度过滤，性能可以大幅度提升。

代码示例

	A	B
1	=file1.cursor()	/打开文件
3	=A1.select(and(1to8,0xA0)==0xA0)	/1、3客群交集

知识点-预处理为小整数

维值、测度值预处理为小整数



说明：小整数对象集应采用后台自动处理的方式。

预先准备小整数对象

在内存中预生成所有的小整数对象集，从1到65535。

维度、测度值转换为引用

维度值大多数是小整数，测度也有一部分是小整数，都可转换为小整数对象集中成员的引用。

例如：D1的取值都是小整数，内存中不再新建小整数对象，改为直接引用小整数对象集的一个成员。K2的部分取值是小整数的，也可以改为引用。

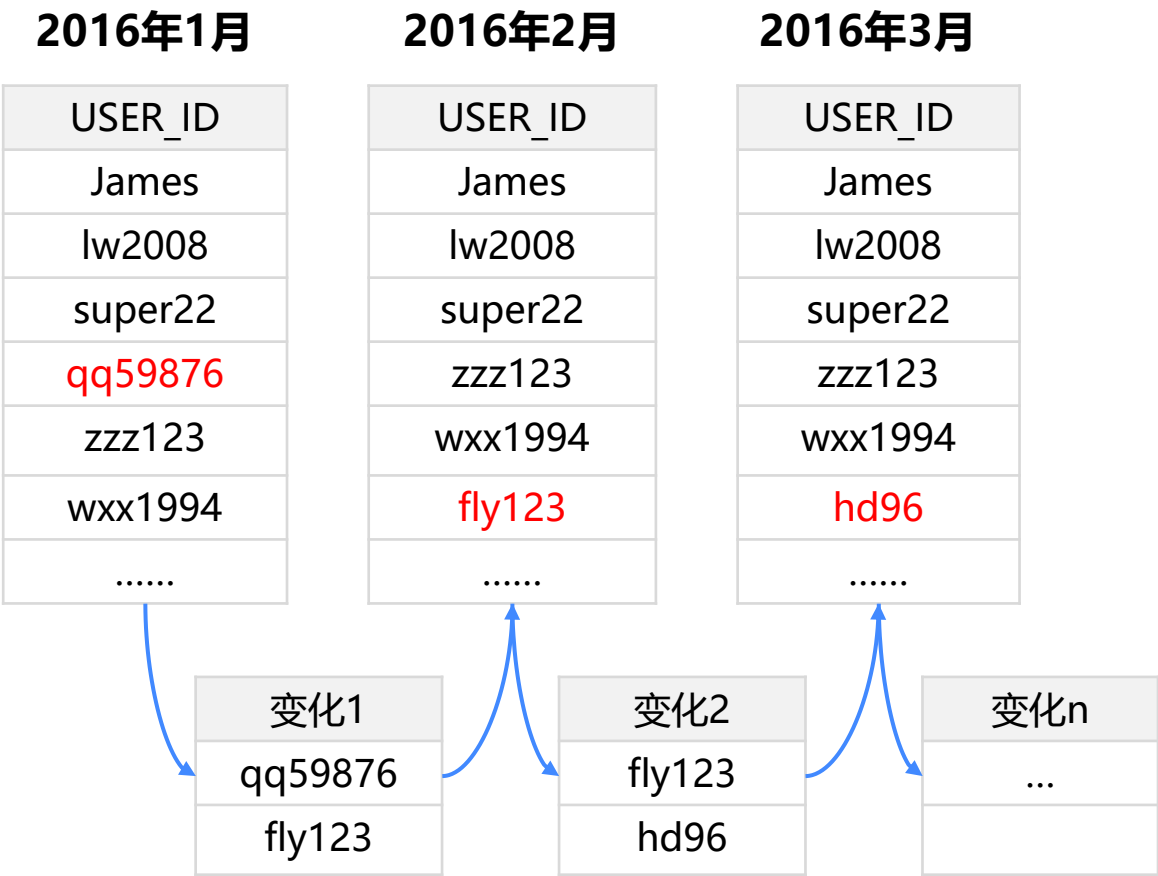
值转为引用的性能提升

对Java应用来说，可减少对象创建，提高文件读入内存的速度。

小整数对象共享，可节省内存。

知识点-随时间变化小的标签数据

随时间变化小的标签数据



仅保存变化部分

不再保存每个月的完整数据，只保存1月数据和以后每个月的变化数据。

性能提升

计算时无须装载每月的历史数据，只装载1份和后续变化部分即可，其他月份可计算得到。
2月 = 1月数据 XOR 变化1
3月 = 1月数据 XOR 变化1 XOR 变化2
n月 = 1月数据 XOR 变化1 XOR ... XOR 变化(n-1)

代码示例

	A	B
1	=data_201601.import@i(USER_ID)	/装载2016年1月数据
2	=data_change.cursor().fetch(5)	/5个月的变化数据
3	=A1.insert(0,A2)	
4	=A3.xunion()	/计算6月份数据

