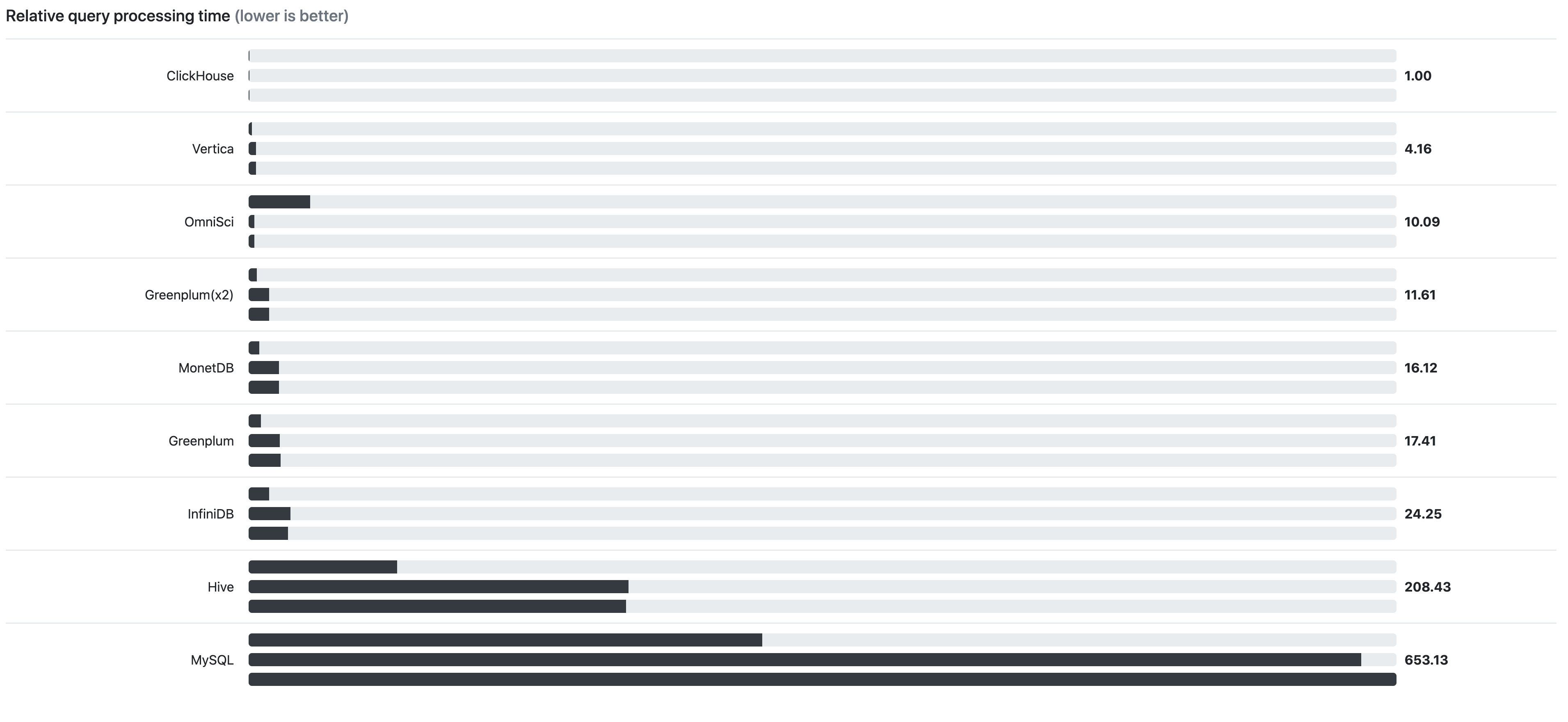
**盘古讲堂——ClickHouse原理、应用以及优化（精简版）**

|  |
| --- |
| 小李飞刀，例无虚发，只出一刀，无人能挡，只因天下武功无坚不摧，唯快不破。 —— 《小李飞刀》 |

**一、ClickHouse 简介**

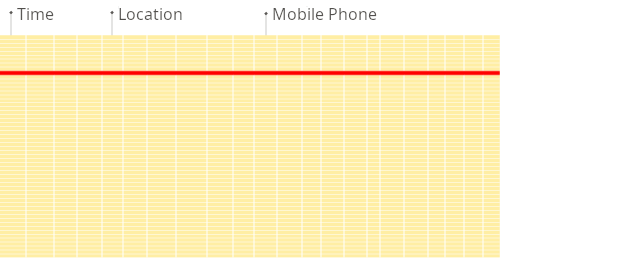
ClickHouse是一个开源的列式数据库管理系统，专为大规模数据分析和OLAP（联机分析处理）而设计。它最初由俄罗斯搜索引擎公司Yandex开发，旨在处理海量数据，并提供高性能的数据查询和分析能力。

ClickHouse性能对比报告：



所有用于对比的数据库都使用了相同配置的服务器，在单个节点的情况下，对一张拥有133个字段的数据表分别在1000万、1亿和10亿三种数据体量下执行基准测试，基准测试的范围涵盖43项SQL查询。

行式数据库：



列式数据库：



ClickHouse性能报告：https://benchmark.clickhouse.com/

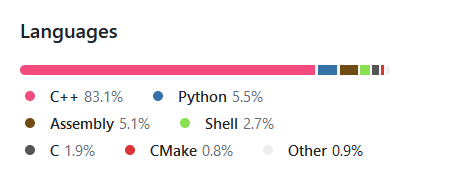
ClickHouse官网地址：https://clickhouse.com/

ClickHouse官方博客：https://clickhouse.com/blog

ClickHouse版本变更：https://clickhouse.com/docs/en/whats-new/changelog

截至2023年10月25日，当前最新版本为：release 23.9 LTS版本：release 23.8 LTS

ClickHouse源码地址：https://github.com/ClickHouse/ClickHouse 主要是c和c++

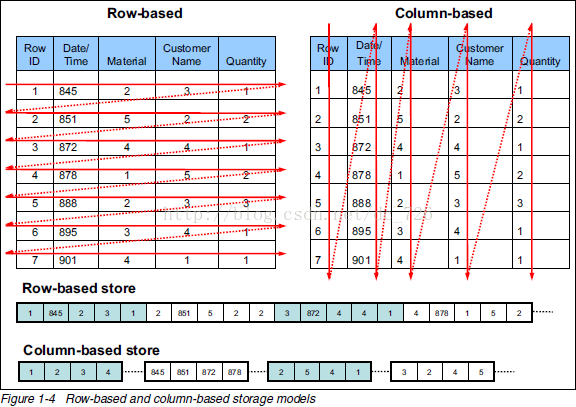


ClickHouse源码编译：[ClickHouse源码编译](https://venusgroup.feishu.cn/docx/WFp5dRczzoxzIrxV6l2cg2mSnwg)

**二、Clickhouse 特点**

**1.列式存储**

行式与列式存储区别：



在基于行式存储的数据库中， 数据是按照行数据为基础的逻辑存储单元进行存储的，一行中的数据在存储介质中以连续存储形式存在；

在基于列式存储的数据库中， 数据是按照列数据为基础的逻辑存储单元进行存储的，一列中的数据在存储介质中以连续存储形式存在。

按列存储与按行存储相比，前者可以有效减少查询时所需扫描的数据量。假设一张数据表A拥有50个字段A1～A50，以及100行数据。现在需要查询前5个字段并进行数据分析，则可以用如下SQL实现：

|  |
| --- |
| SQL SELECT A1，A2，A3，A4，A5 FROM A |

如果数据按行存储，数据库首先会逐行扫描，并获取每行数据的所有50个字段，再从每一行数据中返回A1～A5这5个字段。不难尽管只需要前面的5个字段，但由于数据是按行进行组织的，实际上还是扫描了所有的字段。

如果数据按列存储，就不会发生这样的问题。由于数据按列组织，数据库可以直接获取A1～A5这5列的数据，从而避免了多余的数据扫描。

**2.数据压缩**

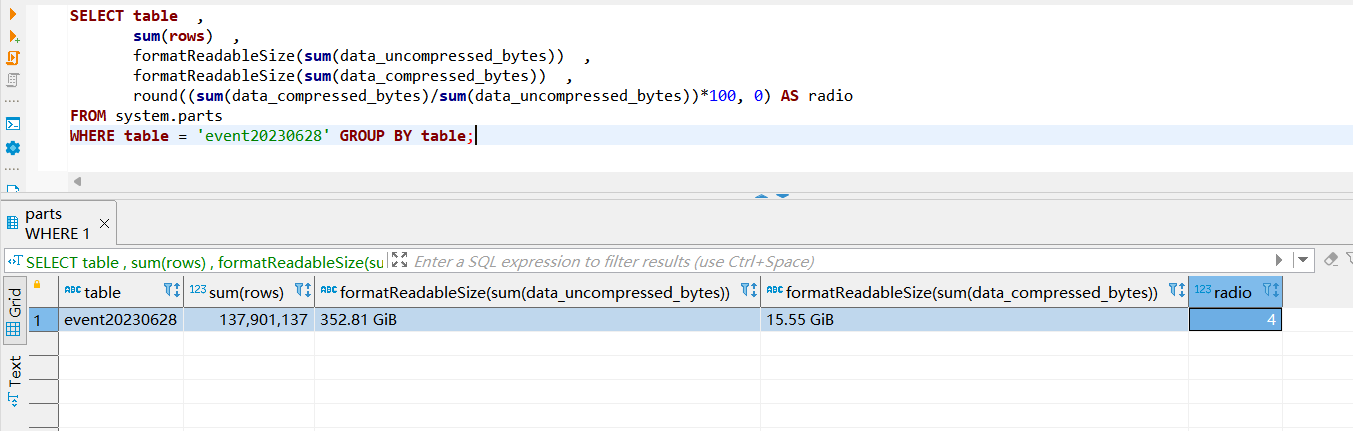
如果想让数据查询变得更快，最简单有效的方法是减少数据扫描范围和数据传输时的大小，而列式存储和数据压缩就可以帮助我们实现上述两点。列式存储和数据压缩通常是伴生的，因为一般来说列式存储是数据压缩的前提。

假设有两个字符串abcdefghi和bcdefghi，现在对它们进行压缩，如下所示：

|  |
| --- |
| Plain Text 压缩前：abcdefghi\_bcdefghi 压缩后：abcdefghi\_(9,8) |

可以看到，压缩的本质是按照一定步长对数据进行匹配扫描，当发现重复部分的时候就进行编码转换。例如上述示例中的(9,8)，表示如果从下划线开始向前移动9个字节，会匹配到8个字节长度的重复项，即这里的bcdefghi。真实的压缩算法自然比这个示例更为复杂，但压缩的实质就是如此。扩展：[Lz4压缩算法学习（转）](https://venusgroup.feishu.cn/docx/OuZbdwdsbo3Awhxb2BSc2NfrnOf)

Clickhouse默认使用LZ4算法压缩，在Yandex.Metrica的生产环境中，数据总体的压缩比可以达到8:1 。在我们部门的生产环境中，压缩比大概为22：1（由于大量字段都为空值）



数据中的重复项越多，则压缩率越高；压缩率越高，则数据体量越小；而数据体量越小，则数据在网络中的传输越快，对网络带宽和磁盘IO的压力也就越小。属于同一个列字段的数据，因为它们拥有相同的数据类型和现实语义，重复项的可能性自然就更高。

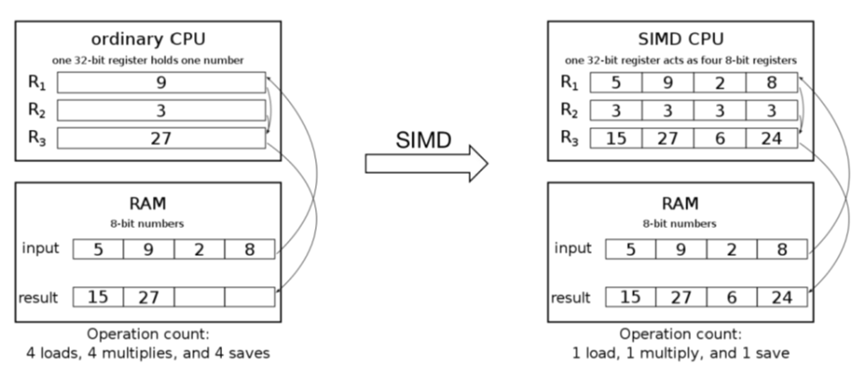
列式存储除了降低IO和存储的压力之外，还为向量化执行做好了铺垫。

**3.向量化执行引擎**

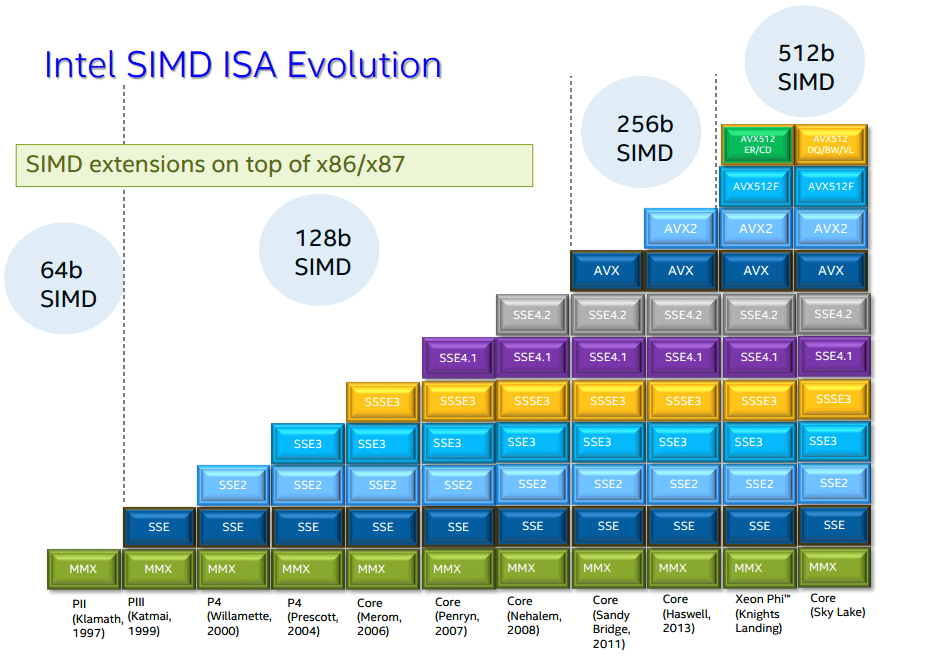
向量化是指计算从一次对一个值进行运算转换为一次对一组值进行运算的过程。

为了实现向量化执行，需要利用CPU的SIMD技术。SIMD的全称是Single Instruction Multiple Data，即用单条指令操作多条数据，它的原理是在CPU寄存器层面实现数据的并行操作。

