1. 本次课程介绍

1.1. 本次系列课程介绍

OLAP 之 ClickHouse 和 Doris 谁与争锋？ClickHouse 和 Doris 深度大 PK？

- 首次完整揭秘 ClickHouse 核心特性，知其然，知其所以然
- 彻底揭秘千亿级企业 ClickHouse 实时处理引擎架构设计、核心技术设计、运行机理全流程;
- 彻底揭秘千亿级企业 ClickHouse 在企业大数据业务场景下的应用实践;
- Doris 源码核心作者揭秘 Doris 架构设计核心原理;
- 首次全方位深度对比 ClickHouse 和 Doris 两大 OLAP 利器。

1.2. 今日课程大纲

今天主要的内容，是跟大家交付，关于 ClickHouse 如何做查询分析那么快的原因原理分析。咱们先从探讨，一款高效的 OLAP 系统组件的核心技术应该有哪些？然后 ClickHouse 实现了那些？最终的工作流程是怎样的？

- ClickHouse 全知全解
- ClickHouse 设计思路和核心特性剖析
- ClickHouse 表引擎详解
- ClickHouse 工作原理（数据分区，一级索引，二级索引，数据压缩，数据标记，数据查询）

2. ClickHouse 表引擎详解和架构原理

2.1. ClickHouse 设计思想和核心技术特征

2.1.1. ClickHouse 全知全解

ClickHouse 是一个用于联机分析 (OLAP) 的列式数据库管理系统 (DBMS)。来自于 2011 年在纳斯达克上市的俄罗斯本土搜索引擎企业 Yandex 公司，诞生之初就是为了服务 Yandex 公司自家的 Web 流量分析产品 Yandex.Metrica，后来经过演变，逐渐形成为现在的 ClickHouse，全称是：Click Stream, Data WareHouse

ClickHouse 官网：https://clickhouse.tech/，它具有 ROLAP、在线实时查询、完整的 DBMS 功能支持、列式存储、不需要任何数据预处理、支持批量更新、拥有非常完善的 SQL 支持和函数、支持高可用、不依赖 Hadoop 复杂生态、开箱即用等许多特点。

在 1 亿数据集体量的情况下，ClickHouse 的平均响应速度是 Vertica 的 2.63 倍、InfiniDB 的 17 倍、MonetDB 的 27 倍、Hive 的 126 倍、MySQL 的 429 倍以及Greenplum 的 10 倍。详细的测试结果可以查阅：https://clickhouse.tech/benchmark/dbms/。

ClickHouse 非常适用于商业智能领域（也就是我们所说的 BI 领域），除此之外，它也能够被广泛应用于广告流量、Web、App 流量、电信、金融、电子商务、信息安全、网络游戏、物联网等众多其他领域。

ClickHouse 是近年来备受关注的开源列式数据库，主要用于数据分析（OLAP）领域。目前国内社区火热，各个大厂纷纷跟进大规模使用：

- 今日头条内部用 ClickHouse 来做用户行为分析，内部一共几千个 ClickHouse 节点，单集群最大 1200 节点，总数据量几十 PB，日增原始数据 300TB 左右。
- 腾讯内部用 ClickHouse 做游戏数据分析，并且为之建立了一整套监控运维体系。
- 携程内部从 18 年 7 月份开始接入试用，目前 80% 的业务都跑在 ClickHouse 上。每天数据增量十多亿，近百万次查询请求。
- 快手内部也在使用 ClickHouse，存储总量大约 10PB， 每天新增 200TB， 90% 查询小于 3S。

ClickHouse 缺点：

```
1、没有完整的事务支持
2、稀疏索引导致 ClickHouse 不擅长细粒度或者 key-value 类型数据的查询需求
3、缺少高频率，低延迟的修改或删除已存在数据的能力。仅能用于批量删除或修改数据
4、不擅长 join 操作
```