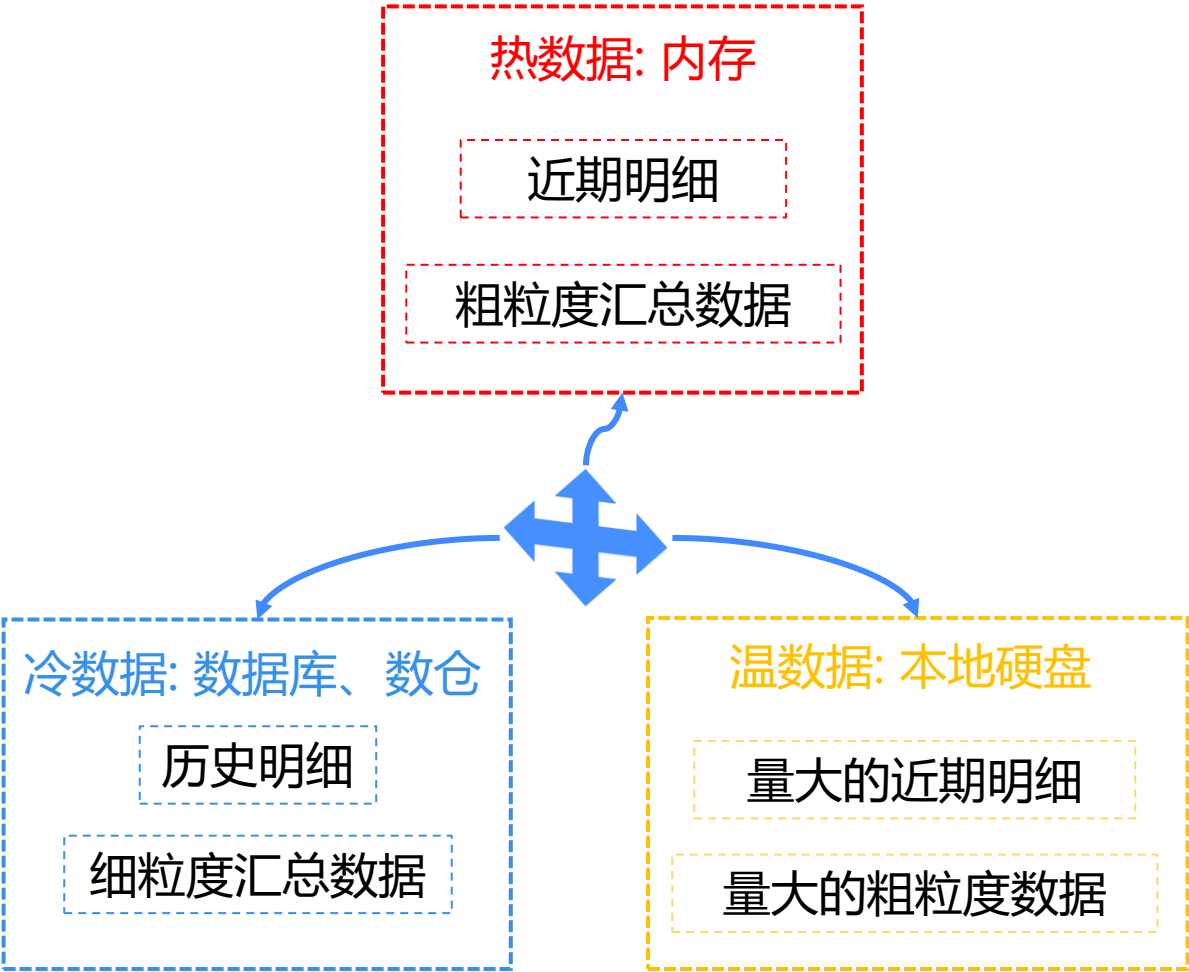
第四章 多维分析

4.3 数据路由

知识点-数据路由

数据计算路由

查找某日订单



冷热数据分层存储

根据数据温度采取分层策略：频繁、高并发访问的热数据前置，海量的低频访问的冷数据后置，通过数据路由控制。

例如：2020年的最新订单是热数据，装入内存；2017-2019年数据是温数据，放入本地硬盘存储；2017年之前的冷数据放到数据仓库中存储。

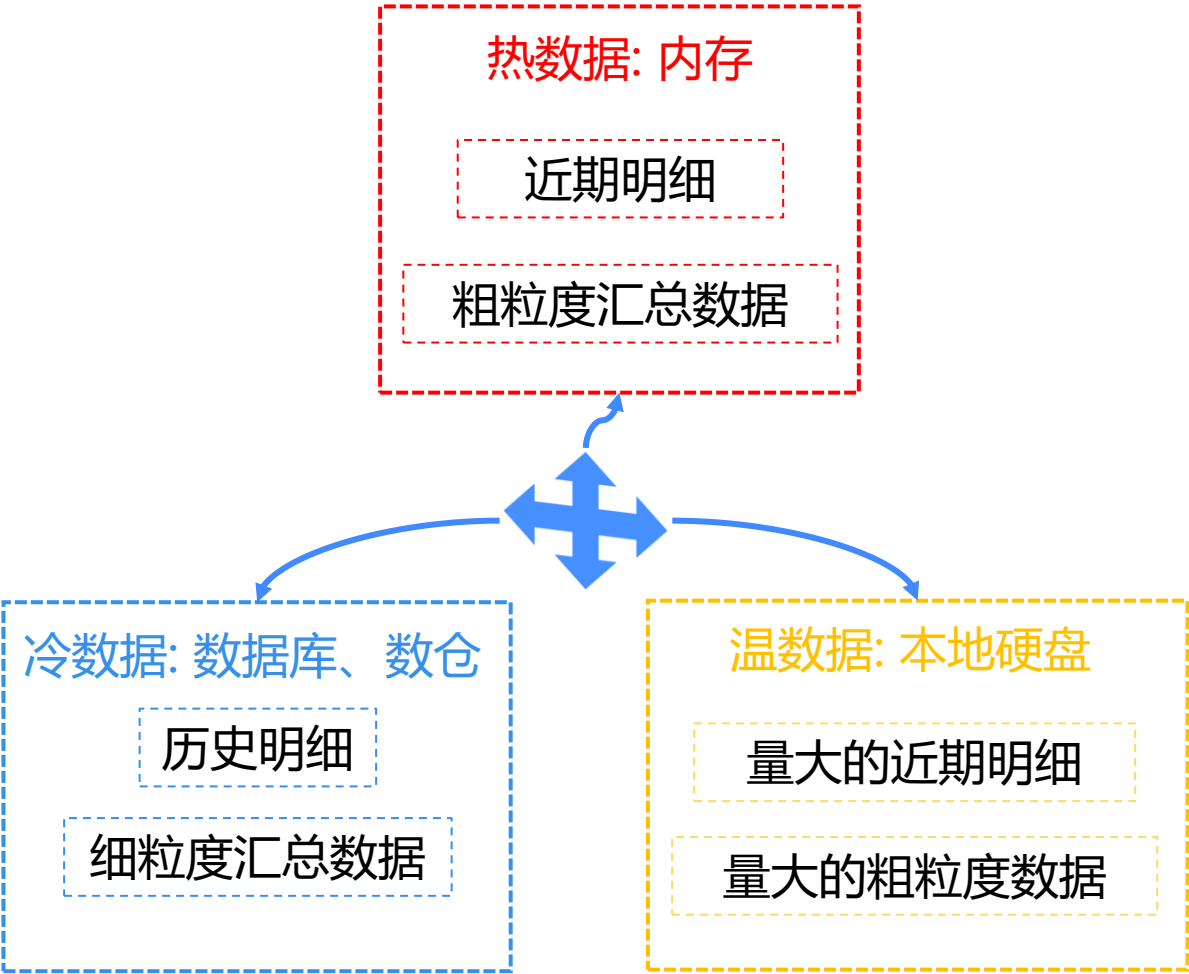
冷热数据路由

根据数据的时间维度、汇总的粒度等来决定使用冷数据还是热数据。

代码示例-数据路由

数据路由

查找某日订单



代码示例

数据准备，最新数据装入内存

	A
1	<code>=env(hotData, file("order2020.btx").import@b())</code>

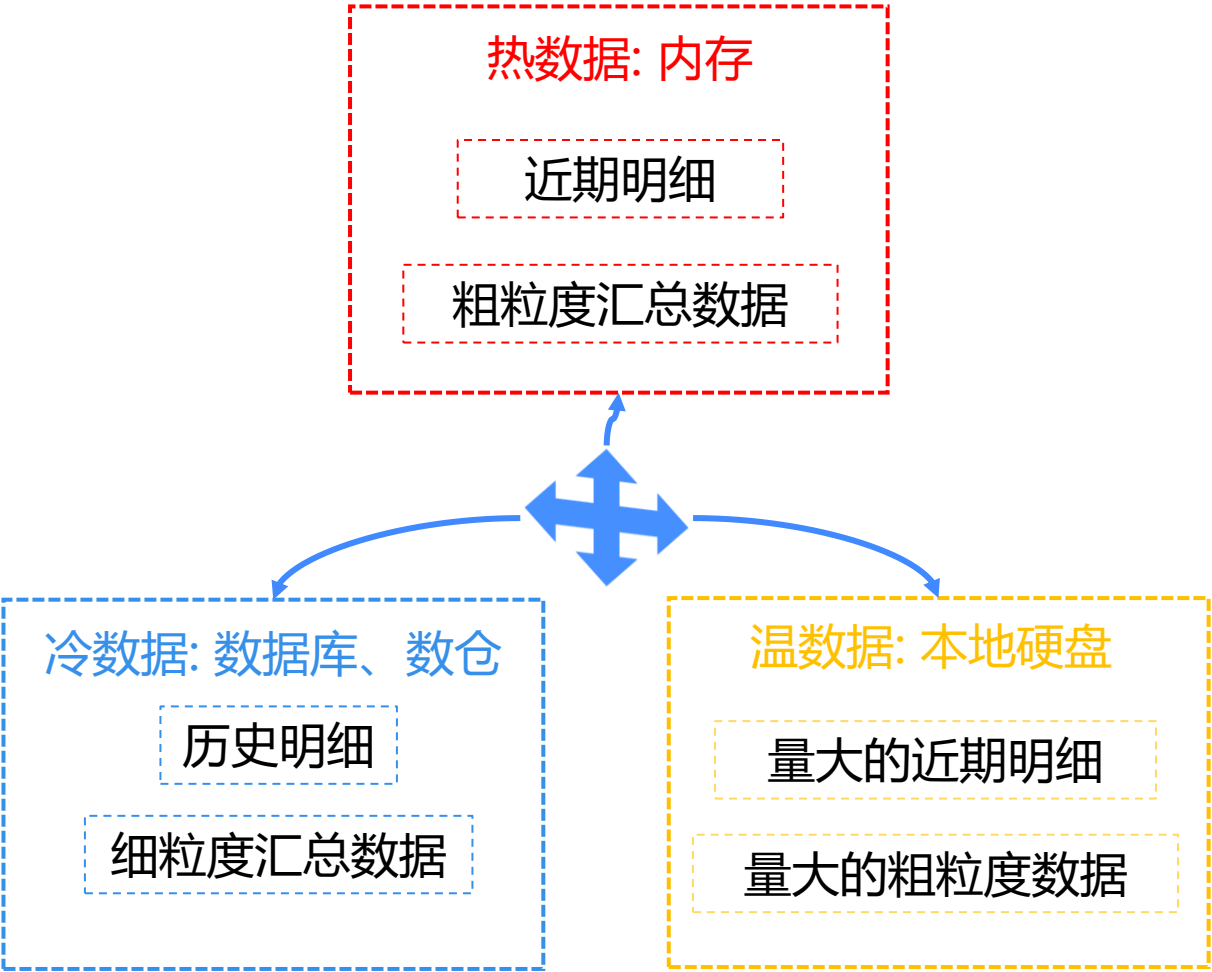
查询时根据输入的日期选择内存中的热数据，或者是文件中的冷数据。

	A	B
1	<code>if(year(indate) == 2020)</code>	<code>=hotData.select(odate == indate)</code>
2	<code>else</code>	<code>=file("data_history.ctx").open()</code>
3		<code>=B2.cursor(; odate == indate).fetch()</code>

说明：indate是传入参数，日期型。

代码示例-数据路由

数据路由 汇总订单金额



代码示例

数据准备，预先做好的年汇总装入内存

	A
1	<code>=env(yearData, file("yearDate.btx").import@b())</code>

查询时根据输入的数据类型选择内存中的热数据年汇总，或者是文件中的冷数据。

	A	B
1	<code>if(ifdate(input))</code>	<code>=file("data_history.ctx").open()</code>
2		<code>=B1.cursor(;odate==input)</code>
3	<code>else</code>	<code>=yearDate.select(year==input)</code>

说明：input是传入参数，如果是日期型则查询某日数据；如果是数值型则查询某年汇总数据。

课堂练习p3.1-数据路由

练习：打开p3.1.dfx，用普通方法查找1982-09-11的订单，并求和金额之和,记录执行时间

	执行时间（毫秒）
普通方法	
数据路由	

练习：改写p3.1.dfx，将1982年9月数据作为热数据装入内存，其他数据作为温数据，用硬盘存储。同样查找1982-09-11的订单金额之和，记录执行时间

提示：使用year()和month()判断年份和月份，用env()加载内存全局变量。

综合练习一 高并发有关联查询

1 需求描述

2 解题思路

3 动手练习

4 延伸思考

准备工作-硬件和主目录

硬件 内存：8G以上 硬盘：空余30G以上 CPU：主频1G以上

主目录 在硬盘适当位置解压缩 "practice1.zip"文件夹，并将集算器
主目录设置成"practice1"文件夹
设置方法：集算器-菜单-工具-选项-主目录

综合练习一 高并发有关联查询

1. 需求描述

需求概述



访问人数特别多

几百万、上千万人访问

不能让用户等待

手机、网页一秒之内要看结果

数据总量超大

海量账户多年数据，以亿为单位

海量用户会带来高并发查询，例如：网上银行、手机银行、手机营业厅、游戏账户等等。如何保证秒级的查询速度？

练习场景：客户查询一年内活期明细

海量客户

十万客户

单个客户

每个客户平均100条明细

数据总量

一千万条活期明细

关联维表

关联银行分支机构表
500记录

客户每次只查询自己的数据，一年活期储蓄明细数量几条到几百条。因为数据总量较多，高并发时做到秒级响应是个挑战。何况还要关联机构表。

练习场景：计算目标

活期明细表detail

一千万数据
十万个custid

did	corpid	custid	amt
1	7	1011	2345.8
2	110	1001	100
3	245	9090	1090
...

分支机构表corp

五百条数据

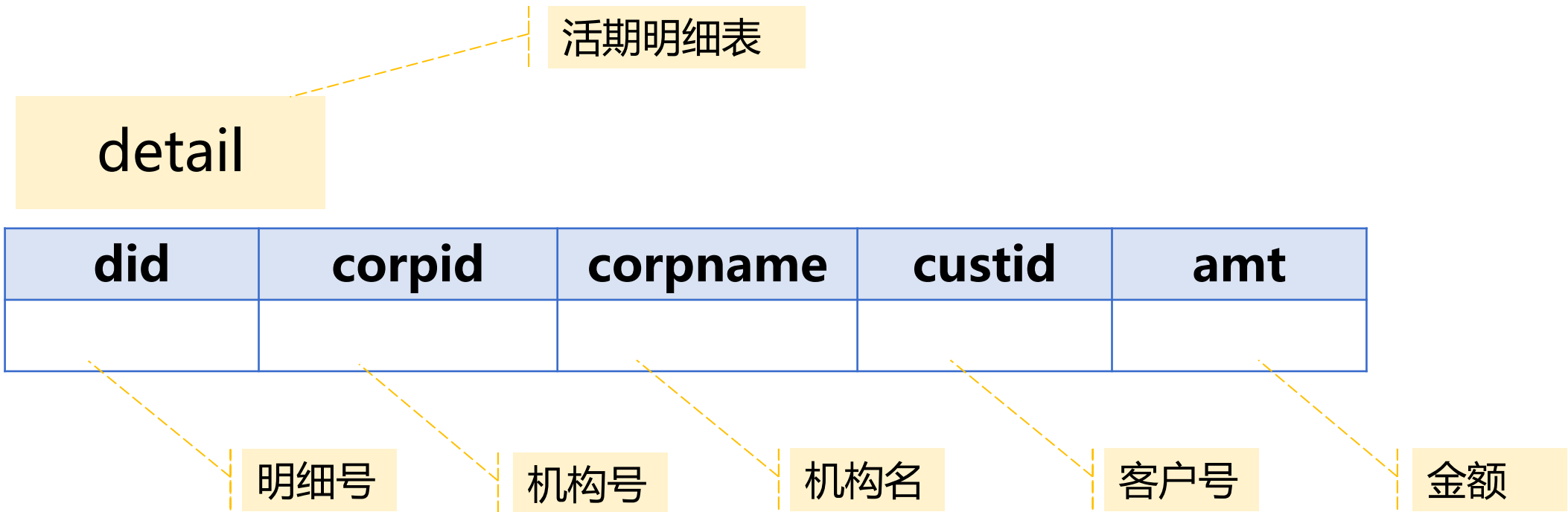
corpid	corpname
1	北京分行
2	上海分行
3	南京分行
...	...

SQL
示例

```
select did,c.corpname,custid,amt from detail d left join corp c
on d.corpid=c.corpid where d.custid=1011
```

活期明细表和分支机构表通过corpid关联，要进行大并发查询，查询条件是custid。

对比方案一：关系数据库宽表冗余



did是明细号，corpid是机构号，amt是金额。corpname是分支结构名称，也保存在活期明细表中。

对比方案一：关系数据库宽表冗余的问题

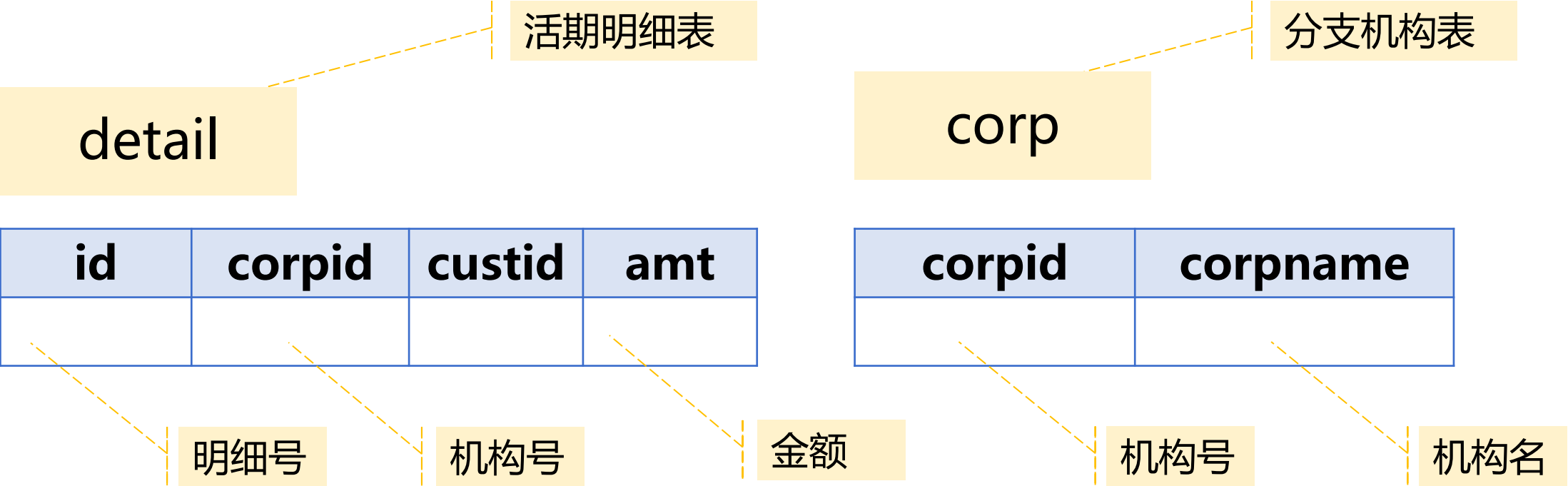
```
select did,corpid,corpname,amt from detail where custid= ' 10100'
```

did	corpid	corpname	amt
10101	010110	涿县支行	2389.00

did	corpid	corpname	amt
10101	010110	涿州市支行	2389.00

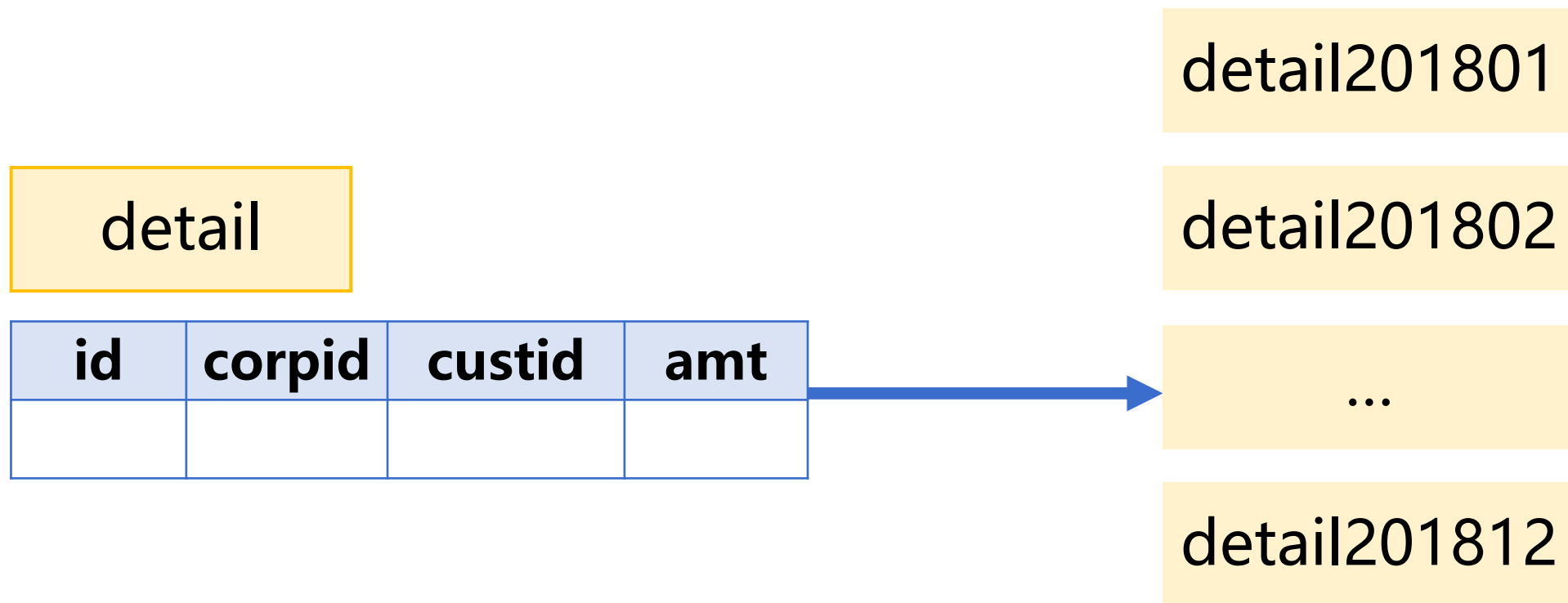
当冗余的分支机构名称发生改变的时候，宽表全部要更新。类似的冗余情况很多，发生变化的情况很多，更新耗时巨大。