例如，具有 128 位寄存器的 CPU可以保存 4 个 32 位数并进行一次计算，当我们在内存当中有4个32位的int数据时，传统的CPU不支持SIMD，进行计算时需要从内存中Load数据4次，再进行4次乘法计算，然后把结果写回到内存当中，这个过程同样要进行4次。假如CPU支持SIMD，就可以一次载入多个连续的内存数据，一次计算多对操作数以及一次写入多个计算结果。这样的话理论上会比传统的CPU快4倍。



支持 SIMD 的指令集有很多。各种 CPU 架构都提供各自的 SIMD 指令集，以下是Intel SIMD的发展历史：



ClickHouse目前利用SSE4.2指令集实现向量化执行。

**4.多样化的表引擎**

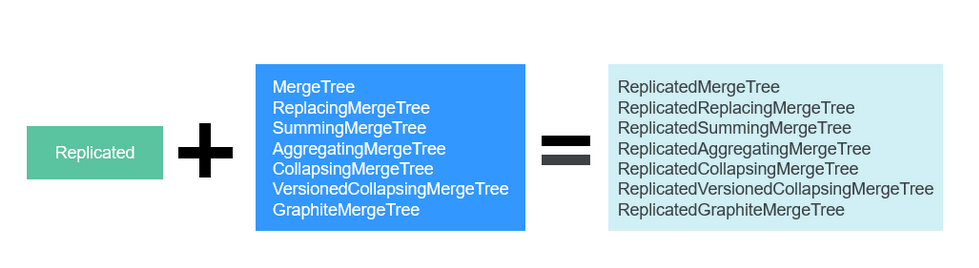
ClickHouse将存储部分进行了抽象，把存储引擎作为一层独立的接口。通过特定的表引擎支撑特定的场景。



完整表引擎：https://clickhouse.com/docs/en/engines/table-engines

表引擎使用：[ClickHouse表引擎使用介绍](https://venusgroup.feishu.cn/docx/LEGXdVnNGozCaPxOusAcrtNrnAe)

MergeTree是基础引擎，有主键索引、数据分区、数据副本、数据采样、删除和修改等功能，  
ReplacingMergeTree 去重功能，  
SummingMergeTree 汇总求和功能，  
AggregatingMergeTree 聚合功能，  
CollapsingMergeTree 折叠删除功能，  
VersionedCollapsingMergeTree 版本折叠功能，  
GraphiteMergeTree 压缩汇总功能，可用于处理图数据，  
在这些的基础上还可以叠加Replicated，组合出拥有副本协同的能力的新引擎

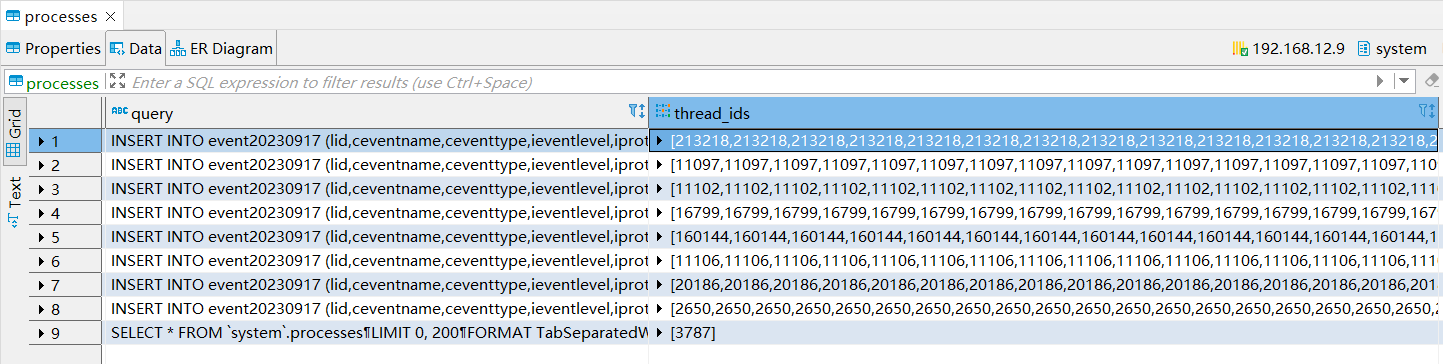


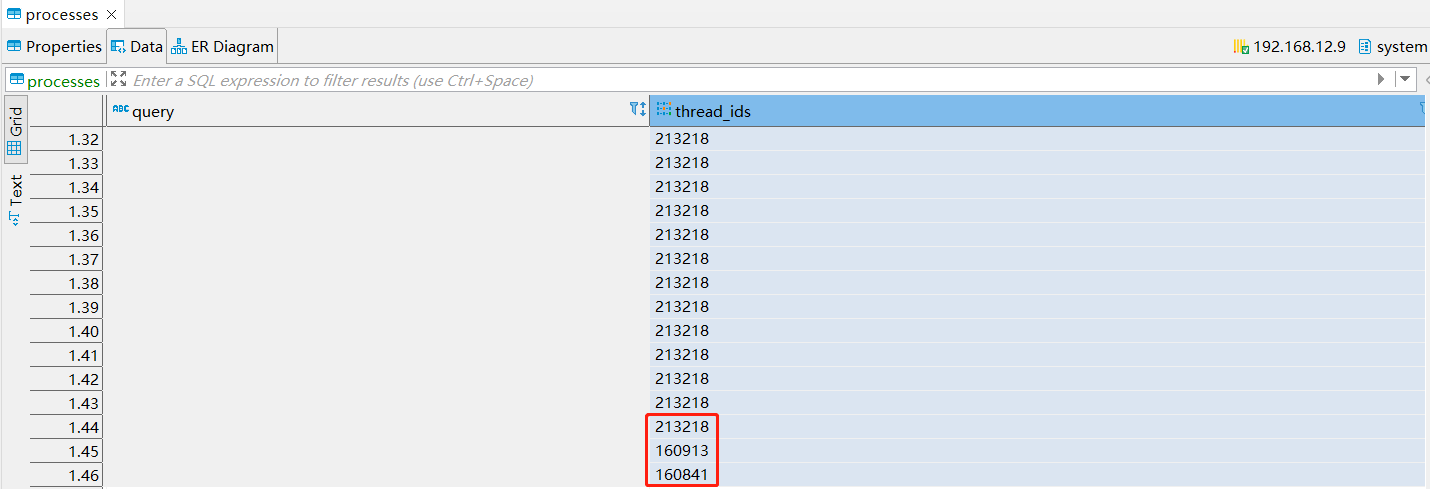
**5.多线程与分布式**

（1）多线程

如果说向量化执行是通过数据级并行的方式提升了性能，那么多线程处理就是通过线程级并行的方式实现了性能的提升。相比基于底层硬件实现的向量化执行SIMD，线程级并行通常由更高层次的软件层面控制。由于SIMD不适合用于带有较多分支判断的场景，ClickHouse也大量使用了多线程技术以实现提速，以此和向量化执行形成互补。

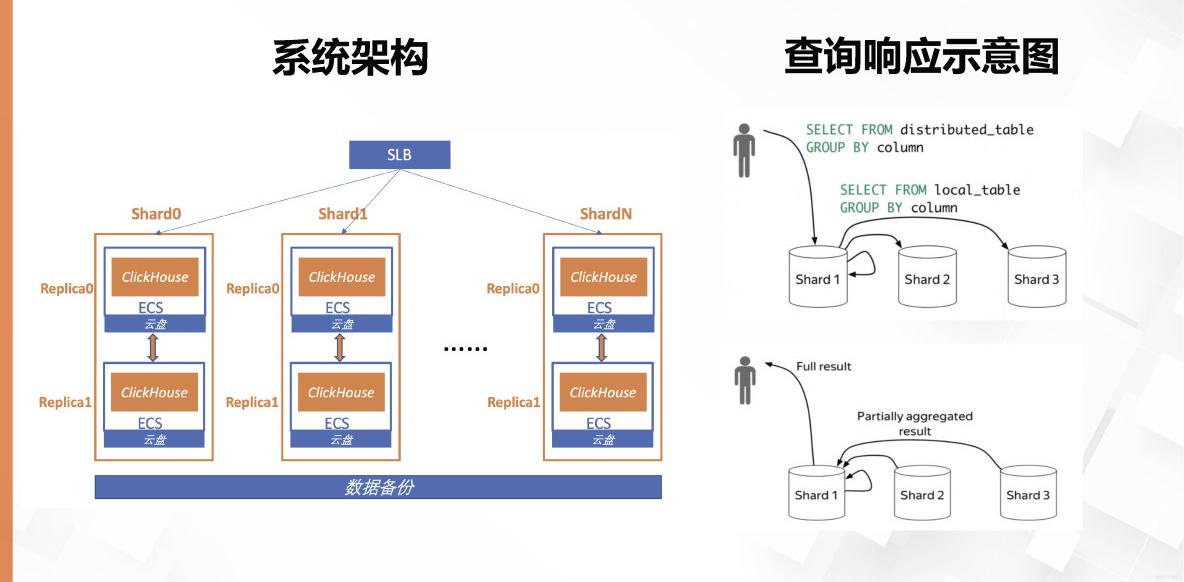
system.processes表查看运行任务：





（2）分布式

ClickHouse在数据存取方面，既支持分区（纵向扩展，利用多线程原理），也支持分片（横向扩展，利用分布式原理），如果一台服务器性能吃紧，那么就利用多台服务的资源协同处理，预先将数据分布到各台服务器，将数据的计算查询直接下推到数据所在的服务器，然后汇聚出各分片上的计算结果返回。



**6.多主架构**

ClickHouse则采用Multi-Master多主架构，集群中的每个节点角色对等，客户端访问任意一个节点都能得到相同的效果。多主架构使系统架构变得更加简单，集群中的所有节点功能相同，天然规避了单点故障的问题，非常适合用于多数据中心、异地多活的场景。