2.1.2. ClickHouse 设计思路剖析

关于 OLTP 系统和 OLAP 系统的核心设计思想和技术路线有哪些呢？我们做一个完整的探讨。

数据存储系统的关于查询的典型操作：

```
-- 第一种需求： 根据 key 找 value
select name, age from student where id = 1;

select name, age from student where age > 30;

-- 第二种需求： 根据 department 统计平均年龄
select department, avg(age) from student group by department;
```

如果：第一种需求多

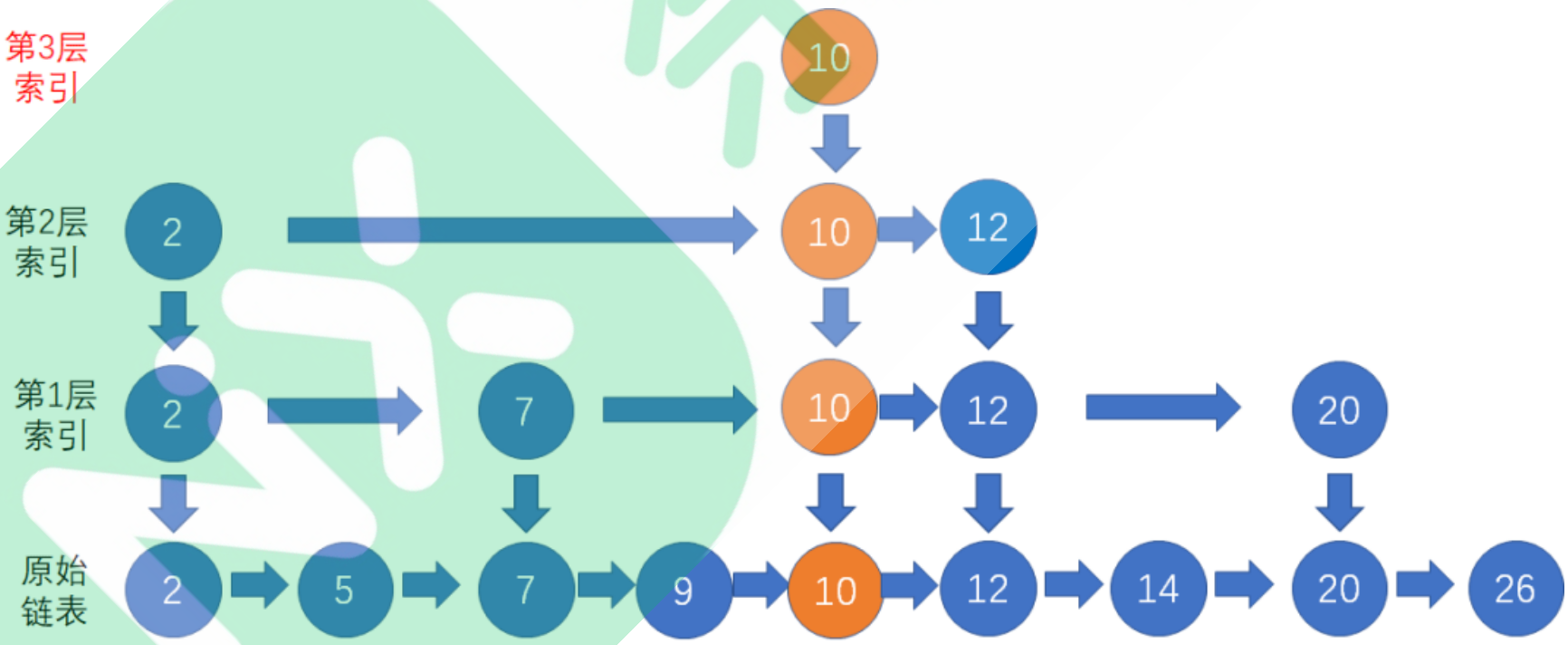
```
如果数据量小，并且数据是结构化的，使用 MySQL 去存储即可
如果数据量大，不管是不是结构化的，可以转成 key-value 的存储，使用 HBase,Cassandra 等来解决
```