对比方案二：关系数据库两表关联



活期明细表和分支机构表通过字段corpid分支机构号关联。

对比方案二：关系数据库两表关联-分表索引



一年活期明细一千万条，若用数据库单表存放，数据量过大，建索引时间非常长。所以一般都要按月分表存放，每个分表建立custid字段的索引。

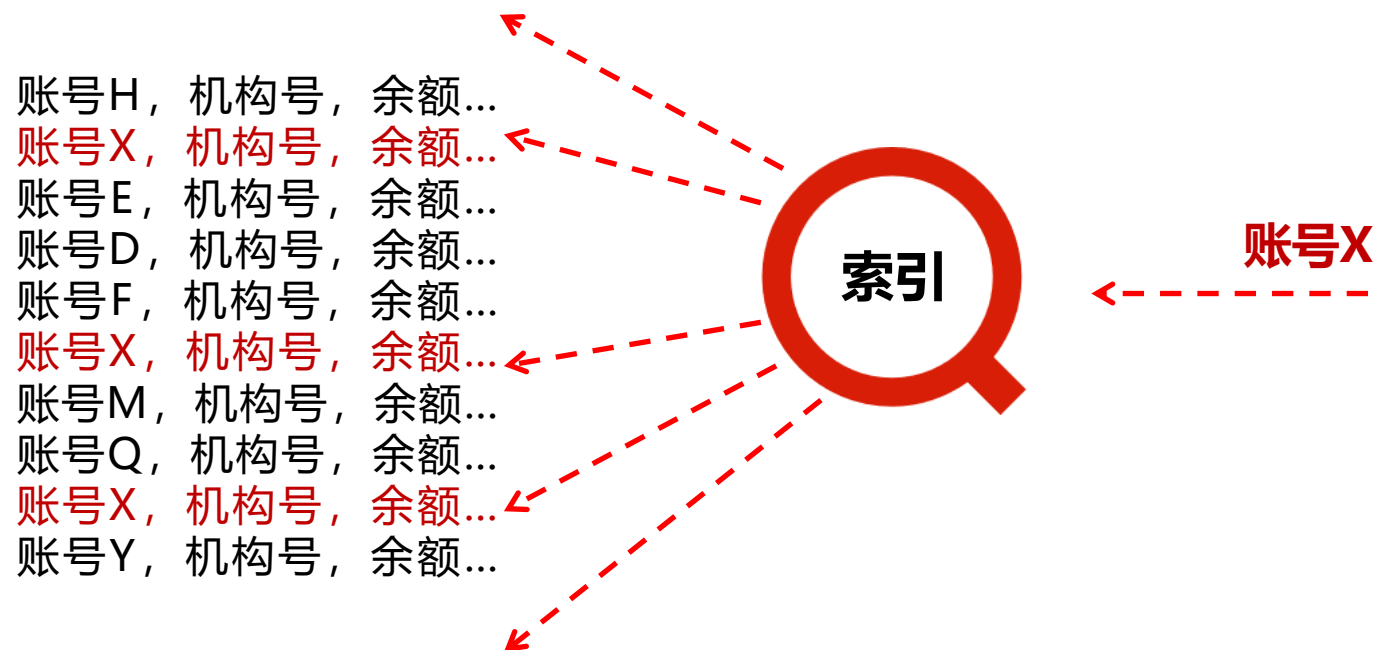
对比方案二：关系数据库两表关联-分表索引

```
create index index_detail_201801_id on detail_201801(custid)
```

```
select dd.id,dd.corpid,dd.amt, c.corpname from  
(select d.id,d.corpid,d.amt from detail_201801 d union all  
select d.id,d.corpid,d.amt from detail_201802 d union all  
select d.id,d.corpid,d.amt from detail_201803 d union all  
...  
select d.id,d.corpid,d.amt from detail_201812 d ) dd  
left join corp c on dd.corpid=c.corpid where dd.custid='10100'
```

因为数据分表，所以SQL要合并多个月份的查询结果，然后再和分支机构表关联。用视图可以简化书写，本质上还是执行这样的SQL。

对比方案：关系数据库索引存在的问题



关系数据库是无序存储，同一账号在各处都可能存在。索引查找不得不在硬盘读取很多无关数据。每个查询都慢些，大并发总体性能就会很差。

综合练习一 高并发有关联查询

2. 解题思路

内存还是外存？



内存

随机存取快
容量有限



硬盘

读写较慢
容量大

活期明细数据随着时间增长很快，一年就有几十亿条记录，全部放到内存中成本太高，要放到硬盘存储。分支机构表可以放到内存存储。

列存还是行存?

活期明细组表-列存

custid	corpid	amt
1	7	2345.8
2	110	100
3	245	1090
...

活期明细组表-行存

custid	corpid	amt
1	7	2345.8
2	110	100
3	245	1090
...

列存数据分块压缩，适合遍历运算。但对于索引随机查询，有额外解压计算。且每次取数都针对整个分块，复杂度较高，性能不如行存。

有序行存

活期明细组表-行存

客户号10100



custid	corpid	amt
...
10099	110	100
10100	245	1090
10100	110	900
10100	389	50
10101	290	700
...

行存组表中数据，已预先按客户号有序存储。查询时在磁盘上连续读取，可以显著减少磁盘 IO，有效提速。

分级索引缓存



索引文件较大，采用三级索引机制，可根据内存的情况选择加载合适的级别。
缓存在内存中的索引可以有效提高查询速度。

综合练习一 高并发有关联查询

3. 动手练习

准备数据

练习目标

写SPL代码，生成测试数据，包括：

存款明细组表detail.ctx（行存），字段包括：明细号did（1到1000万）、客户号custid（1到10万之间的随机整数）、日期odate（1980年之后的随机日期）、机构号corpid（1到500的随机整数）、金额amt（1000以内的随机数）。

机构简表corp.btx，机构号corpid（1到500）、机构名（corpid转字符串后加上“corp”前缀）。

动手做

编写SPL脚本data.dfx，按照练习目标生成detail.ctx和corp.btx。

可以自行编写data.dfx，也可以参考data1.dfx。要求：detail.ctx只生成了两个字段，要把需要的字段补齐。detail.ctx是否行存？如果不是，请改为行存。detail.ctx是否按照custid有序？如果不是，请再续写代码，按照custid排序。

客户号查询

练习目标

1、无索引查询：以custid字段等于输入参数incustid为过滤条件，查询detail.ctx。结果集和corp.btx关联，返回符合条件的活期明细数据，其中的custid替换成custname。

要判断corp是否已经读入内存，如果没有，就以全程变量的方式读入内存。

2、建立索引：为detail.ctx增加custid字段的索引。

3、有索引查询：在1的基础上，改为按照custid的索引查询。

要判断组表对象detail是否已经存入内存，如果没有，就以全程变量的方式加载3级索引，并存入内存。

	执行时间（毫秒）
无索引	
有索引	

动手做

1、自行编写query.dfx，也可以参考query1.dfx，输入参数是incustid，实现目标1的要求。测试无索引情况下查询的速度，并记录下来。

2、改写data.dfx，或者另写SPL代码，为detail.ctx增加custid的索引。

3、改写query.dfx，利用custid的索引查询，记录执行时间。

提示：利用index()、index@3()、icursor()。

并发查询客户号

练习目标

将查询代码部署在集算器服务器中。另外编写SPL代码，模拟前端应用，随机生成100个和1000个custid（大于1小于100000）作为参数，连接集算器服务器，发起100个和1000个访问，记录完成时间。

	执行时间（毫秒）
100并发	
1000并发	

动手做

将修改好的query.dfx，部署在集算器服务器中。
自行编写callquery.dfx，也可以参考callquery1.dfx，发起100个和1000个访问，记录时间。
提示：利用callx()。

回答问题

还有什么办法可以提高速度？
初始化全局变量、判断全局变量是否存在也需要消耗时间，如何避免？

动手做

继续改写query.dfx。
要求：将判断全程变量的代码去掉，移到主目录的init.dfx中，在服务器启动时执行。
再用callquery.dfx，调用query.dfx。

高并发有序行存

回答问题

1000个并发是同事发生的吗？
如果不是同时发生，真正同时并发的查询有多少？

	执行时间（毫秒）
明细号有序	
客户号有序	

回答问题

有序行存是否能提高查询速度？应该按照什么字段排序？

动手做

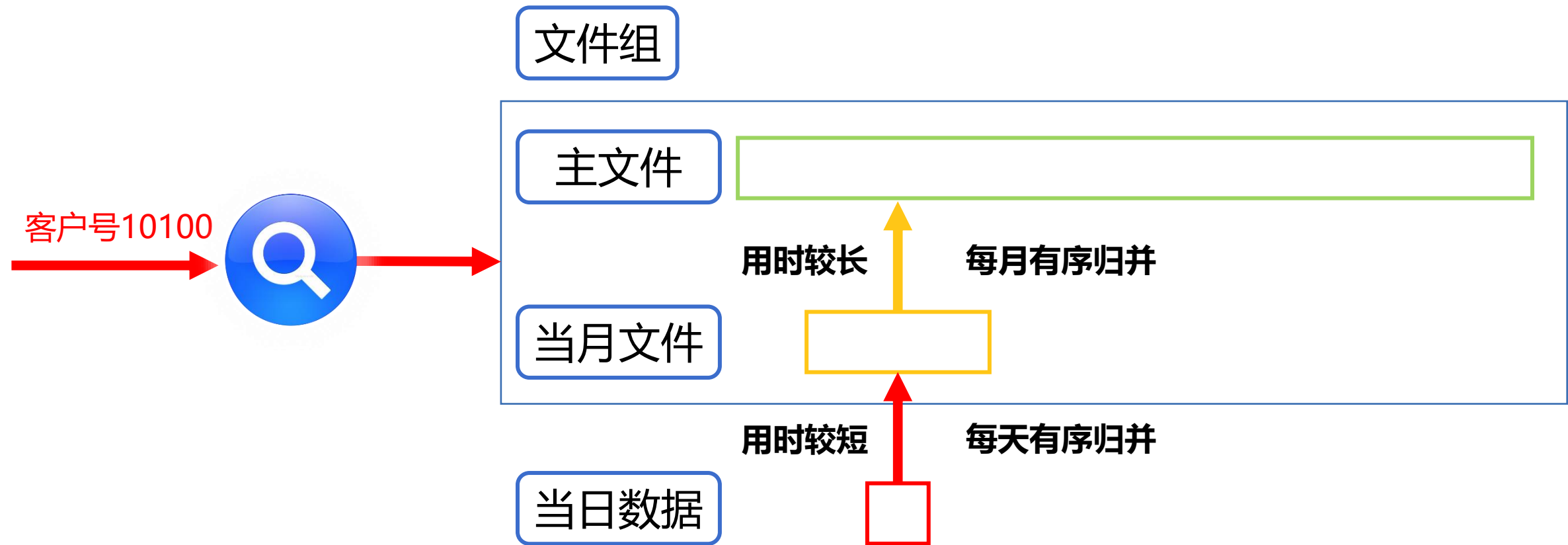
改写data.dfx。
要求：利用已经生成的detail.ctx，按照客户号排序，另存为detailSort.ctx并生成索引。
改写init.dfx，把加载detail.ctx改为加载detailSort.ctx。
再用callquery.dfx测试并发速度，并比较性能。

提示：因为集算器节点服务器仅初始化一次，所以改写init.dfx后要重启节点机。

综合练习一 高并发有关联查询

4. 延伸思考

每日新增活期明细数据的办法



主文件按账号有序，并非按日期有序，不能在末尾追加当日新增数据。主文件几十亿条，每天有序归并新数据时间太长。每月归并一次时间较理想。

每日新增活期明细数据的代码

增量更新

	A	B	C
1	if day==1	=file("detail.ctx").reset()	/每月重整
2	=file("detail.ctx").open().append@a(dataToday)		/每日归并

A1、A2：如果是每月1日，将当月文件有序归并到主文件，同时清空补文件。
A2：将当日数据有序归并到当自动生成的补文件中。

每月重整和每日归并时按照账号有序。在重整和归并的同时，集算器会自动更新索引文件。

每日新增活期明细数据的代码

文件组查询

	A	B
1	<code>=file("detail.ctx")</code>	/文件组
2	<code>=A1.open().icursor(;custid=="10100").fetch()</code>	/查询

通过索引查询时，访问的是主文件和补文件组成的文件组，速度比一个文件稍慢一点。

综合练习二 大明细表自关联

- 1 需求描述
- 2 解题思路
- 3 动手练习
- 4 延伸思考

准备工作-硬件和主目录

硬件 内存：8G以上 硬盘：空余30G以上 CPU：主频1G以上

主目录 在硬盘适当位置解压缩 “practice2.zip”文件夹，并将集算器
主目录设置成"practice2"文件夹
设置方法：集算器-菜单-工具-选项-主目录

综合练习二 大明细表自关联

1. 需求描述



从大量贷款明细数据中去掉冲正交易数据，再用金额分段汇总金额和贷款笔数。分段如：“1000元以下”，“1000元-3000元”，... “30万元以上”。

练习场景：贷款明细排除冲正数据

贷款明细	自关联	分段	汇总
共两千万 - 冲正数据 二十万	排除 冲正数据 和对应明细	30万以上 20万到30万 10万到20万 ...	总金额 去重 贷款笔数

冲正数据是指金额为负值的明细数据。在业务上，其意义在于将与之对应（贷款编号相同）的贷款交易取消掉。

练习场景：计算目标概述

贷款明细表loan

总数据量两千零二十万条

贷款号

贷款日期

金额

分支机构

员工号

种类编号

credit_id	openday	amt	dept_id	employee_id	type_id	...
1	20190101	1000	3	89	3	
1	20190101	-1000	3	89	3	
2	20190101	3999	5	990	2	
...	