**7.极致算法优化**

ClickHouse在算法方面有极致追求，例如：在字符串搜索方面，针对不同的场景，ClickHouse最终选择了这些算法：

（1）对于常量，使用Volnitsky算法；

（2）对于非常量，使用CPU的向量化执行SIMD，暴力优化；

（3）正则匹配使用re2和hyperscan算法。

去重计数uniqCombined函数，会根据数据量的不同选择不同的算法：

（1）当数据量较小的时候，会选择Array保存；

（2）当数据量中等的时候，会选择HashSet；

（3）而当数据量很大的时候，则使用HyperLogLog算法

**三、Clickhouse 基本概念与结构**

**1.表结构说明**

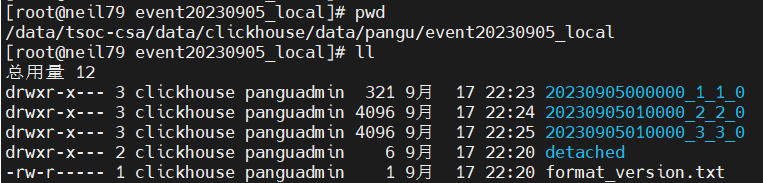
|  |
| --- |
| SQL CREATE TABLE pangu.event20230905\_local //数据库名.表名 (  `lid` Int64 DEFAULT 0, // Int64类型，默认值是0  `ceventname` LowCardinality(String), // 低基数类型，对应字符串类型  `ceventtype` LowCardinality(String),  `ieventlevel` Int32 DEFAULT 0,  `lrecepttime` Int64 DEFAULT 0,  `csrcip` String DEFAULT '',  `cdstip` String DEFAULT '',  `ceventmsg` String DEFAULT '',  `eventdate` DateTime,  //二级索引，也叫跳数索引，用来帮助查询时减少数据扫描的范围  INDEX ceventmsg\_tokenbf\_index ceventmsg TYPE tokenbf\_v1(30720, 3, 0) GRANULARITY 1,  PROJECTION p\_ieventlevel\_csrcip //投影  (  SELECT   toUnixTimestamp(toStartOfDay(fromUnixTimestamp64Milli(lrecepttime))) \* 1000 AS t,  t + 604800000 AS et,  ieventlevel,  csrcip,  count() AS ct  GROUP BY   t,  ieventlevel,  csrcip  ) ) ENGINE = MergeTree //引擎类型 PARTITION BY toYYYYMMDDhhmmss(eventdate) //分区键  ##PRIMARY KEY (lrecepttime, lid, ieventlevel) //主键，一级索引 //排序键，指定在一个数据片段内，数据以何种标准排序。默认情况下主键（PRIMARY KEY）与排序键相同。  ORDER BY (lrecepttime, lid, ieventlevel)  //它表示索引的粒度，默认值为8192，指每间隔8192行数据才生成一条索引  SETTINGS index\_granularity = 8192;     注：PRIMARY KEY [选填]：主键，顾名思义，声明后会依照主键字段生成一级索引，用于加速表查询。 默认情况下，主键与排序键(ORDER BY)相同，所以通常直接使用ORDER BY代为指定主键，无须刻意通 过PRIMARY KEY声明。所以在一般情况下，在单个数据片段内，数据与一级索引以相同的规则升序排列。  如果ORDER BY与PRIMARY KEY不同，PRIMARY KEY必须是ORDER BY的前缀(为了保证分区内数据和主键的有序性) |

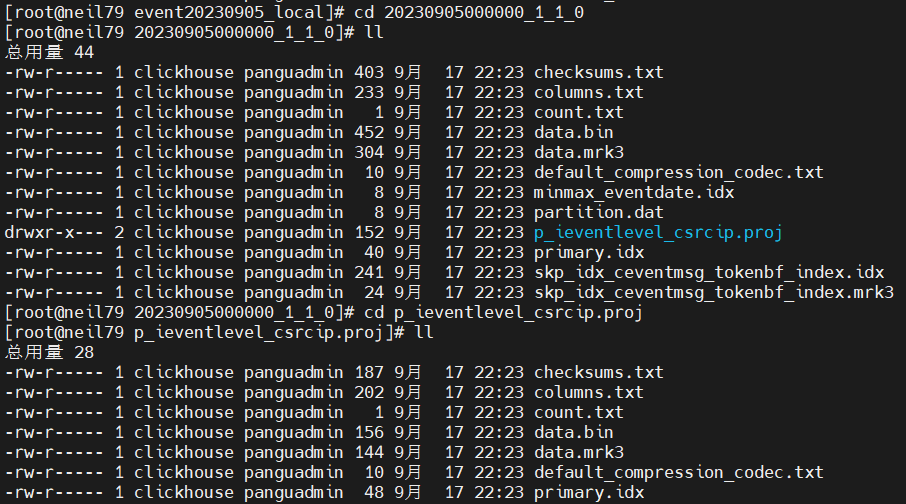
**2.目录结构说明**

用前述表结构创建一个表，执行以下sql插入数据：

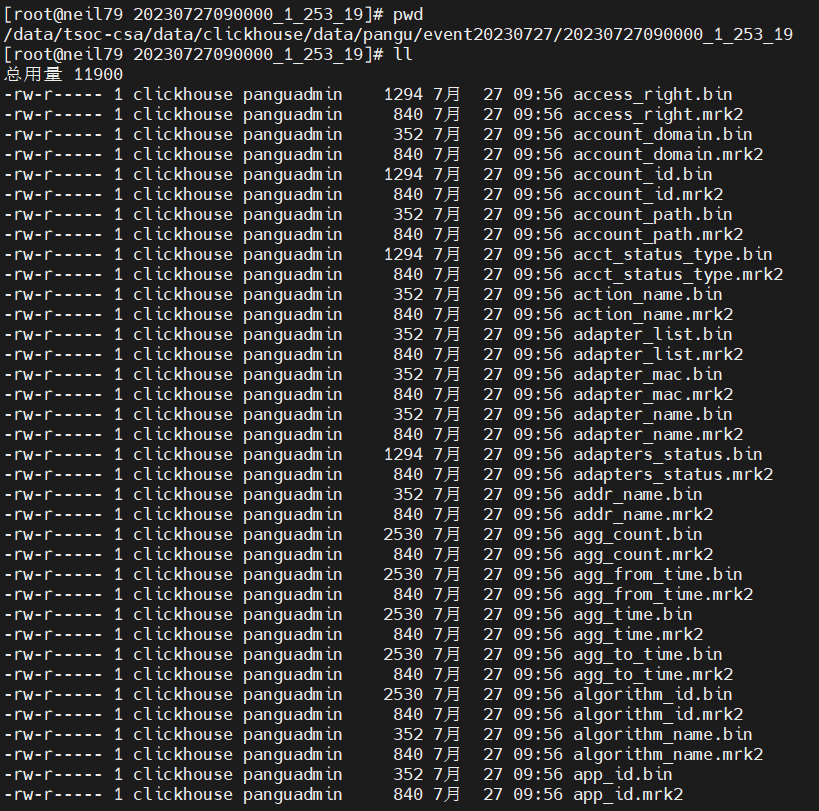
|  |
| --- |
| SQL insert into event20230905\_local values  (1, 'eventname1', 'eventtype1', 1, 1693843200000, '192.168.1.1', '192.168.1.2', 'test mesg 111', '2023-09-05 00:00:00'), (2, 'eventname2', 'eventtype2', 2, 1693843200000, '192.168.1.1', '192.168.1.2', 'test mesg 222', '2023-09-05 00:00:00'), (3, 'eventname3', 'eventtype3', 3, 1693843200000, '192.168.1.1', '192.168.1.2', 'test mesg 333', '2023-09-05 00:00:00');  insert into event20230905\_local values  (4, 'eventname4', 'eventtype4', 1, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 444', '2023-09-05 01:00:00'), (5, 'eventname5', 'eventtype5', 2, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 555', '2023-09-05 01:00:00'), (6, 'eventname6', 'eventtype6', 3, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 666', '2023-09-05 01:00:00');  insert into event20230905\_local values  (7, 'eventname4', 'eventtype4', 1, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 777', '2023-09-05 01:00:00'), (8, 'eventname5', 'eventtype5', 2, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 888', '2023-09-05 01:00:00'), (9, 'eventname6', 'eventtype6', 3, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 999', '2023-09-05 01:00:00'); |

进入后台查看：





与各列分开存储的对比：



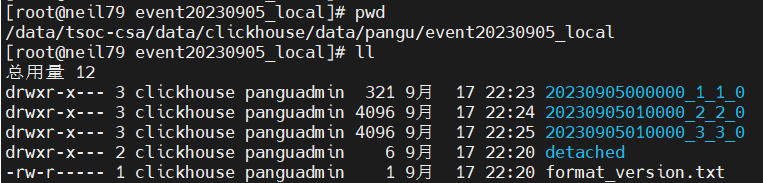
|  |
| --- |
| 注意：   1. 当采用wide part模式时会为每一列生成[column].bin文件，文件名就是列的名字。 2. 另外一种模式是compact part模式，这种模式下所有的列的数据放在一个文件data.bin里面。 3. 当数据的压缩之前的原始大小超过阈值min\_bytes\_for\_wide\_part，或者数据行数超过阈值min\_rows\_for\_wide\_part 时，就会采用wide part模式，否则采用compact part模式。 4. 可在创建表语句中添加上述配置为0，即可用wide part模式存储列信息：   SETTINGS min\_bytes\_for\_wide\_part=0, min\_rows\_for\_wide\_part=0, index\_granularity = 8192; |

|  |
| --- |
| Plain Text 目录/文件分析： [root@neil79 event20230905\_local]# tree . |-- 20230905000000\_1\_1\_0 #分区值\_最小分区块编号\_最大分区块编号\_合并层级 （toYYYYMMDDhhmmss(eventdate)） | |-- checksums.txt #校验文件，用于校验各个文件的正确性。存放各个文件的size以及hash值 | |-- columns.txt #列信息，可以查看列名以及列属性 | |-- count.txt #当前分区目录下数据总行数 | |-- data.bin #数据文件 | |-- data.mrk3 #标记文件（标记文件在 idx索引文件 和 bin数据文件 之间起到了桥梁  #作用，一般是记录了列的偏移量） | |-- default\_compression\_codec.txt #默认数据压缩格式 | |-- minmax\_eventdate.idx #用于记录当前分区下分区字段对应原始数据的最小和最大值  | |-- partition.dat #用于保存当前分区下分区表达式最终生成的值，如：20230905000000  | |-- p\_ieventlevel\_csrcip.proj #投影目录 | | |-- checksums.txt | | |-- columns.txt | | |-- count.txt | | |-- data.bin | | |-- data.mrk3 | | |-- default\_compression\_codec.txt | | `-- primary.idx | |-- primary.idx #一级索引（主键索引）文件，用于存放稀疏索引，加快查询效率 | |-- skp\_idx\_ceventmsg\_tokenbf\_index.idx #二级索引（跳数索引）文件 | `-- skp\_idx\_ceventmsg\_tokenbf\_index.mrk3 #二级索引标记文件 |-- 20230905000000\_2\_2\_0 |-- ......同上 |-- 20230905000000\_3\_3\_0 |-- ......同上 |

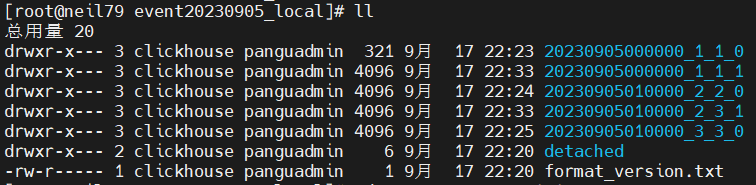
**3.分区合并**

分区合并是针对同一个分区内不同part目录之间进行的（后台自动触发，间隔**10 到 15 分钟**）。

对于每个新建的分区part目录而言，它们的最小分区块编号与最大分区块编号取值相同，均来源于表内全局自增的BlockNum。BlockNum初始为1，每次新建目录或执行mutation操作（删除或修改）后累计加1。

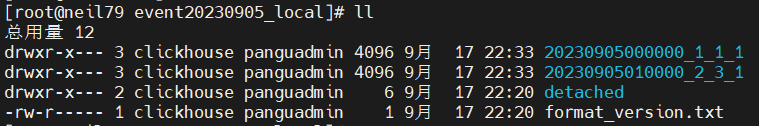


在数据库客户端执行：optimize table event20230905\_local final; 触发手动合并分区后：



|  |
| --- |
| 注意：当执行optimize table event20230905\_local final触发分区part合并时，无数据合并的分区part也会生成一个新目录，合并次数递增1，因此非必要不操作，或者操作时指定具体分区名称 |

几分钟后（默认8分钟）再次查看：

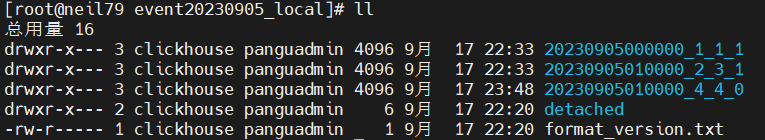


分区part目录在发生合并之后，旧的分区part目录并没有被立即删除，而是会存留一段时间。但是旧的分区part目录已不再是激活状态（即active=0，在system.parts表中可以看到active的值）。所以在数据查询时，它们会被自动过滤掉。

再插入一批数据：

|  |
| --- |
| SQL  insert into event20230905\_local values  (10, 'eventname4', 'eventtype4', 1, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 777', '2023-09-05 01:00:00'), (11, 'eventname5', 'eventtype5', 2, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 888', '2023-09-05 01:00:00'), (12, 'eventname6', 'eventtype6', 3, 1693846801000, '192.168.1.1', '192.168.1.2', 'test mesg 999', '2023-09-05 01:00:00'); |