如果：第二种需求多：

```
如果数据量小，并且数据是结构化的，使用 MySQL 去存储即可
如果数据量大，不管是不是结构化的，设计一个专门用来做分析的存储计算引擎解决分析的低效率问题
```

关于如何设计一个 HBase 存储系统，我们探讨一下它的核心设计思想：

```
01、内存 + 磁盘：保证处理效率，也保证数据安全
02、内存：必须经过设计，内存具备优秀的数据结构，保证基本的读写高效，甚至为了不同的需求，可以让读写效率倾斜。
03、磁盘：数据必须存放在磁盘，保证数据安全。磁盘数据文件必须经过精心设计，保证扫描磁盘数据文件的高效率
04、数据排序：在海量数据中要想保证低延时的随机读写操作，数据最好是排序的
05、范围分区：当数据排序之后，可以进行范围分区，来平摊负载，让多台服务器联合起来对外提供服务
06、跳表：基于数据排序+范围分区构建索引表，形成跳表的拓扑结构，方便用户操作时快速定位数据分区的位置
07、LSM-Tree存储引擎：把随机写变成顺序追加，在通过定期合并的方式来合并数据，去除无效数据，从而实现数据的删除和修改。
```



海量数据中，如果进行高效率的查询的核心思想：设计一种架构，能够快速把待搜寻的数据范围降低到原来的 1/n，然后再结合索引或者热点数据放在内存等思路，就能实现高效率的查询了。

那么一个专门用来做 OLAP 分析的存储引擎该如何设计呢？如何在海量数据中，针对大量数据进行查询分析呢？一些常见的方案和手段如下：

```
01、列式存储 + 字段类型统一
02、列裁剪
03、数据排序
04、数据分区分片 + 分布式查询
05、预聚合
06、利用CPU特性：向量化引擎，操作系统必须支持
07、主键索引+二级索引+位图索引+布隆索引等
08、支持近似计算 pv
09、定制引擎：多样化的存储引擎满足不同场景的特定需要
10、多样化算法选择：Volnitsky高效字符串搜索算法 和 HyperLogLog基于概率高效去重算法
```