贷款明细中金额小于0的是冲正记录，有20万条。汇总计算之前，冲正数据和对应明细（贷款号相等）都要去掉。结果再按照金额分段汇总。

计算目标：1、去掉冲正数据（以SQL为例说明）

```
Select      case
              when t.amtT = '01' then '1000元以下'
              when t.amtT = '02' then '1000元-3000元'
              ...
              else '其他'
            end as 维度,
            sum(t.amt) as 发放金额,
            count(distinct t.credit_id) as 贷款笔数
from        (select      case
                      when amt >= 0 and amt <= 1000 then '01'
                      when amt > 1000 and amt <= 3000 then '02'
                      ...
                      else '09'
                    end as amtT,
                    amt,X1.credit_id
              from      loan   as X1
              left join  (select distinct credit_id
                        from loan where openday <= '2019-10-11' and amt <0.00) as X2
                        on X2.credit_id=X1.credit_id
              WHERE      openday <= '2019-10-11' and X2.credit_id is null
              ) t GROUP BY t.amtT
```

红色框中，X2取冲正记录，X1与X2关联去掉贷款号相同记录，完成去掉冲正数据的计算。

计算目标： 2、按照金额分段汇总（以SQL为例说明）

```
Select
  case
    when t.amtT = '01' then '1000元以下'
    when t.amtT = '02' then '1000元-3000元'
    ...
    else '其他'
  end as 维度,
  sum(t.amt) as 发放金额,
  count(distinct t.credit_id) as 贷款笔数
from
  (select
    case
      when amt >= 0 and amt <= 1000 then '01'
      when amt > 1000 and amt <= 3000 then '02'
      ...
      else '09'
    end as amtT,
    amt,X1.credit_id
  from
    loan as X1
  left join (select distinct credit_id
             from loan where openday <= '2019-10-11' and amt <0.00) as X2 on
    X2.credit_id=X1.credit_id
    WHERE openday <= '2019-10-11' and X2.credit_id is null
  ) t GROUP BY t.amtT
```

分段方式：
"1000元以下",
"1000元-3000元",
"3000元-5000元",
"5000元-1万元",
"1万元-5万元",
"5万元-10万元",
"10万元-20万元",
"20万元-30万元",
"30万元以上"

去掉冲正之后，还有两千万数据，橙色框中为不同分段金额打上分段标示amtT。绿色框根据amtT分段汇总。

综合练习二 大明细表自关联

2. 解题思路

内存还是外存？



内存

随机存取快
容量有限



硬盘

读写较慢
容量大

本例中的贷款数据两千万并不算特别多，但是考虑到未来的增长，所以还是要放硬盘存储。

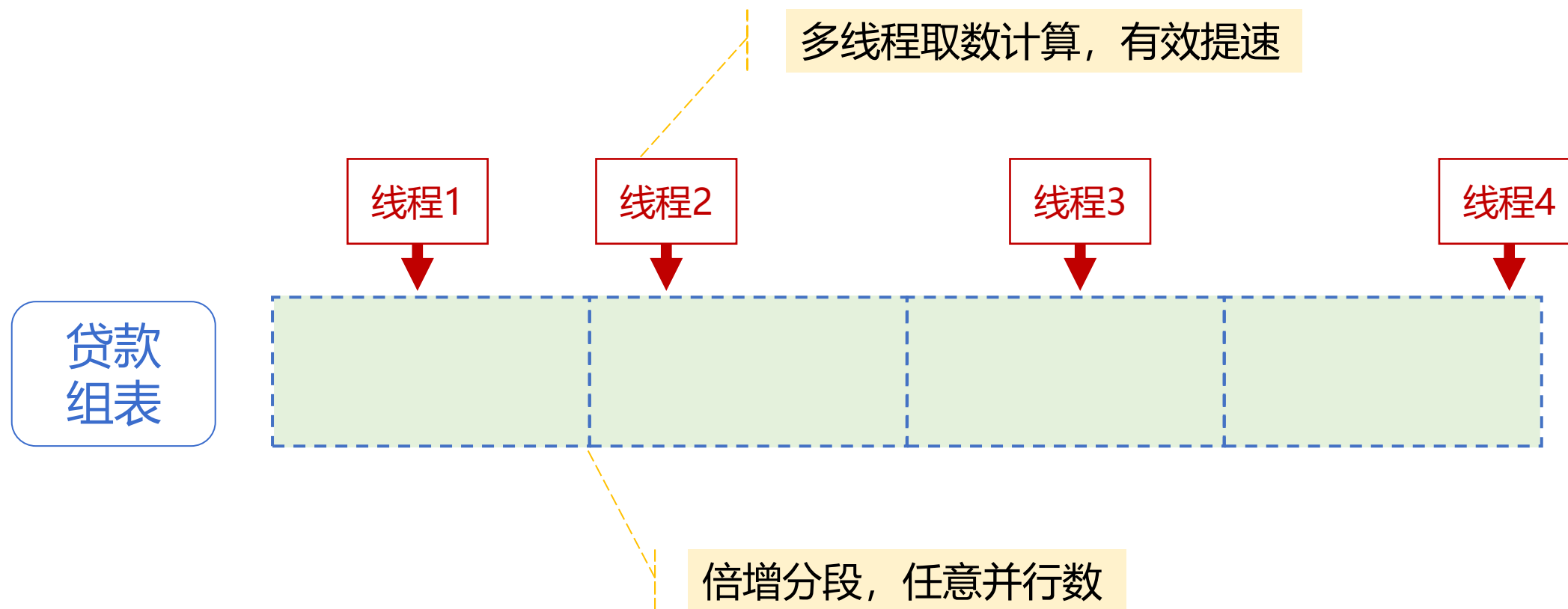
列存还是行存?

贷款组表-有序列存

#credit-id	opendate	amt	...
1	20190101	80000.0	
2	20190101	10000.0	
3	20190101	70900.0	
...	
1299	20190907	5000.0	
1299	20190907	-5000.0	
...	

贷款组表，预先按客户号有序存储。有序列存有利于重复数据的压缩。列数较多的时候，遍历时只取需要的编号、日期、金额三个字段，读取更快。

并行遍历



对贷款组表采用多路游标，多线程同时读取。充分发挥多CPU多核的计算能力，有效提速。

冲正数据单独存放

贷款组表

#credit-id	opendate	amt	...
1	20190101	80000.0	
2	20190101	10000.0	
3	20190101	70900.0	
...	
1299	20190907	5000.0	
1299	20190907	-5000.0	
...	

冲正组表

#credit-id
1299
2096
3115
...

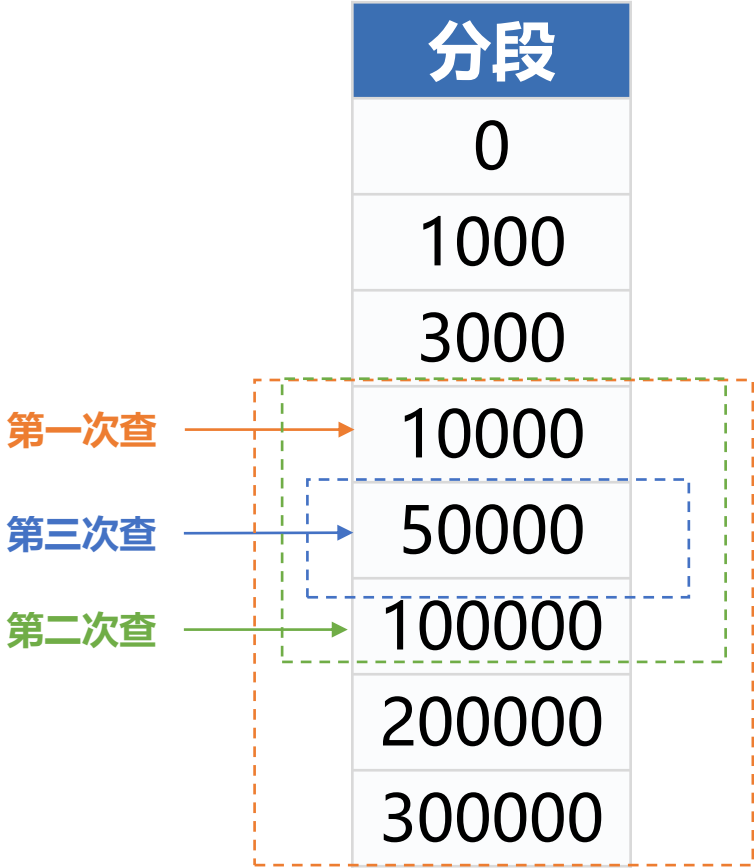
冲正组表单独存放，每次计算的时候就不必在贷款组表中过滤取出，可以有效提高性能。每天更新贷款新数据时，同时计算更新冲正组表数据。

二分法计算分段

用二分法找80000所在分段

贷款组表-去掉冲正数据后

#credit-id	opendate	amt	...
1	20190101	80000.0	
2	20190101	10000.0	
3	20190101	70900.0	
...	
1299	20190907	5000.0	
1299	20190907	-5000.0	
...	

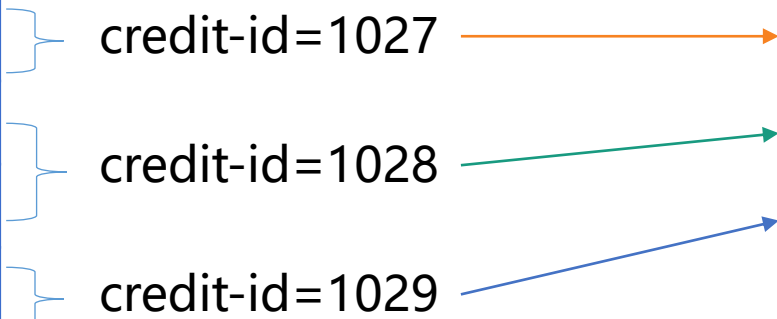


普通方法计算分段，最少计算2个表达式，最多计算16个。本例中80000要计算10个表达式，而两分法查找只需要计算三次数值比较。

有序列存快速去重

贷款组表-去掉冲正

#credit-id
...
1027
1028
1028
1030
...



去重贷款笔数

笔数
...
+1
+1
+1
...

去掉冲正的结果，对贷款号依然有序。对贷款号去重计数的时候，只需要和相邻的贷款号比较即可。多路并行游标也支持快速去重。

综合练习二 大明细表自关联

3. 动手练习

准备数据

练习目标

写SPL代码，生成测试数据：[贷款明细文本文件 \(loan.txt\)](#)。

字段：明细号credit_id (1-2000万)，日期openday (20191021之后100天内的随机日期)，金额amt (1万以内随机数)，部门编号dept_id (1000以内随机整数)，员工号employee_id (1000以内随机整数)，类型type_id (100以内随机整数)，客户号customer_id (1000以内随机整数)

数据量：二千万

排序：无排序

冲正数据：再生成二十万条冲正数据，追加到loan.txt结尾处。字段要求：明细号credit_id (2000万以内的随机整数)、金额amt (1万以内随机数*-1)。其他字段要求同上。

动手做

自行编写SPL，或者参考initdata.dfx，按照练习目标，生成loan.txt。

批处理：数据转换

练习目标

编写SPL代码，完成数据转换。要求将loan.txt，按照明细号排序并转换成列存组表文件data/ctx/loan.ctx。
再将其中金额amt小于0的冲正数据过滤出来，保存为correct.ctx。

动手做

自行编写，或者参考etl.dfx，将loan.txt，按照明细号排序并转换成列存组表文件data/ctx/loan.ctx。

动手做

续写etl.dfx，过滤出贷款明细数据中，amt小于0的记录，存成correct.ctx。

提示：利用sortx()、cursor(;where)、create()。

排除冲正记录之后的贷款总额

练习目标

编写SPL代码，在loan.ctx中，排除掉correct.ctx中的冲正记录。

再对结果分段汇总，统计金额和笔数（按照明细号去重计数）。

分段汇总按照下面的要求分成9段。

分段名称：

["1000元以下","1000元-3000元","3000元-5000元","5000元-1万元","1万元-5万元","5万元-10万元","10万元-20万元","20万元-30万元","30万元以上"]

分段值：

=[0,1000,3000,5000,10000,50000,100000,200000,300000]