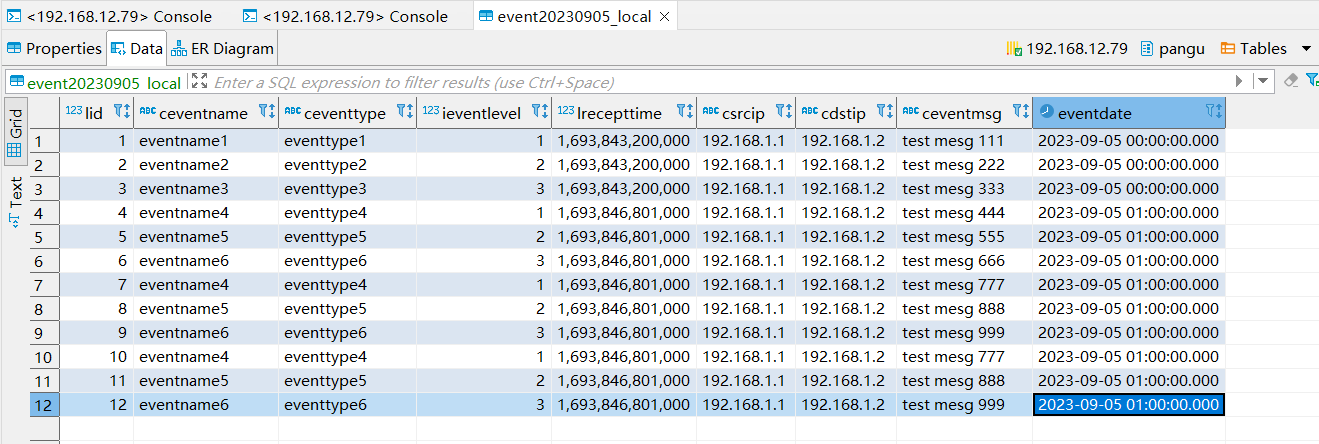
查看新目录结构：



|  |
| --- |
| 小结：每写入一批数据，服务器上都会产生一个对应的分区目录part文件，分区part目录在后台合并时会创建新目录，再删除旧目录，因此尽量避免小批量频繁写入。 |

**4.数据排序**

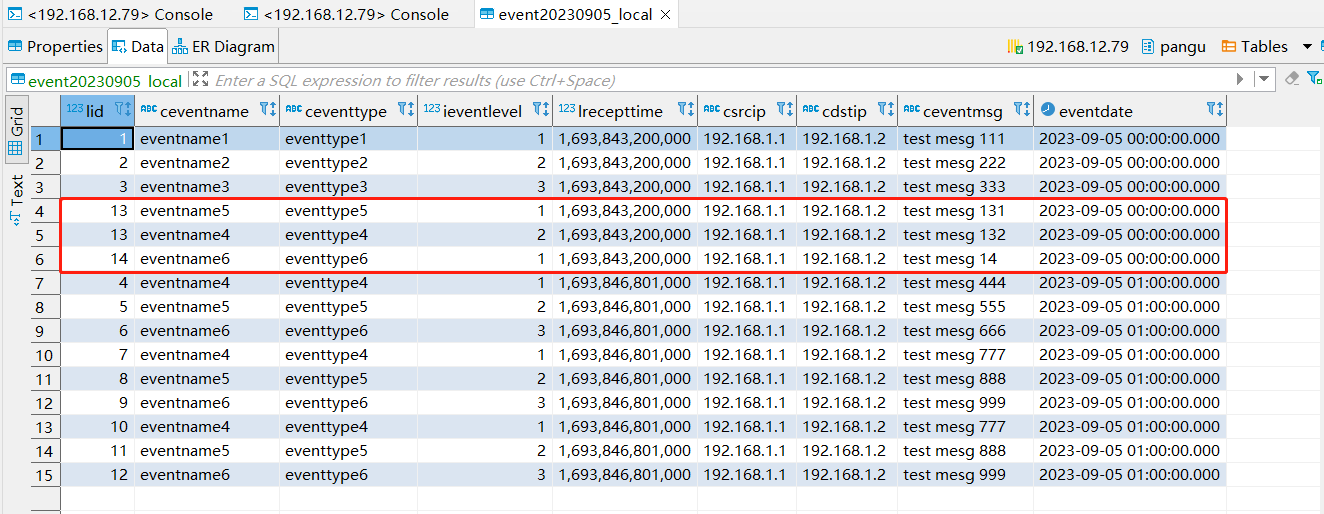
插入前面12条数据后的顺序：



插入乱序数据：

|  |
| --- |
| SQL insert into event20230905\_local values  (14, 'eventname6', 'eventtype6', 1, 1693843200000, '192.168.1.1', '192.168.1.2', 'test mesg 14', '2023-09-05 00:00:00'); insert into event20230905\_local values  (13, 'eventname4', 'eventtype4', 2, 1693843200000, '192.168.1.1', '192.168.1.2', 'test mesg 132', '2023-09-05 00:00:00'); insert into event20230905\_local values  (13, 'eventname5', 'eventtype5', 1, 1693843200000, '192.168.1.1', '192.168.1.2', 'test mesg 131', '2023-09-05 00:00:00');  optimize table event20230905\_local final; //手动触发合并后，将前面3条数据对应的3个分区文件进行合并，合并时按ORDER BY字段排序。 |

插入新数据后查看顺序： ORDER BY (lrecepttime, lid, ieventlevel)

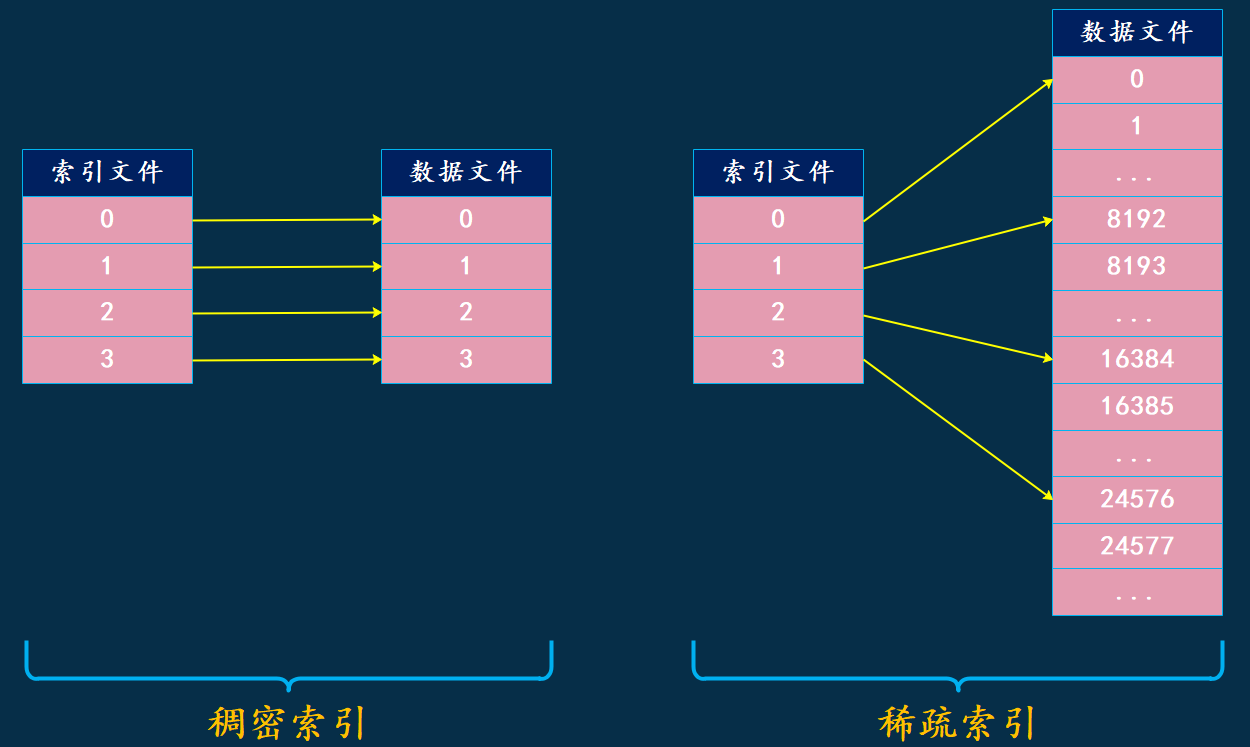


**5.数据组织结构**

**5.1 一级索引**

MergeTree 的主键使用 PRIMARY KEY 定义，主键定义之后，MergeTree 会依据 index\_granularity 间隔（默认 8192 行）为数据表生成一级索引并保存至 primary.idx 文件中，并按照主键进行排序。如果不指定 PAIMARY KEY，那么主键默认和排序键（ORDER BY）相同，在这种情况下，索引（primary.idx）和数据（data.bin）会按照完全相同的规则排序。

（1）稀疏索引



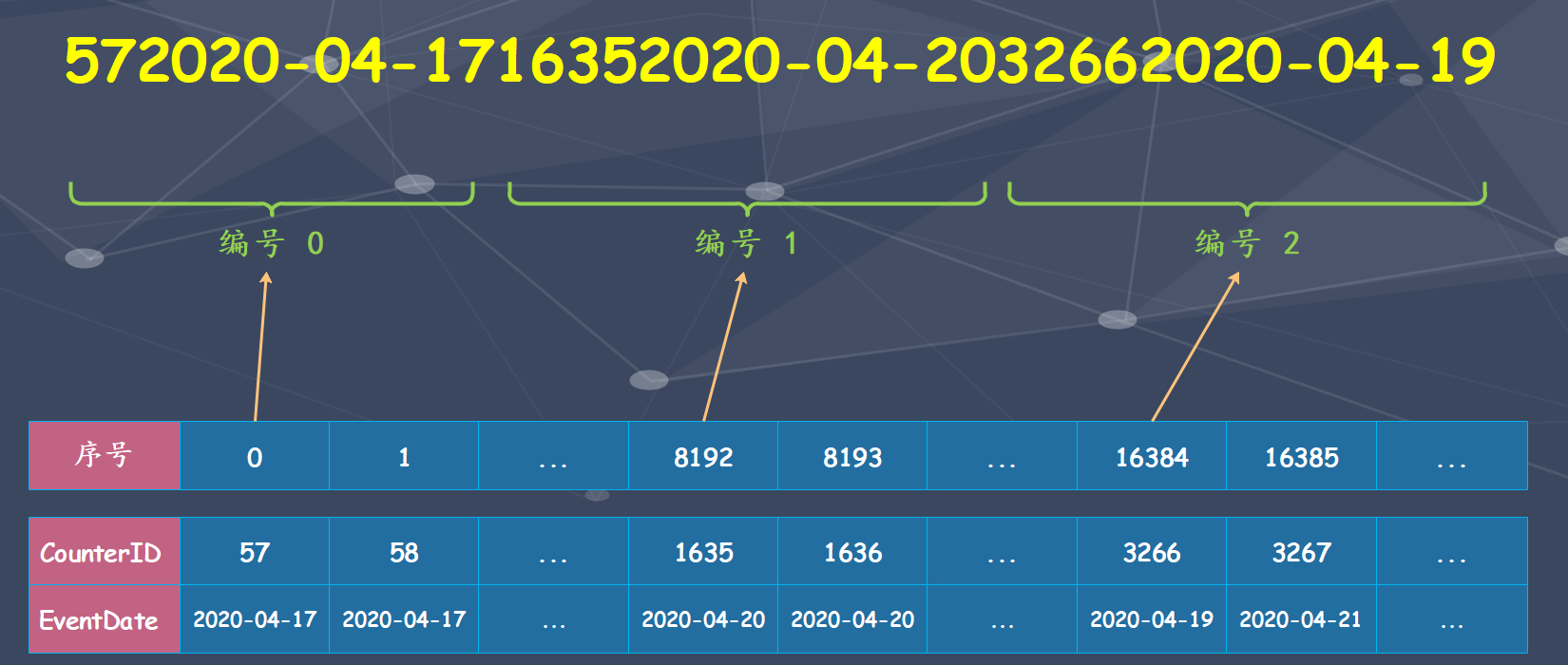
在稠密索引中每一行索引标记都会对应到一行具体的数据记录。

在稀疏索引中，每一行索引标记对应的是一段数据。

当索引粒度是8192时，MergeTree只需要12208行索引标记就能为1亿行数据记录提供索引 ，因此索引数据可以常驻内存，使索引查询时更快。

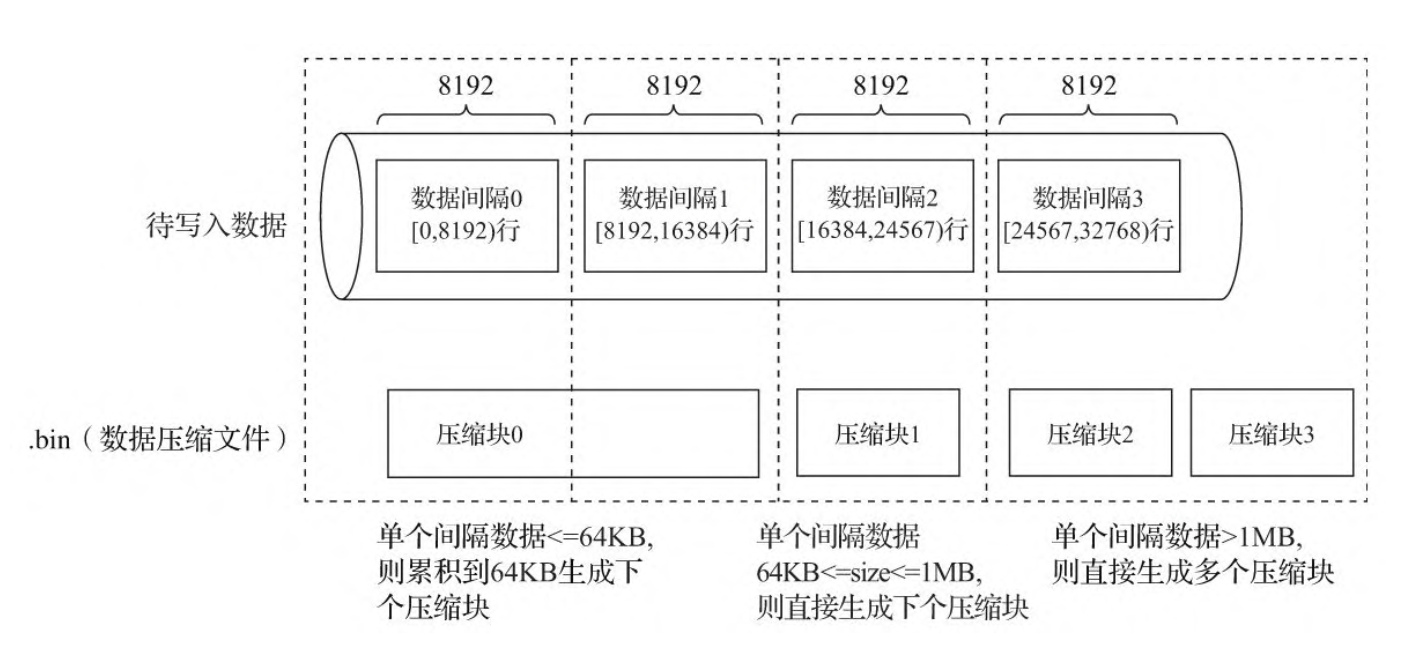
（2）一级索引文件内容

例如 ORDER BY (CounterID, EventDate)，则每间隔 8192 行会同时取 CounterID 和 EventDate 两列的值作为索引值。