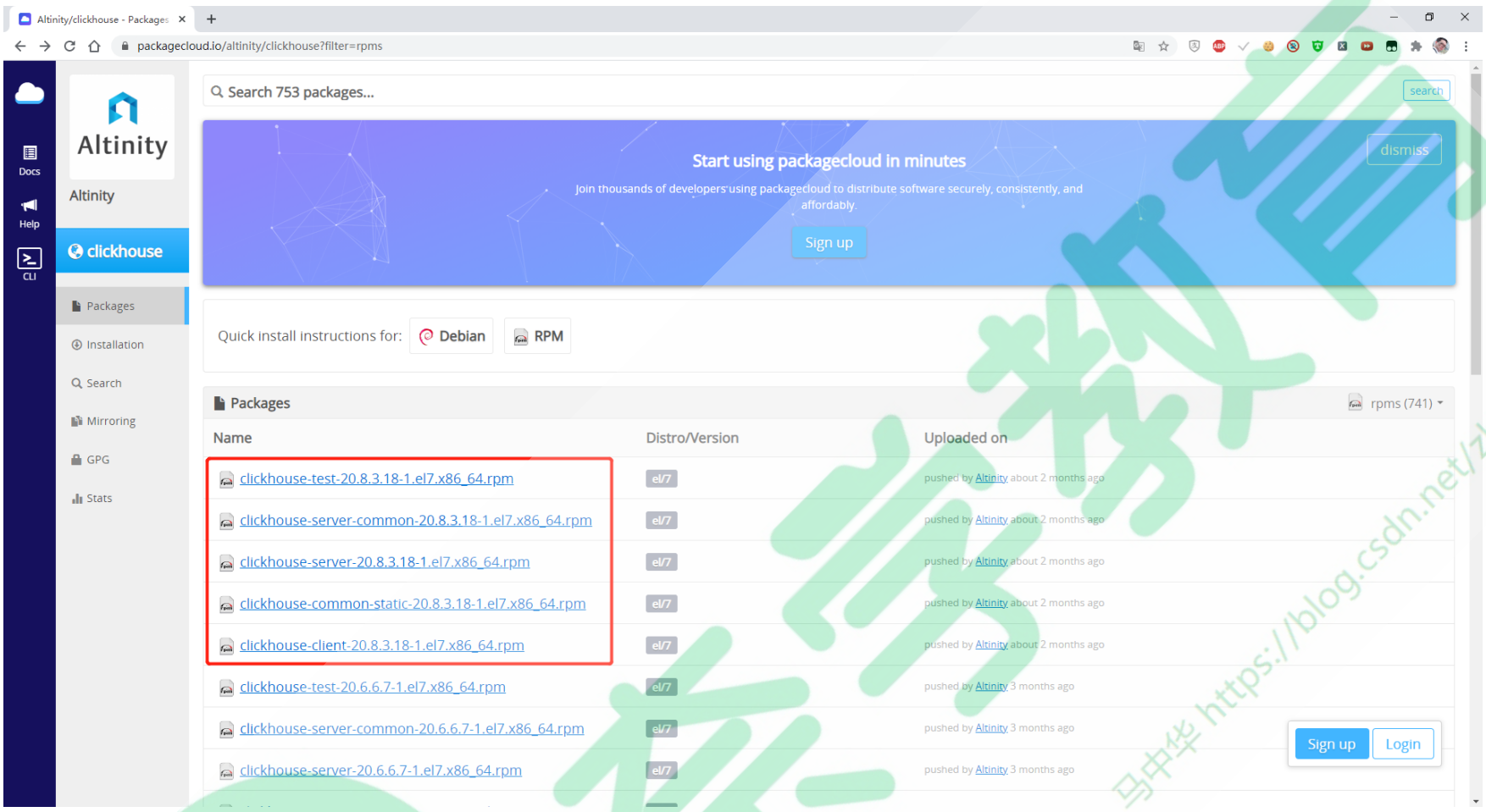
总结一下：单条记录的增删改等操作，通过数据的横向划分，做到数据操作的快速定位，在海量数据查询分析中，一般就是针对某些列做分析，既然并不是全部列，那么把数据做纵向切分把表中的数据按照列来单独存储，那么在做分析的时候，同样可以快速把待查询分析的数据总量降低到原来表的1/n，同样提高效率。

提到预聚合，大家会想到 Kylin，但是 Kylin 也有它的缺点：

1、预聚合只支持固定的分析场景，无法满足自定义分析场景，所以预聚合只能作为一种可选方案
2、维度组合爆炸会导致数据膨胀，这样会造成不必要的计算和存储开销。无必要的维度组合的计算就属于浪费资源
3、大概率数据都是增量生成，预聚合不能进行数据更新。所以会产生大量的重算。

2.1.3. ClickHouse 安装部署

ClickHouse 可以通过源码编译、预编译压缩包、Docker 镜像和 RPM 等多种方法进行安装，在此讲解 RPM 方式的安装，先从这儿下载 相关的 RPM 包：
https://packagecloud.io/altinity/clickhouse



2.2. ClickHouse 表引擎详解

2.2.1. ClickHouse 表引擎介绍

表引擎在 ClickHouse 中的作用十分关键，直接决定了数据如何存储和读取、是否支持并发读写、是否支持 index、支持的 query 种类、是否支持主备复制等。

1、数据的存储方式和位置，写到哪里以及从哪里读取数据
2、支持哪些查询以及如何支持。
3、并发数据访问。
4、索引的使用（如果存在）。
5、是否可以执行多线程请求。
6、数据复制参数。