动手做

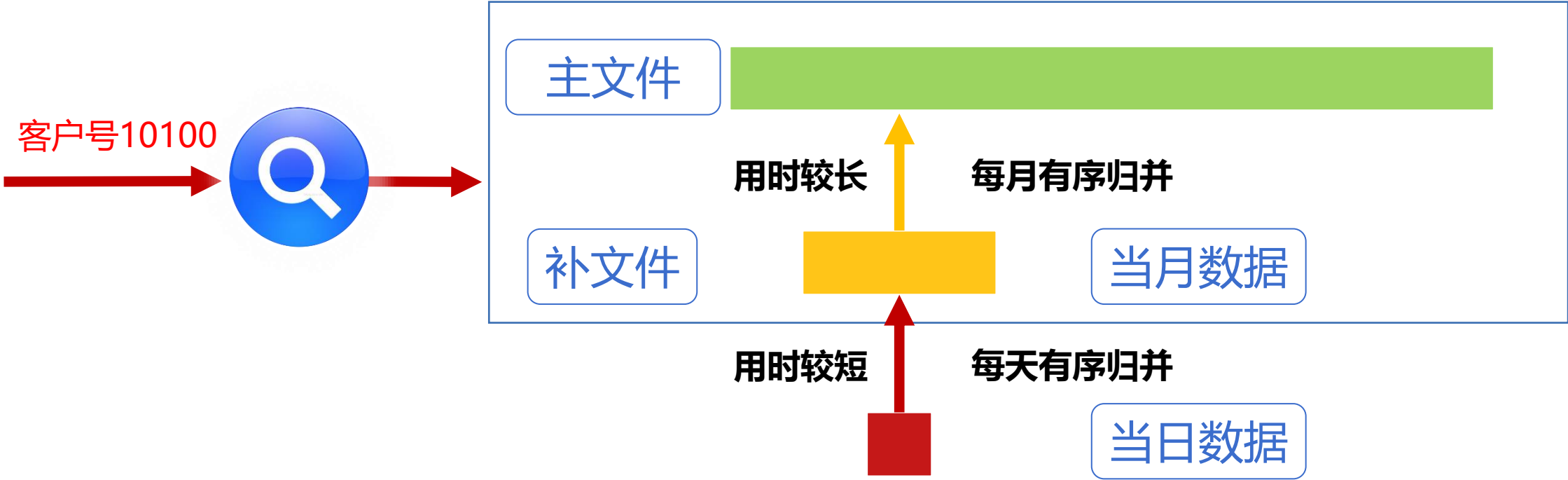
按照练习目标编写query.dfx，在loan.ctx中排除冲正记录，对贷款金额分段统计。截止2019-10-21。

提示： 利用switch@d()、pseg@r()、derive()。

综合练习二 大明细表自关联

4. 延伸思考

每日新增活期明细数据的办法



主文件按账号有序，并非按日期有序，不能在末尾追加当日新增数据。主文件几十亿条，每天有序归并新数据时间太长。每月归并一次时间较理想。

每日新增活期明细数据的代码

增量更新

	A	B	C
1	if day==1	=file("detail.ctx").reset()	/每月重整
2	=file("detail.ctx").open().append@a(dataToday)		/每日归并

A1、A2：如果是每月1日，将补文件有序归并到主文件，同时清空补文件。

A2：将当日数据有序归并到补文件中。

每月重整和每日归并时按照账号有序。在重整和归并的同时，集算器会自动更新索引文件。

每日新增活期明细数据的代码

包含补文件的查询

	A	B
1	=file("detail.ctx")	/包含补文件
2	=A1.open().icursor(;custid=="10100").fetch()	/查询

通过索引查询时，访问的是主文件和补文件组成归并结果，速度比一个文件稍慢一点。

综合练习三 实时指标计算

- 1 需求描述
- 2 解题思路
- 3 动手练习
- 4 延伸思考

准备工作-硬件和主目录

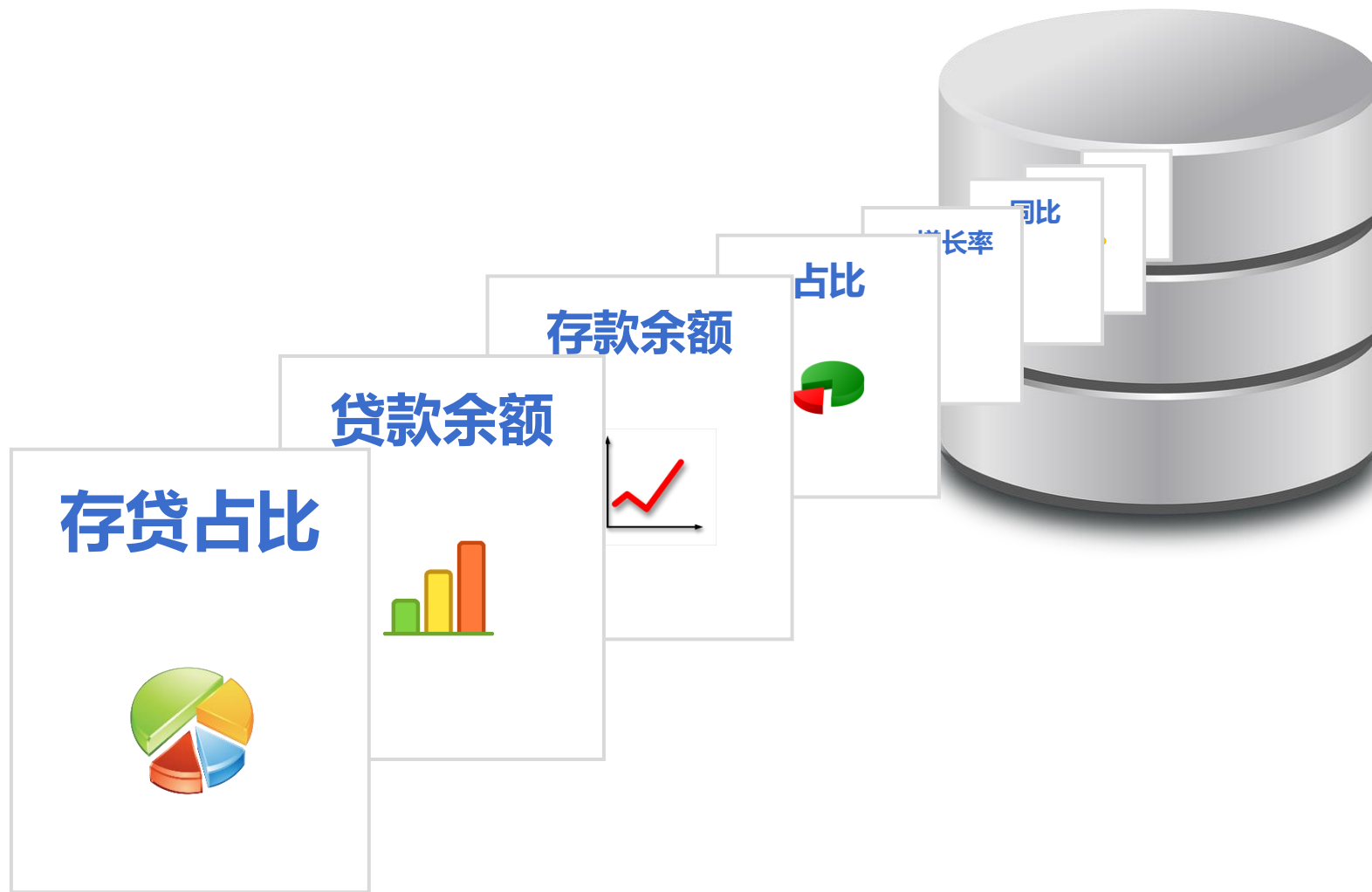
硬件 内存：8G以上 硬盘：空余30G以上 CPU：主频1G以上

主目录 在硬盘适当位置解压缩 “practice3.zip”文件夹，并将集算器
主目录设置成"practice3"文件夹
设置方法：集算器-菜单-工具-选项-主目录

综合练习三 事实指标计算

1. 需求描述

需求概述



统计指标从大量明细数据汇总。传统技术实时计算无法达到性能要求，只能预先算好结果供查询使用。

练习场景：日统计指标计算

过滤条件

日期
币种
机构

...

维度

年龄
客户类型
学历

...

统计

日汇总
同比环比
去年年末

...

并发

用户几千人
集中时间
查询指标

中小银行指标计算体系要实现过滤、维度分组、各种统计计算。查询指标的用户也有几千个，需要较大的并发能力。

练习场景：参与计算的维度标签较多

五级分类

一级贷款
二级贷款
...

四类担保

担保A类
担保B类
...

十种客户

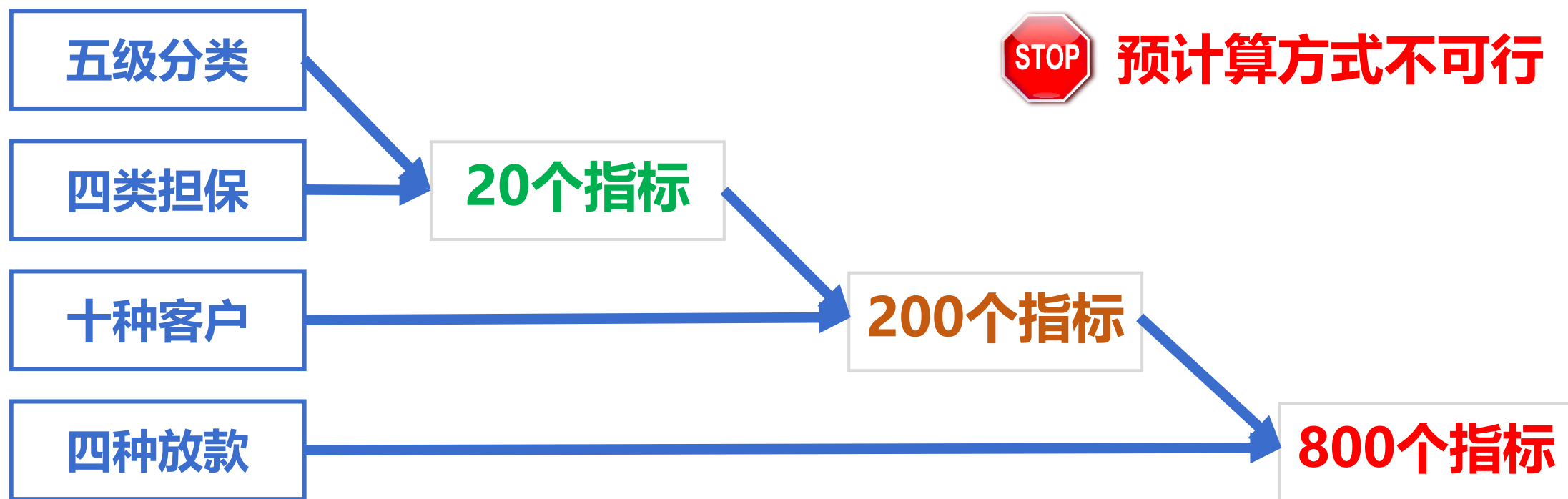
甲种客户
乙种客户
...

四种放款

壹种放款
贰种放款
...

这里只列举了贷款四个维度，实际上还有很多个维度的很多种标签。

练习场景：众多标签排列组合，指标泛滥



贷款四个维度就形成800个指标，还有存款和其他维度。加上日期维度，组合出来的指标太多，预计算方式的结果太多，系统无法承受！

练习场景：日统计指标对应明细数据量很大



实时计算可行吗

存款

每日千万条
十几个字段
保留一年

贷款

每日百万条
几百个字段
保留一年

客户

几百万个
每日存全量
保留一年

其他

员工千人
机构千个
...

银行指标计算体系涉及的明细数据量非常巨大，要保证指标查询的秒级响应速度是个挑战。

对比方案：数据库存款指标

存款主表deposit

dt	curr	cust_no	id	dept	...

日期 币种 客户号 业务号 机构号

客户表cust

dt	cust_no	cust_sub	...

日期 客户号 客户性质

存款主表和客户表通过字段日期dt、客户号cust_no关联。存款主表是窄表，十几个字段，每天两千万条记录。每个表存一个月，六亿条数据。

对比方案：某日人民币存款的非农业去重客户数

```
select count(distinct c.cust_no)
  from deposit d inner join cust c
    on d.cust_no=c.cust_no and d.dt=c.dt
 where d.dt=to_date('20191001','yyyymmdd')
    and d.curr=1 and c.cust_sub=1
    and d.dept in (2,3,6,8...)
```

计算“某日人民币存款的非农业去重客户数”。几亿数据量的存款主表和几千万的客户表关联，很难达到秒级响应速度。

对比方案：某日人民币存款的农业户满足条件的总金额

```
select sum(amt)
  from deposit d
       inner join cust c
         on d.cust_no=c.cust_no and d.dt=c.dt
 where d.dt=to_date('20191001','yyyymmdd' )
       and d.curr=1 and c.cust_sub=0 and d.id<>0
       and code11=1 and code12=2
       and d.dept in (2,3,6,8...)
```

计算“某日人民币贷款的农业客户满足code11和code12条件的总金额”。几十字段、千万数据贷款宽表和客户表关联，也难达到秒级响应。

综合练习三 事实指标计算

2. 解题思路

内存还是外存？



内存

随机存取快
容量有限



硬盘

读写较慢
容量大

存款数据随着时间增长很快，一年就有几十亿条记录，要放到硬盘存储。客户表每天保存全量，也要硬盘存储。员工、机构等可以内存。

计算特征分析：大表遍历

子表：贷款

日期
币种
客户号
业务号
...

主表：客户

日期
客户号
性质
...

子表：存款

日期
币种
客户号
业务号
...