**5.2 数据压缩**

在具体读取某一列数据时（.bin文件），首先需要将压缩数据加载到内存并解压，这样才能进行后续的数据处理。通过压缩数据块，可以在不读取整个.bin文件的情况下将读取粒度降低到压缩数据块级别，从而进一步缩小数据读取的范围。



**5.3 mark标记文件**

mark文件内容：



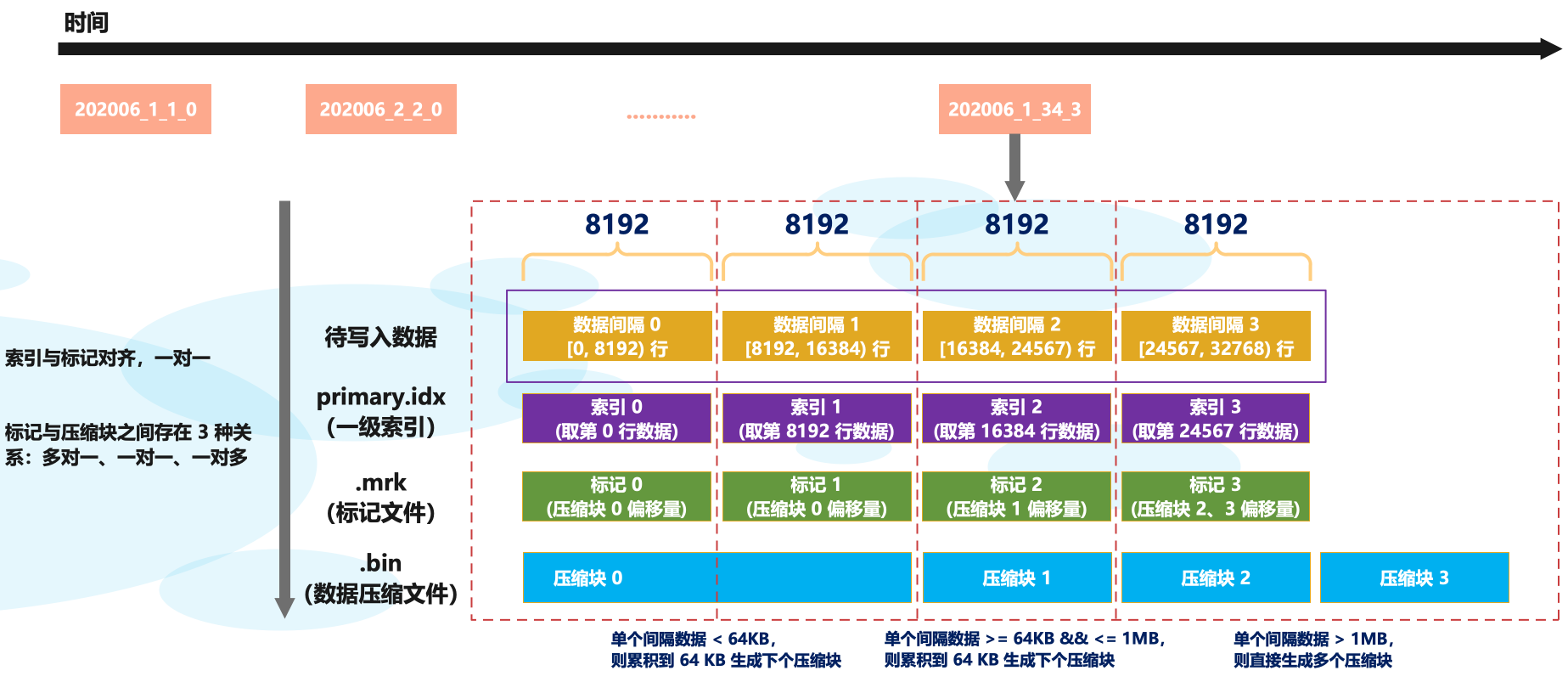
数据标记和索引区间是对齐的，均按照 index\_granularity 的粒度间隔，如此一来只需要简单通过索引下标编号即可直接找到对应的数据标记。并且为了能够与数据衔接，.bin 文件和数据标记文件是一一对应的，即每一个 [Column].bin 文件都有一个 [Column].mrk 数据标记文件与之对应，用于记录数据在 .bin 文件中的偏移量信息。

一行标记数据使用一个元组表示，元组内包含两个整型数值的偏移量信息，分别表示在此段数据区间内：

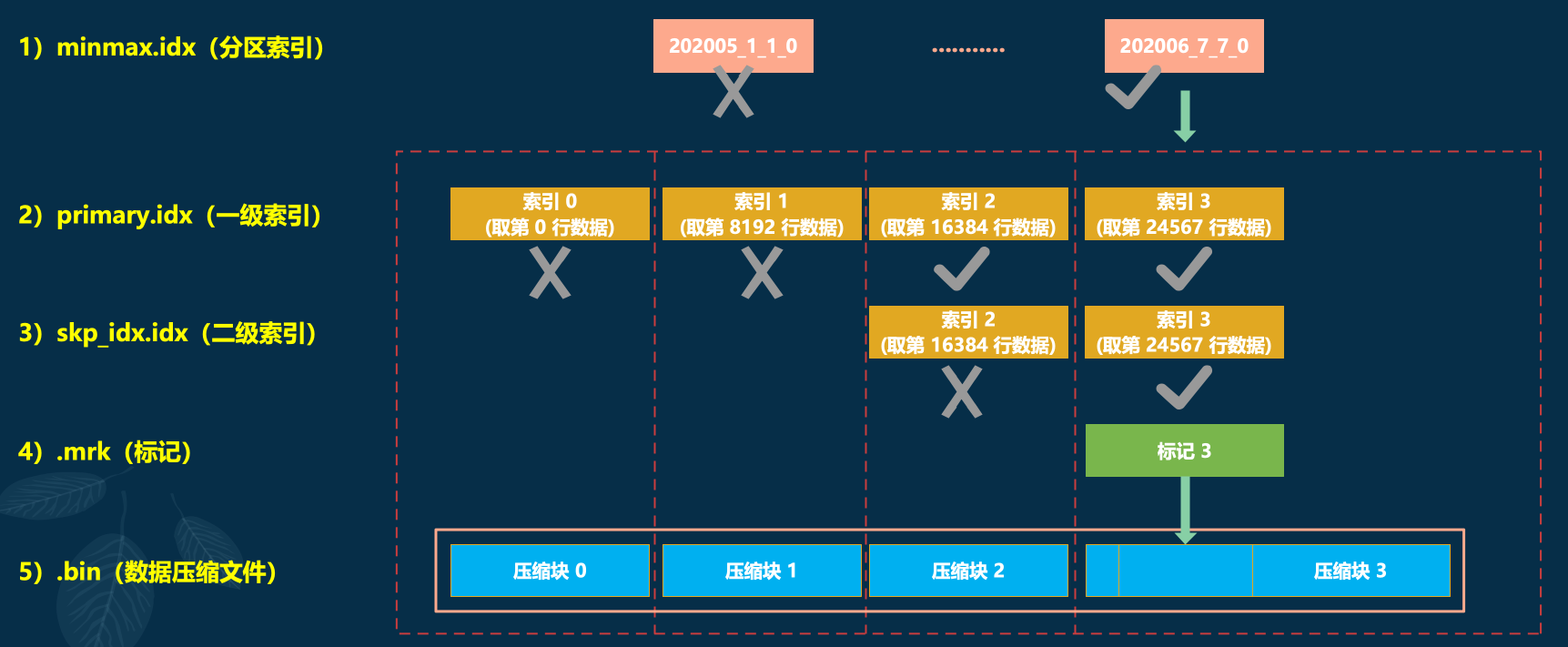
1. 对应 .bin 压缩文件中，压缩数据块的起始偏移量
2. 将该数据块解压缩后，未压缩数据的起始偏移量

偏移量表示的是数据文件中某个数据块的位置相对于文件开头的字节偏移量，偏移量的单位是字节 (byte)，通过记录偏移量，ClickHouse可以根据需要定位和读取数据文件中的特定块，实现高效的数据访问和查询操作，而不需要读取整个.bin文件