具体可看官网：https://clickhouse.tech/docs/zh/engines/table-engines/

关于 ClickHouse 的底层引擎，其实可以分为 数据库引擎 和 表引擎 两种。在此，我们重点讲解 表引擎。

关于库引擎，简单总结一下：ClickHouse 也支持在创建库的时候，指定库引擎，目前支持5种，分别是：Ordinary，Dictionary， Memory， Lazy，MySQL，其实 Ordinary 是默认库引擎，在此类型库引擎下，可以使用任意类型的表引擎。

1、Ordinary引擎：默认引擎，如果不指定数据库引擎创建的就是 Ordinary 数据库
2、Dictionary引擎：此数据库会自动为所有数据字典创建表
3、Memory引擎：所有数据只会保存在内存中，服务重启数据消失，该数据库引擎只能够创建 Memory 引擎表
4、MySQL引擎：改引擎会自动拉取远端 MySQL 中的数据，并在该库下创建 MySQL 表引擎的数据表
5、Lazy延时引擎：在距最近一次访问间隔 expiration_time_in_seconds 时间段内，将表保存在内存中，仅适用于 Log 引擎表

ClickHouse 的表引擎提供了四个系列（Log、MergeTree、Integration、Special）大约 28 种表引擎，各有各的用途。比如 Log 系列用来做小表数据分析，MergeTree 系列用来做大数据量分析，而 Integration 系列则多用于外表数据集成。Log、Special、Integration 系列的表引擎相对来说，应用场景有限，功能简单，应用特殊用途，MergeTree 系列表引擎又和两种特殊表引擎（Replicated，Distributed）正交形成多种具备不同功能的 MergeTree 表引擎。