N:1

1:N

在一天内，指定币种，客户和存款、贷款都是一对多的主子表关系。日统计指标计算本质上是大表关联后，遍历汇总：例如客户数、总金额等。

有序列存

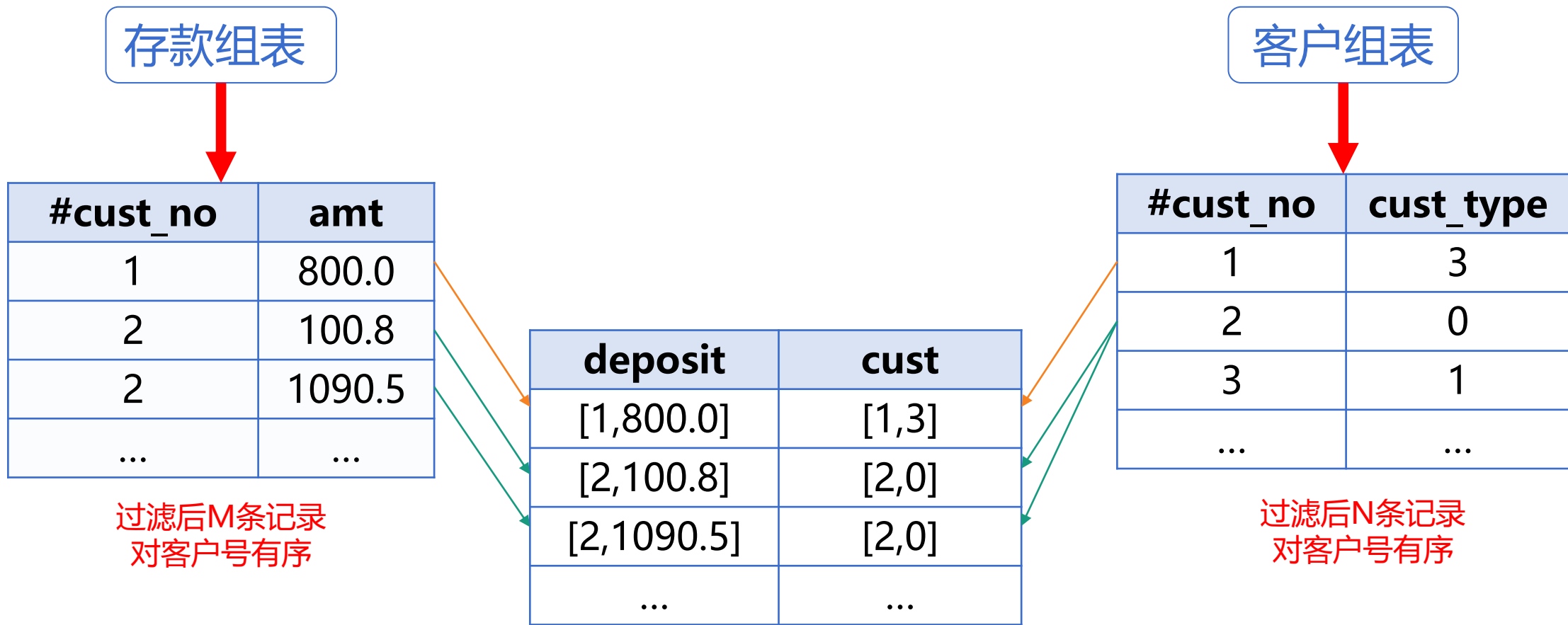
存款组表-有序存储

#cust_no	#curr	amt	...
1	1	800.0	
2	1	100.8	
2	1	1090.5	
...	
1	2	50.0	
6	2	700.0	
...	

币种、客户号等采用整形存储，减少存储量，提高计算的速度。
当整数小于65536时，不再对象化，而是引用内存中预先生成的小整数对象，可减少对象化的时间。

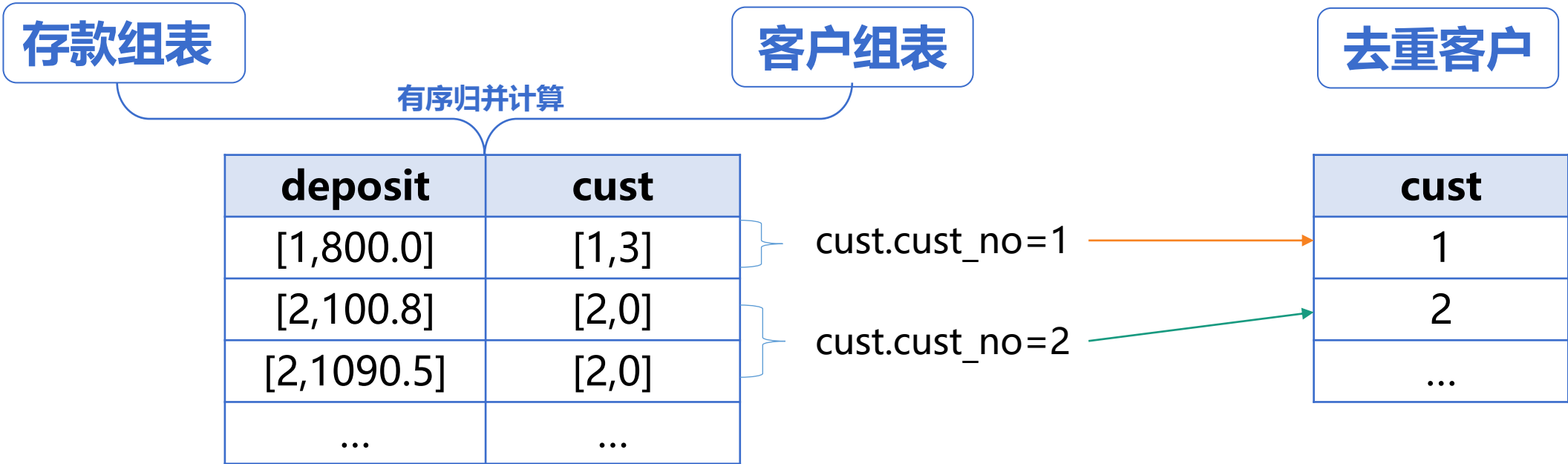
存款组表，预先按客户、币种有序存储。有序列存有利于重复数据的压缩。按照客户号、币种过滤的时候，可以快速跳过不需要的记录。

有序归并



主子表都按照客户号有序存储，可以用归并算法一次遍历实现JOIN，复杂度M+N，比外存分段HASH JOIN的复杂度SUM(Ni*Mi)低很多！

有序去重



主子表有序归并的结果，对客户号依然有序。对客户去重计数的时候，只需要和相邻的客户号比较即可。

布尔维序列

按照部门号过滤

部门号在指定范围内(2,3,6,8...)

序号	cust_no	curr	dept	...
1	1	1	3	
2	2	2	1	
3	2	1	2	
...	

存款数据文件

序号	code	value
1	1	营业所1
2	2	营业所2
3	3	营业所3
...

部门表

IN计算的性能问题

IN计算要在存款数据文件中找n个部门编号。如果存款数据文件中的部门编号无序，那么查找需要遍历整个文件，性能较差。如果部门有序，用二分法会提速，但是要查找多个值，也还是很慢。

而且，计算的耗时和n有关系，n较大时性能会很差。

在部门表，包含多个需要查找的部门号。部门过滤问题最终是要计算部门号 in (2,3,6,8...)。

布尔维序列

按照部门号过滤

部门号在指定范围内(2,3,6,8...)

序号	cust_no	curr	dept	...
1	1	1	3	
2	2	2	1	
3	2	1	2	
...	

存款数据文件

部门表

序号	code	value
1	1	营业所1
2	2	营业所2
3	3	营业所3
...

序号	成员
1	False
2	True
3	True
...	...

布尔维序列

布尔维序列用于IN计算

查询时：根据IN条件和部门表生成布尔维序列。

分段遍历存款文件。第1条记录，用dept字段的值3，找到布尔维序列中第3个成员，值为true，因此第1条客户记录满足条件。

如果值为false，则不满足条件。其他记录以此类推。

布尔维序列将值比较转换为序号引用，有效的减少了计算时间。IN计算时间和IN枚举值的多少无关，不会随着枚举值的增加而增加。

综合练习三 事实指标计算

3. 动手练习

准备数据

练习目标

编写SPL代码，生成2020年10月21日的客户文本文件和存款文本文件。

分支机构文件CORP.txt

字段：机构号id，父机构号pid，递归结构，已经预先手工编好，可以直接使用。

数据量：258；排序：id

客户文件CUST_BASEINFO20201021.txt

字段：客户号CUST_NO（1-2000万），客户性质CUST_SUB（随机取值1或者2。1：农业；2：非农业）。数据量：2000万；排序：无

存款文件DEPOSIT_CURR20201021.txt

字段：LN_CURR_COD币种（随机取值1或者0。1：人民币；0：其他），CUST_NO客户号（2000万以内的随机数），LN_BUSN_NO存款流水号（1到3000万），DEPTCODE机构号（先读入一个集合CORP.txt中取大于等于100的id，然后在集合中随机取一个作为机构号），AMT金额（1万以内的随机数）。数据量：3000万；排序：无

动手做

自行编写SPL或者参考initdata.dfx，生成客户文本文件和存款文本文件。

批处理：数据转换

练习目标

编写SPL代码，

将客户文本文件排序，转换成列存组表文件CUST_BASEINFO20201021.ctx。

将存款明细文本文件按照客户号排序，转换成列存组表文件DEPOSIT_CURR20201021.ctx。

排序字段为：排序字段为客户号CUST_NO,币种LN_CURR_CO,存款流水号LN_BUSH_NO

动手做

自行编写，或者参考etl.dfx，按照练习目标要求，将客户文本文件排序并转换成组表文件。

动手做

续写etl.dfx，按照练习目标要求，将存款明细数据按照客户号排序，转换成组表文件。

提示：利用sortx()、create()。

去重客户数和存款总金额

练习目标

编写SPL代码，计算去重客户数和存款总金额

输入参数：

日期：IN_BATCH_DATE=20201021

选项：

IN_OPTION= DISTINCT CUST去重客户数，此时要计算客户号CUST_NO去重计数值

IN_OPTION= TOTAL总金额，此时要计算金额AMT汇总值

过滤条件-分支机构和币种：

机构IN_ORG_CD=6需要找到所有的子机构，再查找这些子机构对应的存款记录

币种IN_CURR_CD=1

过滤条件-其他：例如IN_WHERE="CUST_SUB==0"，允许前端任意组合查询条件

动手做

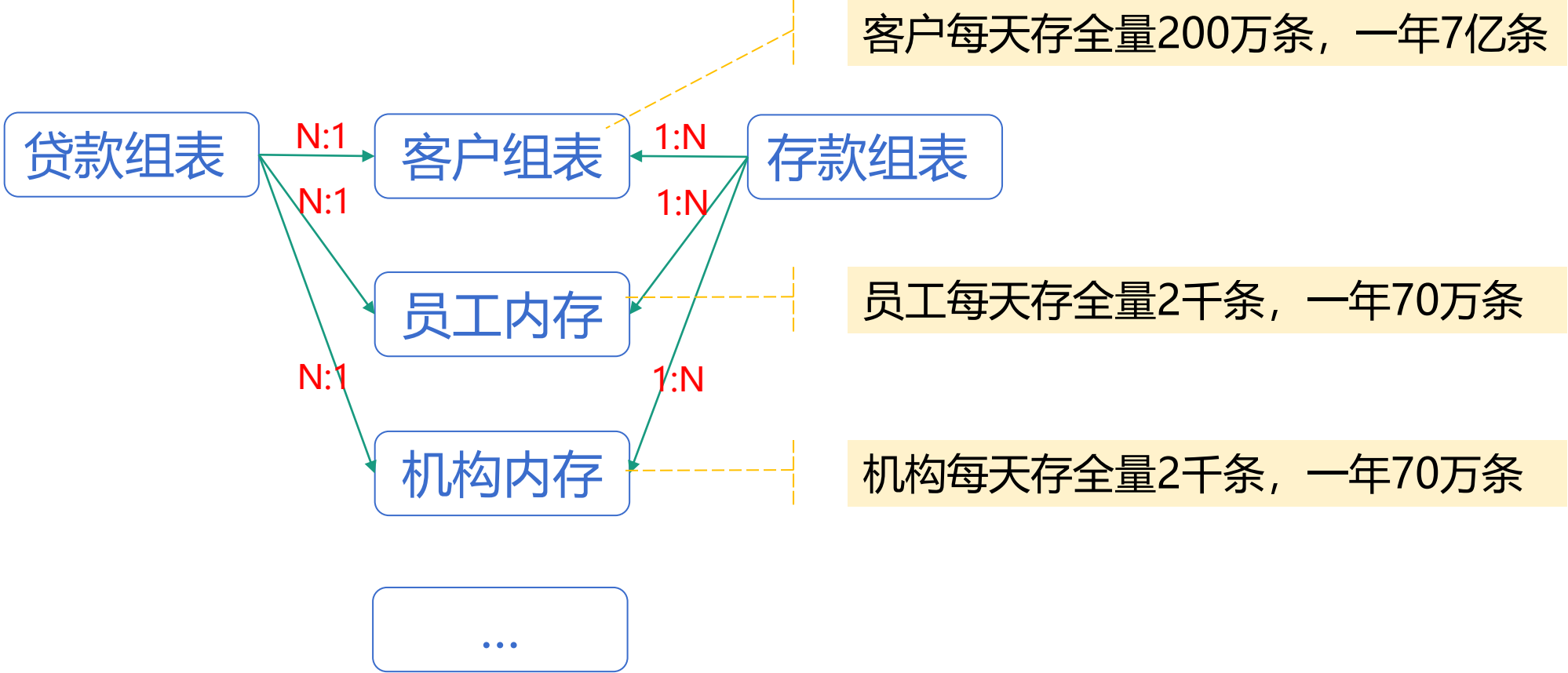
按照练习目标的要求，编写query.dfx。

提示：利用布尔维序列计算机构过滤、group@1()、total()、\${}。

综合练习三 事实指标计算

4. 延伸思考

客户之外的其他维度



员工、机构等统计方式和贷款、存款也是主子表关系。但全年只有几十万条记录，可以全内存存储，每天更新增加当日新数据。

内存初始化加载员工序表

	A	B
1	if !ifv(employee)	=file("employee.btx").import@b(dt,employeeid,type)
2		=env(employee,B1)