**5.4 数据写入**

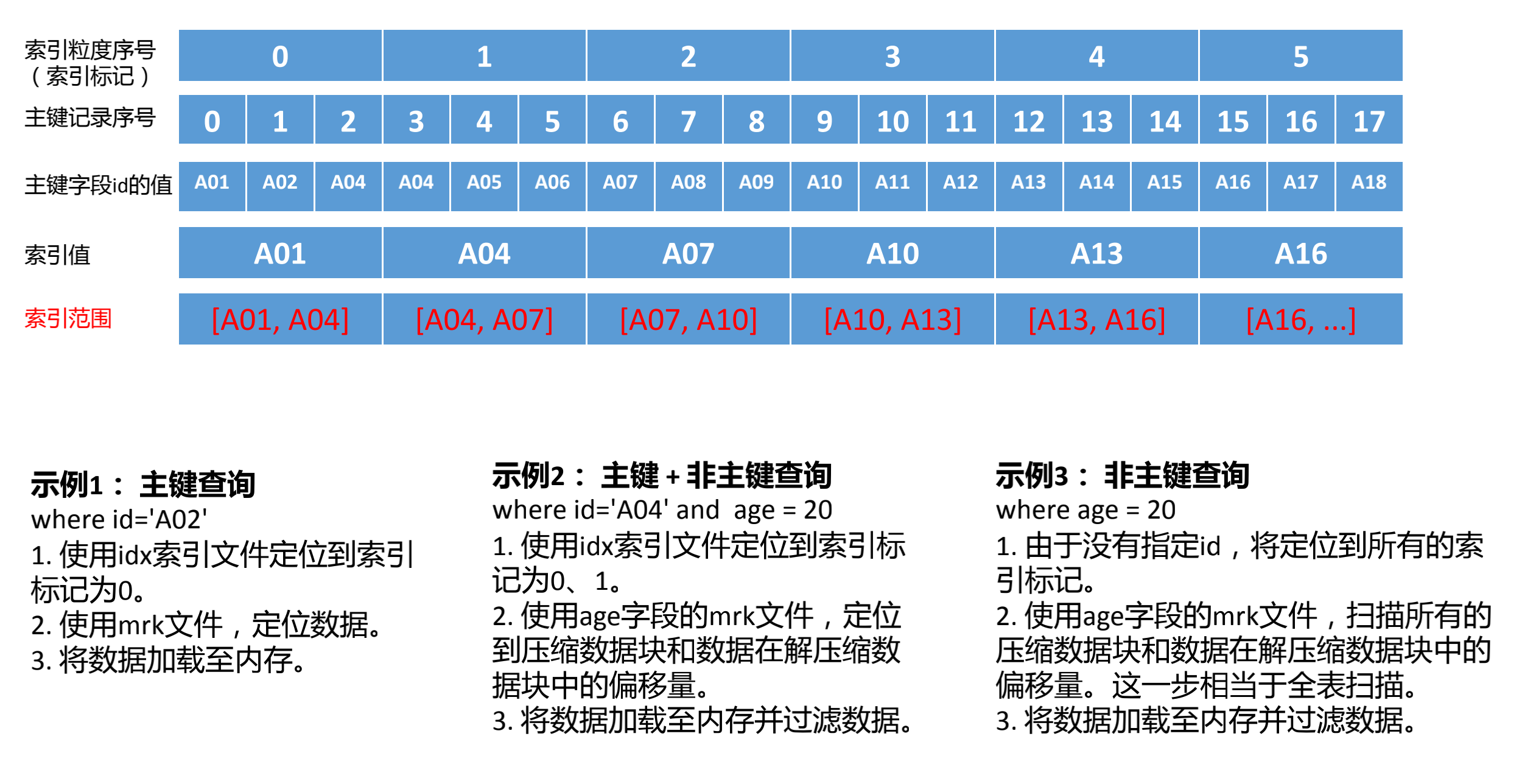


**5.5 数据块筛选过程**



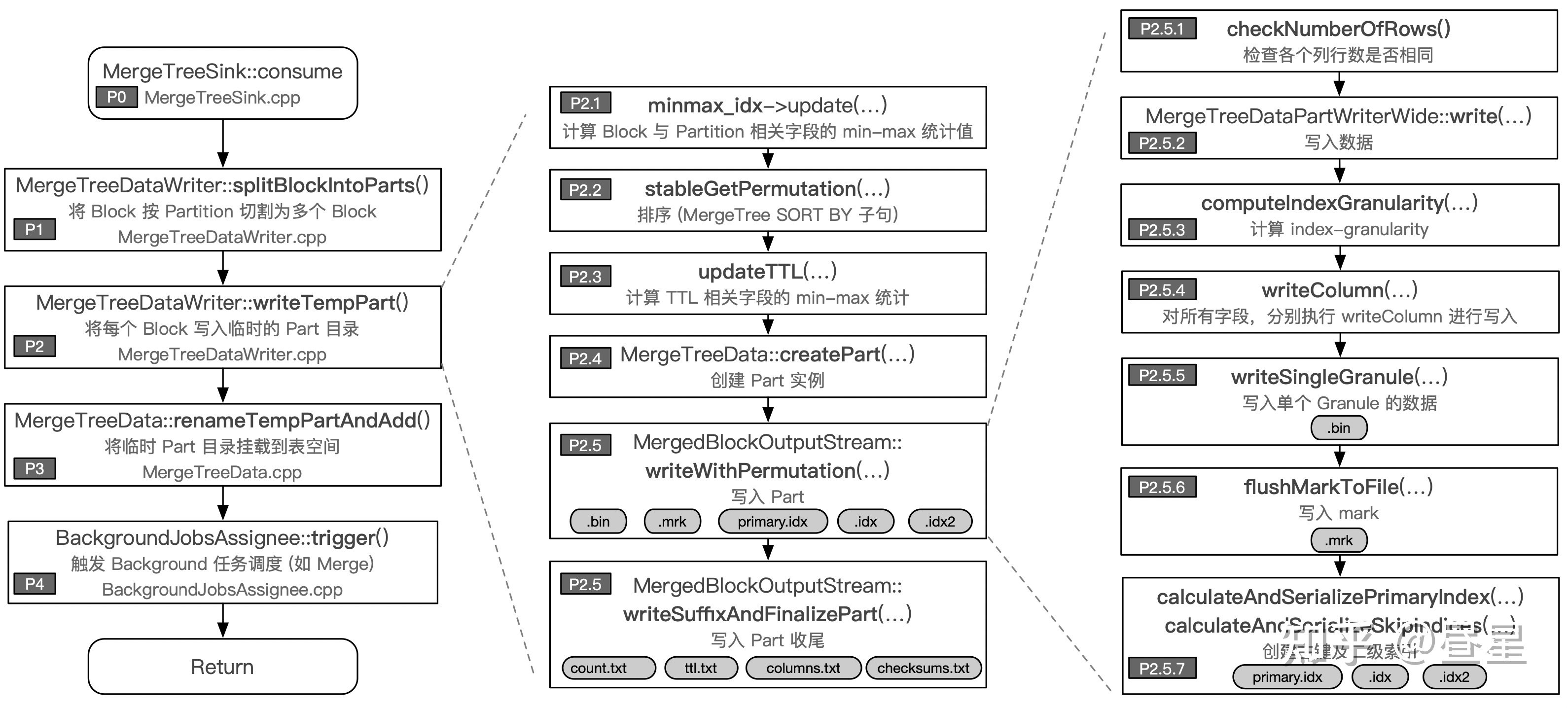


举例说明：

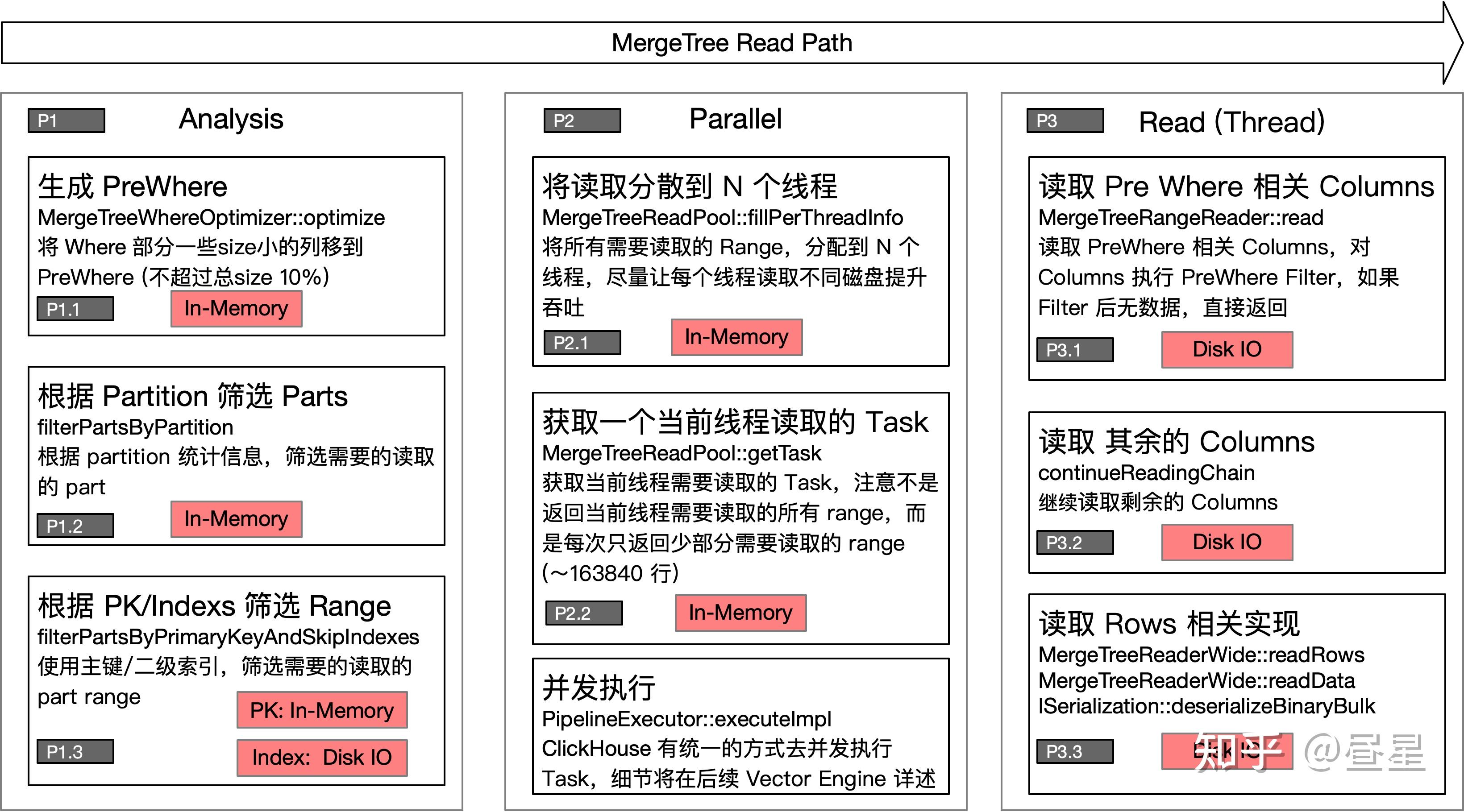


**四、Clickhouse 数据操作流程详解**

**1.数据写入流程**



**2.数据读取流程**



MergeTree 读取过程中的耗时主要分布于:

* 读取二级索引判断需要读取的 Part Ranges
* 读取数据时的 Disk IO
* 将数据反序列化

性能建议:

* 避免 Part 总数过多，把查询时间过多浪费在分析 Part Range 上
* 使用多块磁盘，ClickHouse 可以充分利用磁盘 IO 带宽 (建议使用 RAID)
* 二级索引不是越多越好，如果加的索引不能有效地减少Scan数据量，会有查询 overhead 浪费性能
* PreWhere 子句谨慎手动填写，除非确认自己可以做的更好。

**3.数据的删除与修改**

ClickHouse提供了DELETE和UPDATE的能力，这类操作被称为Mutation（突变），它可以看作ALTER语句的变种。语法如下：

|  |
| --- |
| SQL //删除数据： ALTER TABLE [db\_name.]table\_name DELETE WHERE filter\_expr  或  DELETE FROM [db.]table [ON CLUSTER cluster] WHERE expr //从 ClickHouse v22.8 开始，提供了这种轻量级删除的功能  //修改数据： ALTER TABLE [db\_name.]table\_name UPDATE column1 = expr1 [, ...] WHERE filter\_expr |

Mutation操作特点：

（1）Mutation语句是一种“很重”的操作，更适用于批量数据的修改和删除；

（2）其次，它不支持事务，一旦语句被提交执行，就会立刻对现有数据产生影响，无法回滚；

（3）最后，Mutation语句的执行是一个异步的后台过程，语句被提交之后就会立即返回。所以这并不代表具体逻辑已经执行完毕，它的具体执行进度需要通过system.mutations系统表查询。

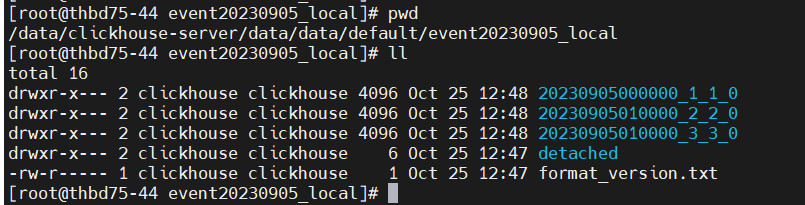
每执行一条ALTER DELETE语句，都会在mutations系统表中生成一条对应的执行计划，当is\_done等于1时表示执行完毕。

与此同时，在数据表的根目录下，会以mutation\_id为名生成与之对应的日志文件用于记录相关信息。

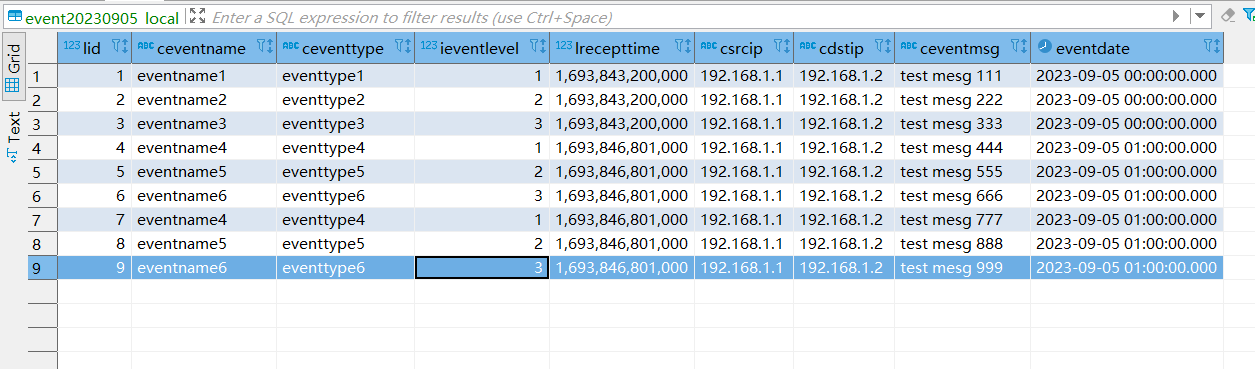
而数据删除的过程是以数据表的每个分区目录为单位，将所有目录重写为新的目录，新目录的命名规则是在原有名称上加上system.mutations.block\_numbers.number。数据在重写的过程中会将需要删除的数据去掉。

旧的数据目录并不会立即删除，而是会被标记成非激活状态（active为0）。等到MergeTree引擎的下一次合并动作触发时，这些非激活目录才会被真正从物理意义上删除。

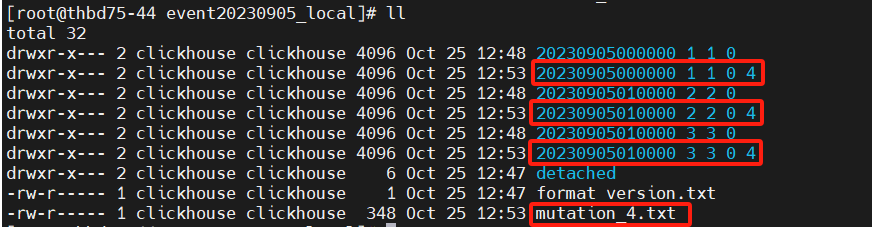
删除之前目录：



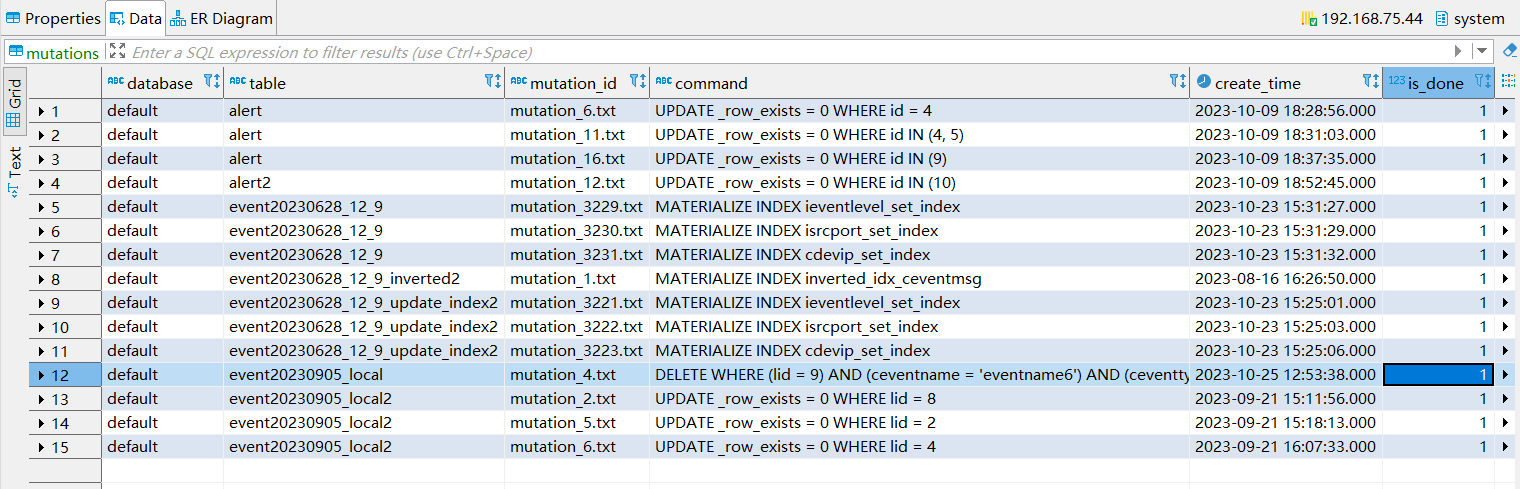
删除一条数据：



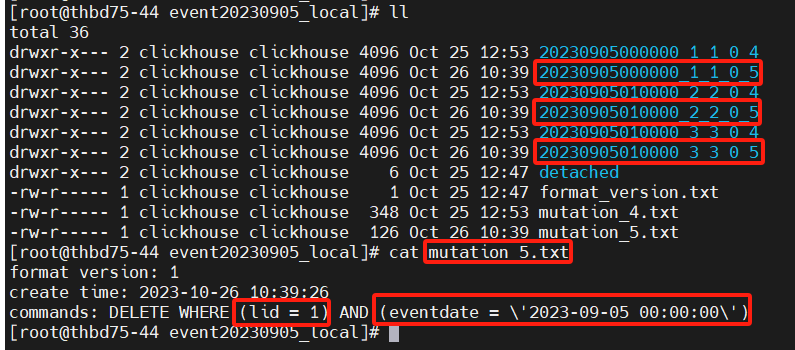
删除之后目录：



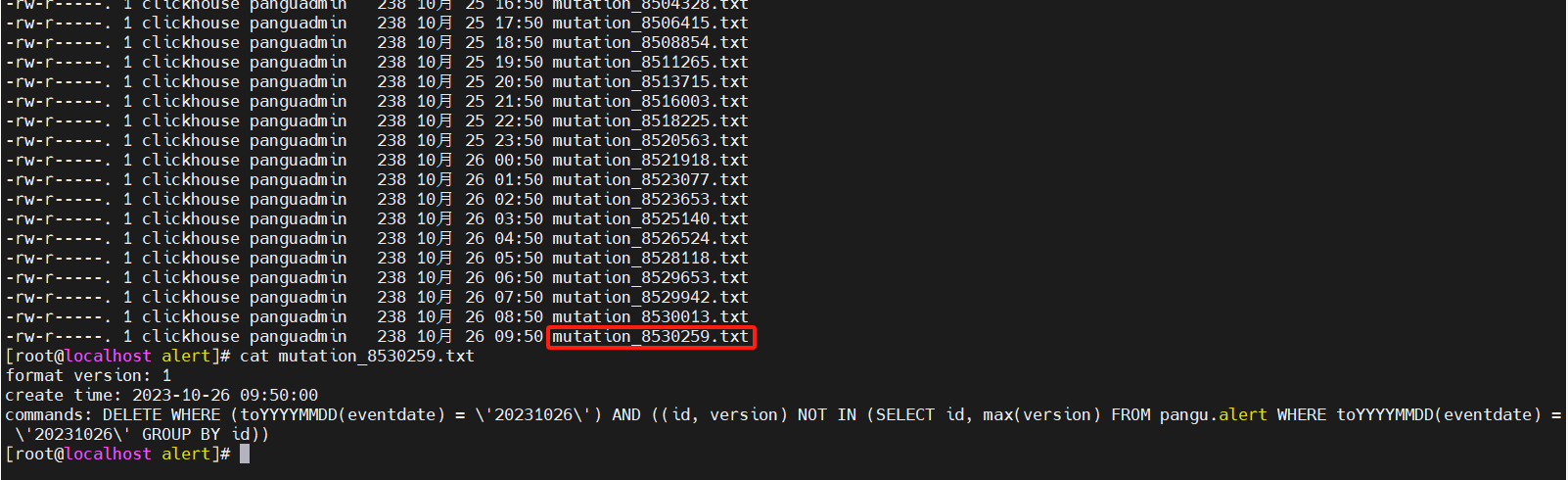
system.mutations表记录：



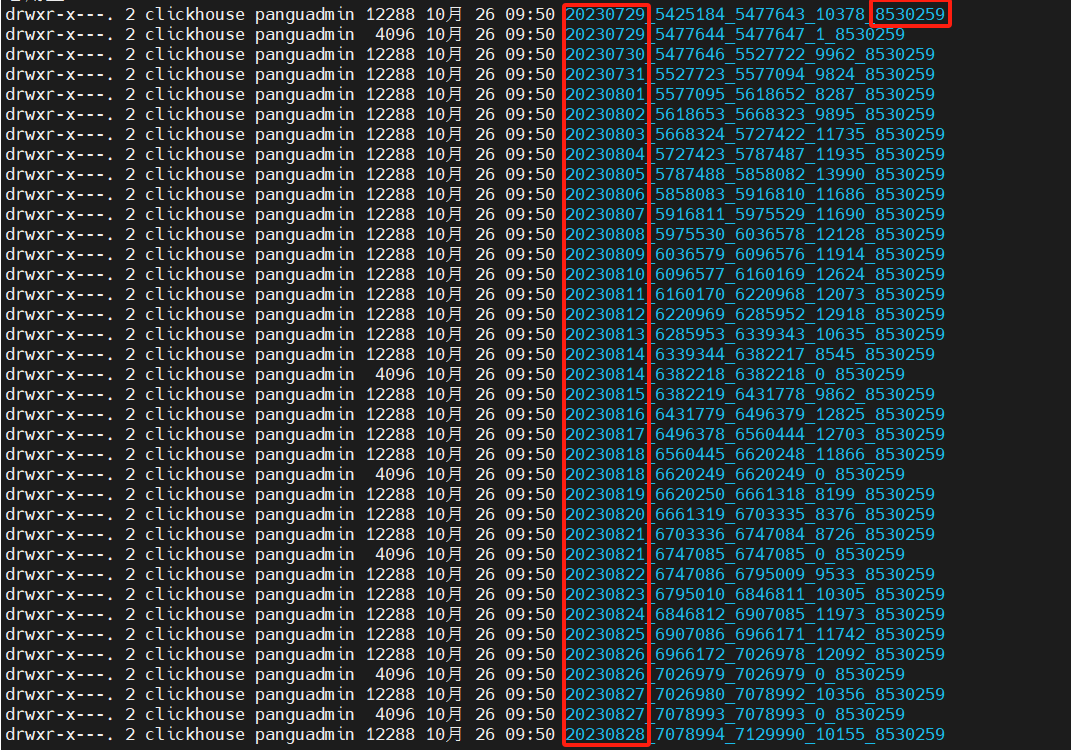
指定分区删除时，不相干的分区part目录也会生成一个新目录：



某生产环境每隔1小时删除一次重复的告警数据：



造成历史数据不相关的分区大量复制，因此，在最新版本中把删除重复告警的频率改为每天执行一次

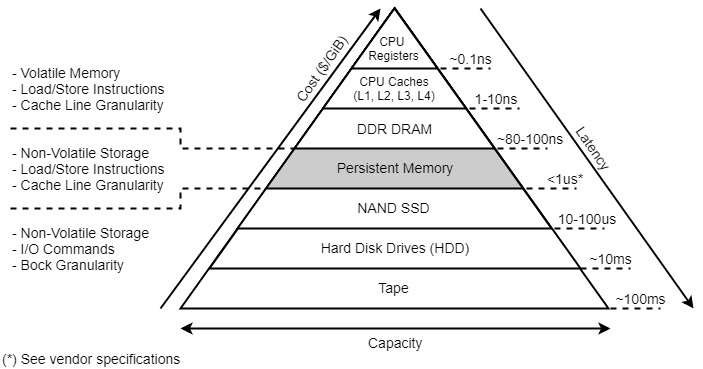


轻量级删除会对需要删除的数据做标记（merge的时候再删除这些已标记的数据），查询时会自动过滤掉做了删除标记的数据，但还是会复制各分区目录。

|  |
| --- |
| 注意：删除和更新操作不能太频繁 |

**五、Clickhouse 优化**

**1.概述**

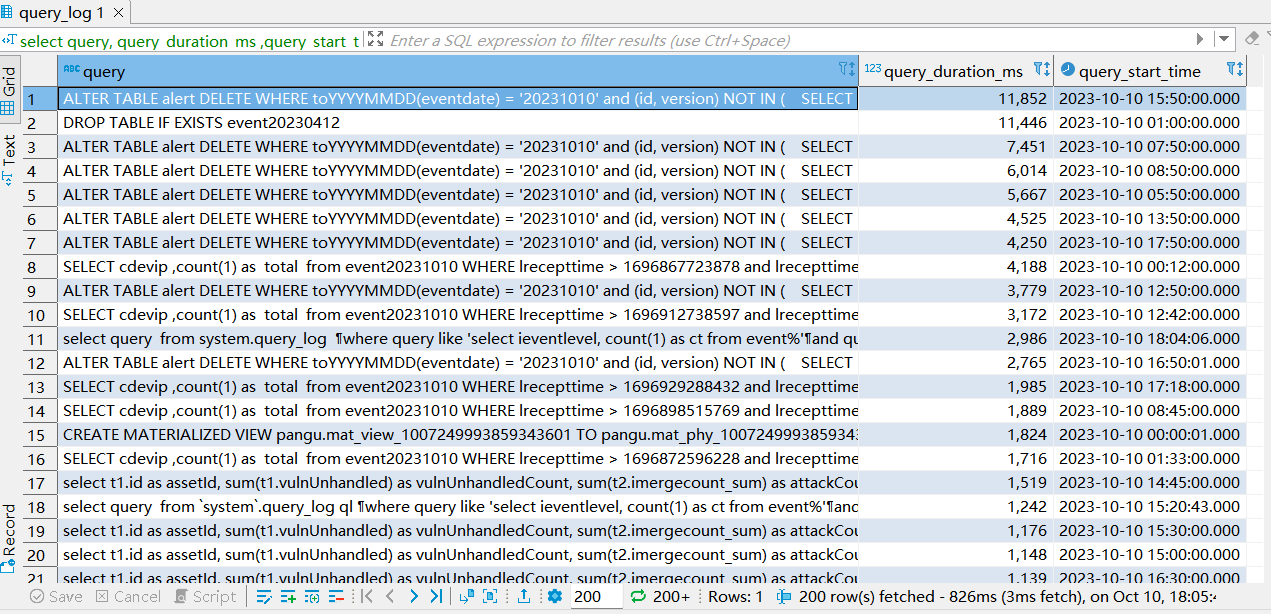


**2.辅助分析工具介绍**

（1）查看system.query\_log表：

|  |
| --- |
| SQL select query, query\_duration\_ms ,query\_start\_time  from system.query\_log  where query like '%event%'  and query not like 'INSERT INTO%'  and query\_start\_time > '2023-10-10 00:00:00' order by query\_duration\_ms desc limit 200;  查询最近sql： select query, query\_duration\_ms ,query\_start\_time  from system.query\_log  where query like '%event%'  and query not like 'INSERT INTO%'  and query\_duration\_ms > 0  and query\_start\_time > '2023-10-10 00:00:00' order by query\_start\_time desc limit 200; |