A1：判断全局变量中是否存在employee，如果存在，表示已经加载。

B1：如果全局变量中没有，那么打开集文件读入。

B2： employee存入全局变量。

员工只需在每天新增数据时加载一次，结果放到全局变量中缓存。每次查询的时候无需重复加载。

某日一类员工人民币存款总金额

	A	B
1	>in_dt=20191001	>in_curr=1
2	=employee.select(dt ==in_dt && employeetype==1)	
3	=file("data/ctx/deposit.ctx").create()	
4	=A3.cursor(employeeid,amt;dt==in_dt&&id!=0&&curr==in_curr)	
5	=A4.switch@i(employeeid,A2:employeeid)	
6	return A5.groups(;sum(amt):amtall)	

A5：主子表的有序关联joinx，改为用游标switch员工内存序表。将员工和存款看做外键关系。@i选项表示如果员工表中找不到记录的话，就去掉这个存款记录。

注意这个switch在这里也没有立即执行。

A6：对游标执行之前定义好的过滤、外键关联和汇总计算，一次遍历即可完成。

其他和之前的代码类似。

每次实时计算指标的时候可以直接用全局变量employee。这里用cursor.switch@i(...)的方式，比两个组表文件关联速度更快。

附录 SPL和组表

什么是SPL

SPL (Structured Process Language) 是一种面向结构化数据计算的程序语言，采用了类似其它高级程序设计语言的语句、函数风格语法，其代码更象是Java、Basic等语言

本章介绍SPL特有的数据类型、语法特征以及高性能数据存储方案

多层次参数

与普通程序语言不同，SPL的函数参数是分层次的
使用逗号、冒号和分号作为函数参数的分隔符，层次更明晰

比如多表关联：

	A
1	=lineitem.switch(L_PARTKEY,part:P_PARTKEY;L_SUPPKEY,supp:S_SUPPKEY)

Index	L_O...	L_LI...	L_PARTKEY	L_SUPPKEY	L_Q...
1	1	1	155190	7706	
2	1	2	67310	7311	
3	1	3	63700	3701	
4	1	4	2132	4633	
5	1	5	24027	1534	
6	1	6	15635	638	
7	2	1	106170	1191	
8	3	1	4297	1798	
9	3	2	10026	6540	

lineitem

Index	P_PARTKEY	P_NAME	P_MFGR	P_BRAND
1	1	goldenro...	Manufact...	Brand#13
2	2	blush this...	Manufact...	Brand#13
3	3	spring gr...	Manufact...	Brand#42
4	4	cornflowe...	Manufact...	Brand#34

part

Index	S_SUPPKEY	S_NAME	S_ADDRESS	S_NATION...
1	1	Supplier#...	N kD4on...	17
2	2	Supplier#...	89eJ5ksX...	5
3	3	Supplier#...	q1,G3Pi6...	1
4	4	Supplier#...	Bk7ah4C...	15

supp

分号将参数分为了两组，分别是：L_PARTKEY,part:P_PARTKEY与L_SUPPKEY,supp:S_SUPPKEY



Index	L_O...	L_LI...	L_PARTKEY	L_SUPPKEY	L_Q...
1	1	1	155190	7706	
2	1	2	67310	7311	
3	1	3	63700	3701	
4	1	4	2132	4633	
5	1	5	24027	1534	
6	1	6	15635	638	
7	2	1	106170	1191	
8	3	1	4297	1798	
9	3	2	10026	6540	

P_PARTKEY	P_NAME	P_MFGR	P_BRAND	P_TYPE
155190	chocolate ...	Manufact...	Brand#44	PROMO B...

S_SUPPKEY	S_NAME	S_ADDRESS	S_NATION...	S_PHONE
7706	Supplier#...	BIHq75Vo...	23	33-481-...

函数选项

SPL中函数可以使用选项，使同一个函数有不同的工作方式
函数选项的基本格式是f@o(...), o就是函数f的选项

比如，分别按日、月、年来统计两个日期的间隔时间：

	A
1	=interval("2012-12-12","2020-02-02")
2	=interval@m("2012-12-12","2020-02-02")
3	=interval@y("2012-12-12","2020-02-02")

A1:

Value
2608

A2:

Value
86

A3:

Value
8

A1用interval函数计算两个日期的间隔天数，A2和A3中的@m和@y就是interval函数的选项，有了选项后，函数计算时间间隔时，将分别以月和年为单位。使用函数选项，可以使得一个函数满足多种需要，在扩充了函数功能的同时，也避免了过多的函数名或者函数参数。

网格与变量

SPL脚本写在一个网格中，称为网格文件（后缀为dfx）

在网格中可以使用单元格作为变量

如果一个单元格有值，那么在计算格或者执行格中，就可以直接使用单元格名称引用单元格值，
比如：

	A	B
1	3	/常数格，值为3
2	=3*5	/计算格，结果为15
3	=A1*A2	/单元格变量作为参数，结果为45

有值的单元格可以是常数格、计算格，也可以是被执行格赋值的其它单元格
单元格名称由列号字母和行号整数组成

序列

序列是SPL的一种关键数据类型，是有序的泛型集合

集合性： 序列具有集合的一般特性，可以进行集合运算

	A
1	<code>= [5,6,7]^ [6,7,8]</code>

Index	Member
1	6
2	7

泛型性： 成员的数据类型可以不同，成员本身也可以是序列

	A
1	<code>= [1,"abc", [1,"abc"]]</code>

Index	Member
1	1
2	abc
3	[1,abc]

有序性： 序列具有有序性，成员一样但顺序不同的两个集合不相等

	A
1	<code>= [1,2]== [2,1]</code>

Value
false

序表

序表是SPL的一种关键数据类型，是有结构的序列

序表类似我们常说的数据表，不同之处在于序表的成员强调有次序

可以用T(k)访问其成员, 其中k为记录在表中的行位置

比如：访问订单表的第5条记录

	A
1	=file("orders.txt").import@t()
2	=A1(5)

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	
1	1	36901	Q	173665.47	1996-01...	
2	2	78002	Q	46929.18	1996-12...	
3	3	123314	F	193846.25	1993-10...	
4	4	136777	Q	32151.78	1995-10...	
5	5	44485	F	144659.2	1994-07...	
6	6	55624	F	58749.59	1992-02...	
7	7	39136	Q	252004.18	1996-01...	
8	32	130057	Q	208660.75	1995-07...	
9	33	66958	F	163243.98	1993-10...	

所有订单记录



O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_...
5	44485	F	144659.2	19

排列

排列是对序表记录的引用，其成员属于其它序表，不再复制记录

例：从订单表中查找订购日期年份为1992年的订单记录，再修改原序表的值，观察数据变化

	A
1	=file("orders.txt").import@t()
2	=A1.select(year(O_ORDERDATE)==1992)
3	>A1.modify(6, 100:O_TOTALPRICE)

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	
1	1	36901	<u>O</u>	173665.47	1996-01...	
2	2	78002	<u>O</u>	46929.18	1996-12...	
3	3	123314	<u>F</u>	193846.25	1993-10...	
4	4	136777	<u>O</u>	32151.78	1995-10...	
5	5	44485	<u>F</u>	144659.2	1994-07...	
6	6	55624	<u>F</u>	58749.59	1992-02...	

执行A1后，原序表数据

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_	
1	6	55624	<u>F</u>	58749.59	1992-02...	4-	
2	37	86116	<u>F</u>	206680.66	1992-06...	3-	
3	128	73957	<u>F</u>	66195.16	1992-06...	1-	
4	129	71134	<u>F</u>	261013.14	1992-11...	5-	
5	130	36964	<u>F</u>	189484.12	1992-05...	2-	
6	134	6199	<u>F</u>	200354.3	1992-05...	4-	

执行A2后，排列数据

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	
1	1	36901	<u>O</u>	173665.47	1996-01...	
2	2	78002	<u>O</u>	46929.18	1996-12...	
3	3	123314	<u>F</u>	193846.25	1993-10...	
4	4	136777	<u>O</u>	32151.78	1995-10...	
5	5	44485	<u>F</u>	144659.2	1994-07...	
6	6	55624	<u>F</u>	100	1992-02...	

执行A3后，原序表数据发生变化

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_	
1	6	55624	<u>F</u>	100	1992-02...	4-	
2	37	86116	<u>F</u>	206680.66	1992-06...	3-	
3	128	73957	<u>F</u>	66195.16	1992-06...	1-	
4	129	71134	<u>F</u>	261013.14	1992-11...	5-	
5	130	36964	<u>F</u>	189484.12	1992-05...	2-	
6	134	6199	<u>F</u>	200354.3	1992-05...	4-	

执行A3后，排列中的数据也发生变化

排列不支持modify等修改原序表的函数，所以这种影响是单向和安全，用户可以放心使用排列和序表。

游标

SPL有游标，可以使用类似处理小数据量的代码来处理大数据

例：使用游标过滤订单表文件中客户号为10001的订单记录

	A
1	=file("orders.txt").cursor@t()
2	=A1.select(O_CUSTKEY==10001)
3	=A2.fetch()
4	=A2.fetch(1)



O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE
1	72634	O	173665.47	1996/1/2
2	78002	O	46929.18	1996/12/1
3	123314	F	193846.25	1993/10/14
4	136777	O	32151.78	1995/10/11
5	44485	F	144659.2	1994/7/30
6	55624	F	58749.59	1992/2/21
7	39136	O	252004.18	1996/1/10
...

A3的结果为：

O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORITY
51223	36901	O	193846.25	1993/10/14	5-LOW
142332	36901	F	32151.78	1995/10/11	5-LOW
167855	36901	F	144659.2	1994/7/30	4-NOT SPECIFIED
478745	36901	O	58749.59	1992/2/21	2-HIGH

A4的结果为空

游标类似于一个指针，通过移动指针的位置来获取数据
游标一般具有单向性，也就是只能向后读取或跳过数据，不可回头。

延迟计算与立即计算

立即计算

游标上有两种类型的运算，一种称为立即计算：
使用游标时，一些计算会立即执行
比如：cs.groups这样的聚合函数

例如，订单表（orders.txt）中，统计各年份的订单总数。

	A
1	=file("orders.txt").cursor@t()
2	=A1.groups(year(O_ORDERDATE):Year;count(~):OrderCount)

A2:

Index	🔑 Year	OrderCount
1	1992	227089
2	1993	226645
3	1994	227597
4	1995	228637
5	1996	228626
6	1997	227783
7	1998	133623

游标的立即计算，意思就是游标函数执行后，
该单元格返回的是最终的执行结果数据

延迟计算与立即计算

延迟计算

另一种称为延迟计算：
使用游标时，一些运算在游标上定义后不会马上计算
比如：cs.select/cs.new/cs.derive/cs.run
例如，查找订单表中客户号（O_CUSTKEY）为4的所有记录

	A
1	=file("orders.txt").cursor@t()
2	=A1.select(O_CUSTKEY==4)
3	=A2.fetch()

A3							
O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORITY	O_CLERK	
164711	4	F	311722.87	1992/4/26	3-MEDIUM	Clerk#000000361	
385825	4	O	277493.04	1995/11/1	2-HIGH	Clerk#000000465	
1192231	4	O	143971.54	1996/6/3	1-URGENT	Clerk#000000978	
1226497	4	F	88317.19	1993/10/4	1-URGENT	Clerk#000000154	
...	