查询最慢的200条sql：



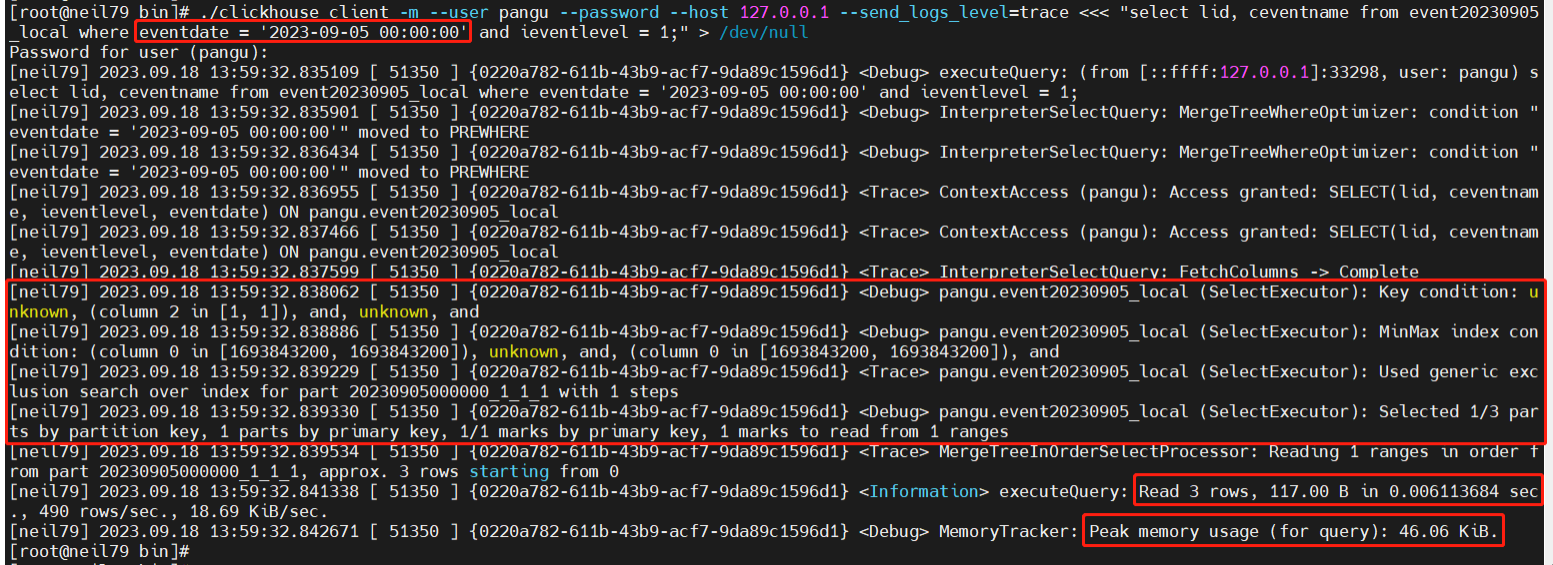
（2）打印出具体执行trace日志

|  |
| --- |
| Shell [root@neil79 bin]# pwd /data/tsoc-csa/clickhouse/usr/bin [root@neil79 bin]# ./clickhouse client -m --user pangu --password --host 127.0.0.1 --send\_logs\_level=trace <<< "select lid, ceventname from event20230905\_local where eventdate = '2023-09-05 00:00:00' and ieventlevel = 1;" > /dev/null Password for user (pangu): 输入数据库密码 |

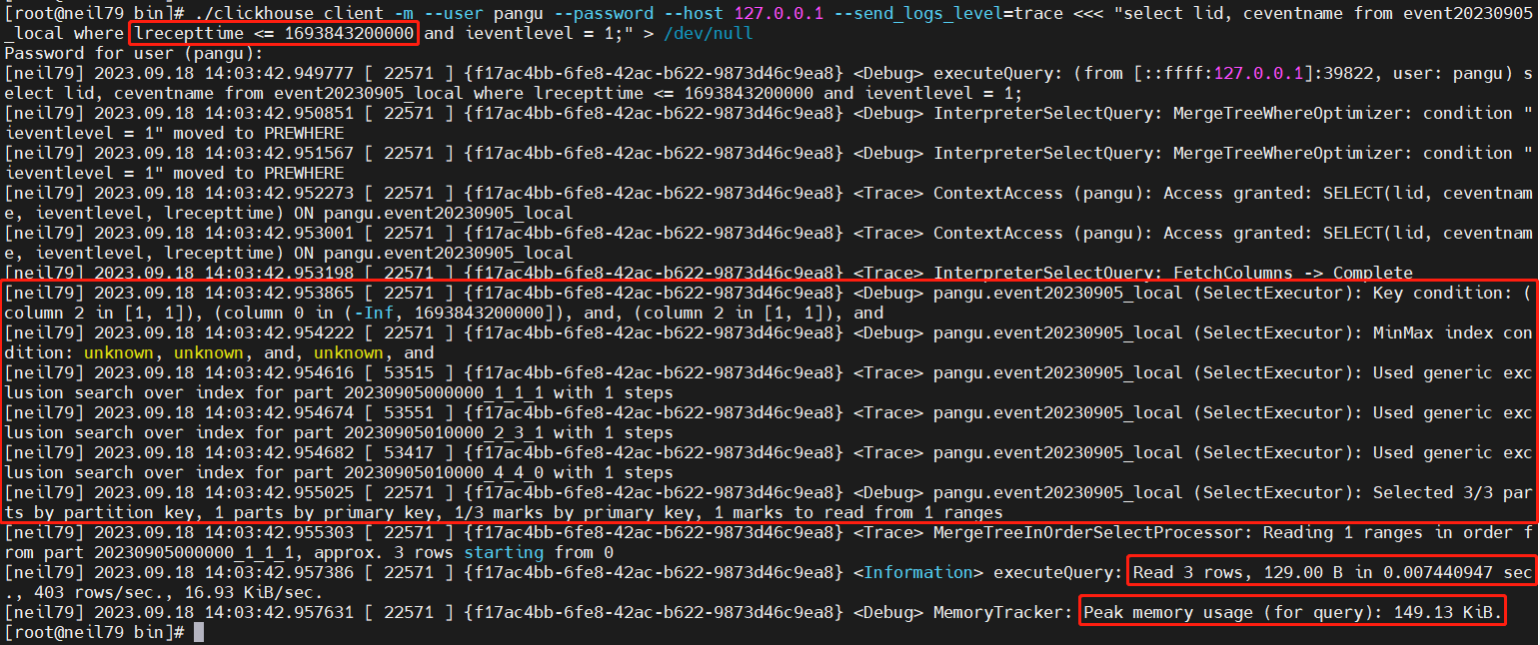
trace日志详解： ORDER BY (lrecepttime, lid, ieventlevel)

|  |
| --- |
| Shell #打印执行查询的一般信息，IP、端口、用户和sql <Debug> executeQuery: (from [::ffff:127.0.0.1]:33298, user: pangu) select lid, ceventname from event20230905\_local where eventdate = '2023-09-05 00:00:00' and ieventlevel = 1; #eventdate查询条件从 WHERE 子句移动到PREWHERE子句中，以减少需要从磁盘读取的数据量 <Debug> InterpreterSelectQuery（解释器）: MergeTreeWhereOptimizer（优化器）: condition "eventdate = '2023-09-05 00:00:00'" moved to PREWHERE #访问控制与授权记录 <Trace> ContextAccess (pangu): Access granted: SELECT(lid, ceventname, ieventlevel, eventdate) ON pangu.event20230905\_local #查询执行过程中，获取列操作已经完成 <Trace> InterpreterSelectQuery: FetchColumns -> Complete #使用主键索引，column 2 in [1, 1]：主键第三列取值是[1,1] <Debug> pangu.event20230905\_local (SelectExecutor): Key condition: unknown, (column 2 in [1, 1]), and, unknown, and #使用分区索引,column 0 in [1693843200, 1693843200]：主键第一列取值[1693843200, 1693843200] <Debug> pangu.event20230905\_local (SelectExecutor): MinMax index condition: (column 0 in [1693843200, 1693843200]), unknown, and, (column 0 in [1693843200, 1693843200]), and #在20230905000000\_1\_1\_1 分区中使用基于索引的通用排除搜索算法来查找符合条件的数据（marks） <Trace> pangu.event20230905\_local (SelectExecutor): Used generic exclusion search over index for part 20230905000000\_1\_1\_1 with 1 steps #part和mark筛选结果 <Debug> pangu.event20230905\_local (SelectExecutor): Selected 1/3 parts by partition key, 1 parts by primary key, 1/1 marks by primary key, 1 marks to read from 1 ranges #在20230905000000\_1\_1\_1的1个range中，从行号0开始，顺序读取3行数据 <Trace> MergeTreeInOrderSelectProcessor: Reading 1 ranges in order from part 20230905000000\_1\_1\_1, approx. 3 rows starting from 0 #读取结果信息：读取3条数据，117.00 B，耗时0.006113684 sec <Information> executeQuery: Read 3 rows, 117.00 B in 0.006113684 sec., 490 rows/sec., 18.69 KiB/sec. #消耗内存最大为46.06 KiB <Debug> MemoryTracker: Peak memory usage (for query): 46.06 KiB. |

查询对比，过滤字段是分区字段时：



过滤字段是一级索引接收时间字段时：



|  |
| --- |
| 小结：如果确定数据落在哪个分区时，可以在查询过滤条件中加上分区过滤条件 |

（3）查询执行计划

前述表结构：ORDER BY (lrecepttime, lid, ieventlevel)

|  |
| --- |
| Shell EXPLAIN indexes = 1 SELECT  lid,  ceventname FROM event20230905\_local WHERE (eventdate = '2023-09-05 00:00:00') AND (ieventlevel = 1)  Query id: d68fcfef-5b50-4457-9b97-99774b85be41  ┌─explain────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐ │ Expression ((Projection + Before ORDER BY)) │ │ Filter (WHERE) │ │ SettingQuotaAndLimits (Set limits and quota after reading from storage) │ │ ReadFromMergeTree │ │ Indexes: │ │ MinMax │ │ Keys: │ │ eventdate │ │ Condition: and((eventdate in [1693843200, 1693843200]), (eventdate in [1693843200, 1693843200])) │ │ Parts: 1/3 │ │ Granules: 1/3 │ │ Partition │ │ Keys: │ │ toYYYYMMDDhhmmss(eventdate) │ │ Condition: and((toYYYYMMDDhhmmss(eventdate) in [20230905000000, 20230905000000]), (toYYYYMMDDhhmmss(eventdate) in [20230905000000, 20230905000000])) │ │ Parts: 1/1 │ │ Granules: 1/1 │ │ PrimaryKey │ │ Keys: │ │ ieventlevel │ │ Condition: (ieventlevel in [1, 1]) │ │ Parts: 1/1 │ │ Granules: 1/1 │ └────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘ |

**3.ClickHouse修改与删除优化**

通过使用ReplacingMergeTree引擎实现修改，通过使用VersionedCollapsingMergeTree 实现修改和删除功能。

[ClickHouse修改与删除优化](https://venusgroup.feishu.cn/docx/YpHCdVgwHo6tbCxK7X2colUmnGd)

**4.物化视图原理与应用**

ClickHouse拥有普通和物化两种视图，其中物化视图拥有独立的存储，而普通视图只是一层简单的查询代理。

[ClickHouse物化视图简介与应用](https://venusgroup.feishu.cn/docx/TbcedGAnyopU2VxPhgfclWE4n0N)

**5.投影原理与应用**

ClickHouse 投影（Projection） 可以看做是一种更加智能的物化视图。

[ClickHouse投影原理与应用](https://venusgroup.feishu.cn/docx/LK8Pdi3ycoZ77exnj3gcIp4Rnjb)

**6.一级索引优化**

一级索引也叫主索引，由表结构中的PRIMARY KEY的字段决定，当PRIMARY KEY缺省时，由ORDER BY字段决定

[ClickHouse一级索引及优化](https://venusgroup.feishu.cn/docx/NGtFdkUvVoSdpCxpxqWcOZEJn6f)

**7.二级索引应用**

除了一级索引之外，MergeTree同样支持二级索引。二级索引又称跳数索引，由数据的聚合信息构建而成。

[ClickHouse二级索引及应用](https://venusgroup.feishu.cn/docx/PljkdQChbova82xB083cuUlWnQf)

**8.字段类型优化**

将表结构中具有类型性质的String类型改为LowCardinality(String) 低基数类型

[表字段类型优化——LowCardinality（低基数）](https://venusgroup.feishu.cn/docx/FUaEd2xKZo7ZNdx80vEctA86ntb)

**9.SQL优化案例**

[ClickHouse SQL 优化](https://venusgroup.feishu.cn/docx/MdFed1HX4oYQcTxuRQHcJhxSnLd)

**10.硬件提升**

[云平台上部署的cdn环境cpu st（steal）值高导致写入慢的问题](https://venusgroup.feishu.cn/docx/IkshdpTV8oHtyvxlLkrcAlvVnWd)