A3使用cs.fetch函数，此时才真正计算，返回数据

延迟计算的好处在于写起来和内存运算差不多，容易理解，但又不会真地生成中间结果集（内存计算则会）占用空间（或缓存）

A1
Value
com.raqsoft.dm.cursor.FileCursor@1f48da9f

A1打开文件创建游标，返回游标

A2
Value
com.raqsoft.dm.cursor.FileCursor@1f48da9f

A2使用游标函数，仅在A1的游标之上定义了游标，结果仍旧返回游标

管道

SPL提供了管道用于复用游标的数据，游标一次遍历过程中可以计算出多个结果

比如，计算按优先级分组的金额求和与按年份分组的金额求和

A	
1	=file("orders.txt").cursor@t()
2	=channel(A1).groups(O_ORDERPRIORITY;sum(O_TOTALPRICE):amount)
3	=A1.groups(year(O_ORDERDATE):year;sum(O_TOTALPRICE):amount)
4	=A2.result()

A3		
Index	year	amount
1	1992	3.4330674052430122E10
2	1993	3.4340410079030388E10
3	1994	3.441636905296985E10
4	1995	3.454613318359988E10
5	1996	3.460936476086032E10
6	1997	3.437363341304094E10
7	1998	2.0212721905530304E10

游标中定义了按年份分组的金额求和

游标

游标压入管道

管道

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_ORDER...
1	1	36901	O	173665.47	1996-01...	5-LOW
2	2	78002	O	46929.18	1996-12...	1-URGENT
3	3	123314	F	193846.25	1993-10...	5-LOW
4	4	136777	O	32151.78	1995-10...	5-LOW
5	5	44485	F	144659.2	1994-07...	5-LOW
6	6	55624	F	58749.59	1992-02...	4-NOT SP...

orders.txt

同一个游标可以压入多个不同的管道中。
遍历一次游标可计算出多个结果。

A4		
Index	O_ORDERPRIORITY	amount
1	1-URGENT	4.541872943708031E10
2	2-HIGH	4.547977624302999E10
3	3-MEDIUM	4.51536080884604E10
4	4-NOT SPECIFIED	4.5276033983099785E10
5	5-LOW	4.5501158695791756E10

管道中定义了按优先级分组的金额求和

程序游标

有些算法可能产生落地的中间结果，SPL提供程序游标以避免数据落地，从而提高性能

订单表按订购日期有序，按日期、产品去除重复，再统计记录条数
中间结果落地：

	A
1	=file("orders.txt").cursor@t()
2	=A1.groupx(O_ORDERDATE,O_CUSTKEY)
3	=A2.skip()

tmpdata7761142513456679165
tmpdata5857897864734014615
tmpdata8127755923321165032
tmpdata1268588399243624370
tmpdata6708140956637753188
tmpdata6319508895148103638
tmpdata2361076422568055873

临时文件保
存中间结果

要使去重的中间结果不落地，可以先生成程序游标，distinctCUST.dfx，脚本如下：

	A	B
1	=file("orders.txt").cursor@t()	
2	for A1;date(O_ORDERDATE)	=A2.id(O_CUSTKEY)
3		return B2

主程序可通过cursor函数调用程序游标，用法与普通游标类似：

	A
1	=cursor("distinctCUST.dfx")
2	=A1.skip()

组表

组表是SPL的高性能存储方案，类似数据库中的数据表，物理上以文件的形式存在。

组表的特点有：有序、列存、分段

有序——不论范围查询或等值查询都可以快速查询

列存——按需取列减少读取提升性能

分段——多路游标并行计算提升效率

充分利用好这些特点可以有效提升查询性能、节省存储空间

列、键、维

列——即常规意义的数据表的字段

键——即主键，可由多列构成，主键可以唯一确定一条记录

维——有序的列，可以不唯一，只有维没有主键的表不能更新

例如，订单表的文本文件（orders.txt）转为订单表的组表文件（orders.ctx）

	A	
1	=file("orders.txt").cursor@t()	
2	=file("orders.ctx").create(#O_ORDERKEY,O_CUSTKEY, ...)	/维用#开头，这里的O_ORDERKEY是维也是主键
3	=A2.append(A1)	
4	=A2.close()	

Index	O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_C	
1	1	36901	<u>O</u>	173665.47	199	
2	2	78002	<u>O</u>	46929.18	199	
3	3	123314	<u>F</u>	193846.25	199	
4	4	136777	<u>O</u>	32151.78	199	
5	5	44485	<u>F</u>	144659.2	199	

orders.txt



Index	 O_ORDERKEY	O_CUSTKEY	O_ORDER...	O_TOTAL...	
1	1	36901	<u>O</u>	173665.47	
2	2	78002	<u>O</u>	46929.18	
3	3	123314	<u>F</u>	193846.25	
4	4	136777	<u>O</u>	32151.78	
5	5	44485	<u>F</u>	144659.2	

orders.ctx

列存和行存

组表默认是按列存储数据，使用@r选项可以变为按行存储

例如，订单表的文本文件（orders.txt）转为订单表的行存组表文件（orders.ctx）

	A
1	=file("orders.txt").cursor@t()
2	=file("orders.ctx").create@r(#O_ORDERKEY,O_CUSTKEY,O_ORDERSTATUS,...)
3	=A2.append(A1)
4	=A2.close()

如果需要在组表中执行某条或某几条数据的查询，可以选择按行存储，以便于在查询时获得更好的性能，但行式存储的组表不便于处理整表的遍历计算，也不支持用多路游标访问数据

分段

组表还可按字段分段存储

例如，订单表的组表文件（orders.ctx），按照客户号分段，取100份中的前两份，观察对比

	A
1	=file("orders.ctx").create()
2	=A1.cursor(;;1:100).fetch()
3	=A1.cursor(;;2:100).fetch()
4	=A2.close()

设定字段按组分段，每段之间不会出现重复值，这对于聚合运算来说，可以使得计算变得更加简单，提高计算效率。

A2:

Index	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_C	^
8193	835	Q	166820.28	1997-02...	3-N	
8194	835	Q	168878.59	1996-02...	5-L	
8195	835	Q	84578.75	1998-01...	4-N	
8196	835	Q	110826.86	1995-07...	1-L	
8197	835	Q	62844.12	1997-06...	3-N	
8198	835	F	239455.44	1992-05...	3-N	
8199	835	Q	74293.44	1997-01...	3-N	
8200	835	F	36524.18	1994-10...	1-L	
8201	835	Q	236117.08	1996-08...	2-H	
8202	835	F	49894.56	1992-08...	2-H	

A3:

Index	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_C	^
1	836	F	87792.44	1992-08...	1-L	
2	836	F	249124.09	1994-01...	4-N	
3	836	F	144870.84	1992-03...	2-H	
4	836	Q	55709.59	1997-11...	2-H	
5	836	Q	182531.92	1995-10...	3-N	
6	836	Q	147471.41	1998-03...	4-N	
7	836	F	68998.07	1993-10...	4-N	
8	836	Q	345753.36	1997-12...	3-N	
9	836	Q	214415.11	1995-11...	3-N	
10	836	Q	189048.29	1996-05...	4-N	

附表

允许把主子关系的两个表数据存储在同一个组表中，子表称为附表

例如，订单表与订单明细这样的主子表，其中订单表为主表，订单明细为子表（附表）

	A
1	=file("orders.ctx").create().cursor()
2	=file("lineitem.ctx").create().cursor(L_ORDERKEY:O_ORDERKEY,L_LINENUMBER,L_PARTKEY,L_SUPP KEY)
3	=file("orders_lineitem.ctx").create(#O_ORDERKEY,O_CUSTKEY,O_ORDERSTATUS,O_TOTALPRICE)
4	=A3.attach(lineitem,#L_LINENUMBER,L_PARTKEY,L_SUPPKEY)
5	=A3.append(A1)
6	=A4.append(A2)
7	=A3.close()

用主表.attach(子表,...)来建立主子表间的依附关系

同步分段

使用多个组表时，可以使用多路游标进行同步分段

例如，统计每位雇员的订单数与销售额，销售数据在雇员组表中，订单数据在订单组表中

	A	B
1	=file("employees.ctx")	=file("orders.ctx")
2	=A1.create()	=B1.create()
3	=A2.attach(stable)	=B2.attach(otable)
4	=A3.cursor@m(;;3)	=B3.cursor(;;A4)
5	=joinx(A4:s,EID;B4:o,EID)	=A5.groups(s.EID:EID;count(~):Count, sum(o.Amount):Sum)

A3（销售数据）和B3（订单数据），分别在两个不同的组表中
A4用函数T.cursor@m(;;n) 将销售数据分为n段生成多路游标
B4用订单数据根据A4中的多路游标同步分段
执行同步分段后，在A5中将A4和B4中的游标用joinx函数将同步的多路游标连接起来
B5对A5连接后的结果进行聚合运算

索引

类似数据库，组表上可以建立索引，以提高查询效率

例如，根据订单号查询订单记录

根据订单号为订单组表建立索引：

	A
1	<code>=file("orders.ctx").create().index(orderkey_idx;O_ORDERKEY)</code>

利用组表的索引查找某订单记录：

	A
1	<code>=file("orders.ctx").create().icursor(O_ORDERKEY==123456).fetch()</code>

内表

组表可以加载进内表，获得更高的访问速度，并支持随机访问

例如，将订单表组表数据加载进内表，查找订单号为123456的订单

	A	B
1	=file("orders.ctx").create().memory()	
2	>A1.index()	/读入时会继承外表的主键，索引要单独建立
3	=A1.find(123456)	

Index	O_ORDERKEY	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_C
1	1	36901	O	173665.47	1996-01...	5-L
2	2	78002	O	46929.18	1996-12...	1-L
3	3	123314	F	193846.25	1993-10...	5-L
4	4	136777	O	32151.78	1995-10...	5-L
5	5	44485	F	144659.2	1994-07...	5-L
6	6	55624	F	58749.59	1992-02...	4-N
7	7	39136	O	252004.18	1996-01...	2-H
8	32	130057	O	208660.75	1995-07...	2-H
9	33	66958	F	163243.98	1993-10...	3-N



O_ORDER...	O_CUSTKEY	O_ORDER...	O_TOTAL...	O_ORDER...	O_C
123456	34162	F	71573.45	1994-07...	2

补文件

组表需要频繁地有序归并新数据，可以利用补文件，避免每次都重整主文件

例如，对客户号有序的订单表组表新增数据（每日归并、每月重整）

增量更新（重整、归并时自动更新索引文件）

	A	B	C
1	If day(now())==1	=file("orders.ctx").reset()	/每月重整，reset将补文件有序归并到主文件，同时清空补文件
2	=file("orders.ctx").create().append@a(dataToday)		/每日归并，@a代表归并追加到补文件上，没有补文件则创建

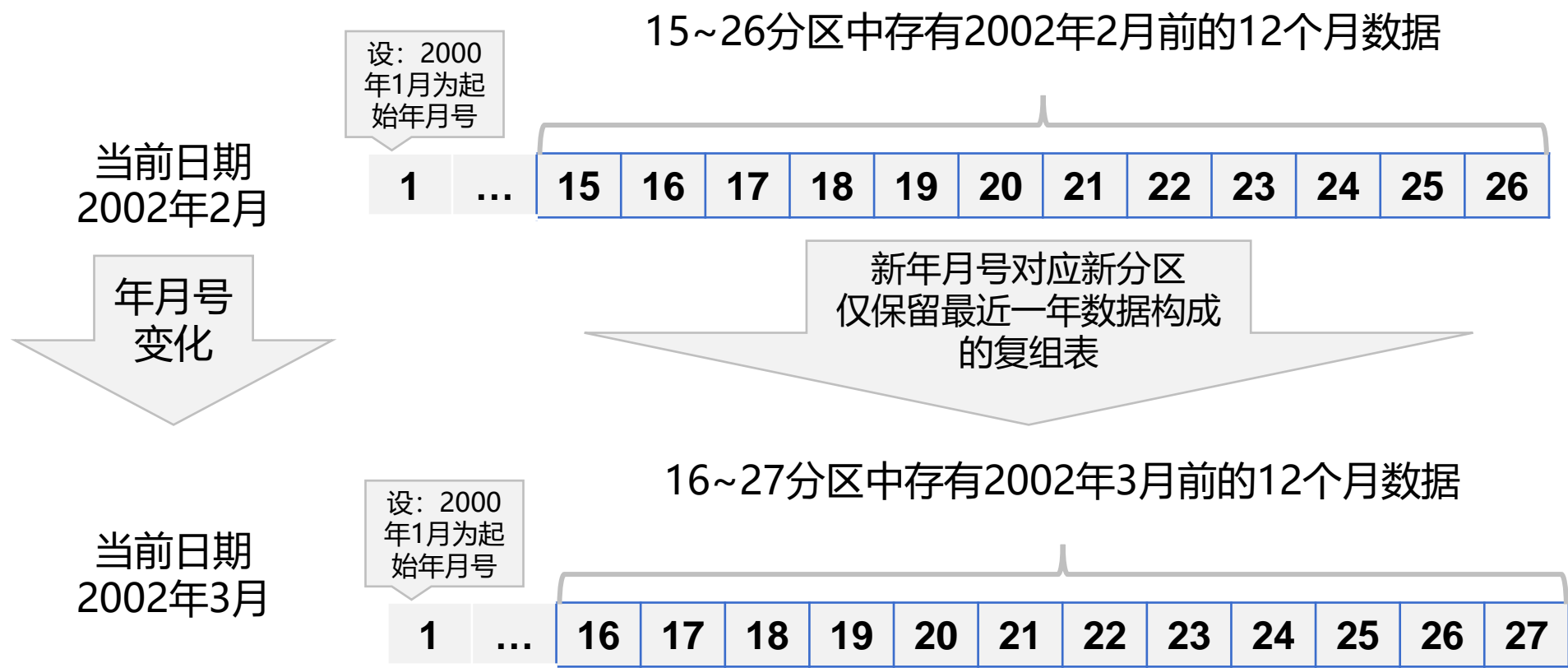
包含补文件的查询

	A	C
1	=file("orders.ctx")	/包含补文件
2	=A1.open().icursor(;C_CUSTKEY==36901).fetch()	/查询

分区

利用分区可以将大组表拆分成多份同结构组表（组文件），且按分区有序
分区的目的在于容易管理，方便组合

例如，某电商的订单表组表数据，仅保存一年以内的数据



分区

利用分区可以将大组表拆分成多份同结构组表（组文件），且按分区有序分区的目的在于容易管理，方便组合

例如，某电商的订单表组表数据，仅保存一年以内的数据

按年月号创建名称相同分区号不同的多个同结构文件（组文件），并产生复组表

	A	B
1	=12*(year(now())-begyear-1)+month(now())	/去年当月年月号，设分区1为初始年（begyear）1月的年月号
2	>ym=12.(~+A1)	/当前日期前12个月的年月号序列
3	=file("orders.ctx":ym)	/ym对应12个分区，其中每个小组表名称均为orders.ctx
4	=A3.create(#O_ORDERKEY, O_ORDERDATE,O_TOTALPRICE;; 12*(year(O_ORDERDATE)-inityear-1)+month(O_ORDERDATE))	/按“年月号表达式”创建分区组表文件
5	=A4.append@x(orders)	/@x表示游标orders按“年月号表达式”对应复组表的多个分区追加数据

复组表的运算函数和普通组表相同

	A	B
1	=file("orders.ctx":ym).create().cursor()	/组文件多路游标，ym为年月号序列
2	=A1.groups(month(O_ORDERDATE):month;count(~):count)	/统计每月订单数

THANKS

感谢观看