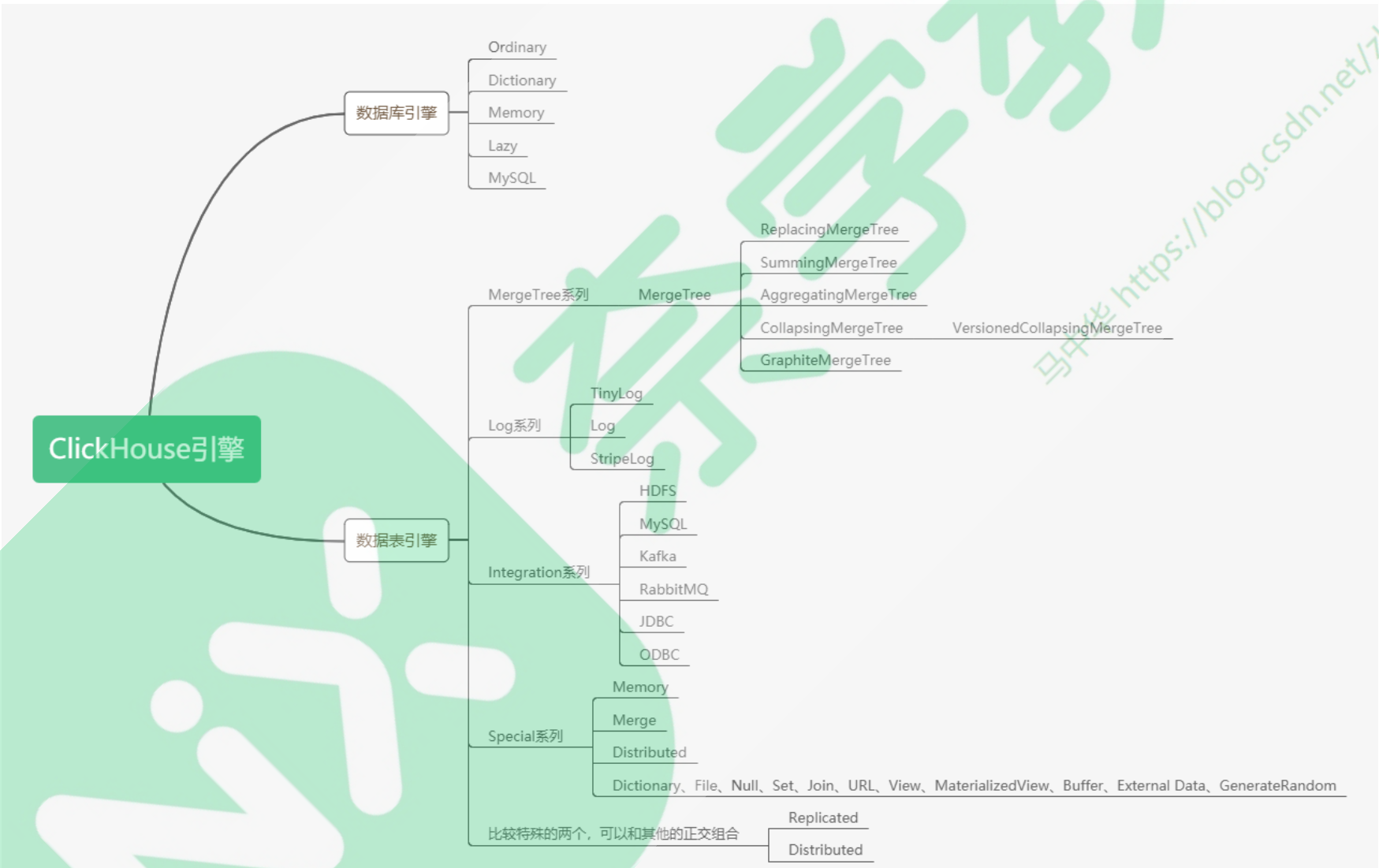
这是 ClickHouse 的表引擎系列家谱：

MergeTree Family ^
- MergeTree
- Data Replication
- Custom Partitioning Key
- ReplacingMergeTree
- SummingMergeTree
- AggregatingMergeTree
- CollapsingMergeTree
- VersionedCollapsingMerge...
- GraphiteMergeTree

Log Family ^
- Introduction
- StripeLog
- Log
- TinyLog

Integrations ^
- Kafka
- MySQL
- JDBC
- ODBC
- HDFS

Special ^
- Distributed
- External data
- Dictionary
- Merge
- File
- Null
- Set
- Join
- URL
- View
- MaterializedView
- Memory
- Buffer



MergeTree 作为家族中最基础的表引擎，提供了主键索引、数据分区、数据副本和数据采样等基本能力，而家族中其他的表引擎则在 MergeTree 的基础之上各有所长：

项目	类别	基础
Replicated 支持数据副本	Replacing Summing Aggregating Collapsing VersionedCollapsing Graghite	MergeTree 基础表引擎

2.2.2. MergeTree 引擎工作机制详解

MergeTree 系列是官方主推的存储引擎，支持几乎所有 ClickHouse 核心功能，该系列中，常用的表引擎有：MergeTree、ReplacingMergeTree、CollapsingMergeTree、VersionedCollapsingMergeTree、SummingMergeTree、AggregatingMergeTree 等。学习好 MergeTree 表引擎的工作机制，是应用好 ClickHouse 的最基本基础。

关于表引擎类型：

第一：**MergeTree** 表引擎主要用于海量数据分析，支持数据分区、存储有序、主键索引、稀疏索引、数据 TTL 等。MergeTree 支持所有 ClickHouse SQL 语法，但是有些功能与 MySQL 并不一致，比如在 MergeTree 中主键并不用于去重。

第二：为了解决 MergeTree 相同主键无法去重的问题，ClickHouse 提供了 ReplacingMergeTree 引擎，用来做去重。**ReplacingMergeTree** 确保数据最终被去重，但是无法保证查询过程中主键不重复。因为相同主键的数据可能被 shard 到不同的节点，但是 compaction 只能在一个节点中进行，而且 optimize 的时机也不确定。

第三：**CollapsingMergeTree** 引擎要求在建表语句中指定一个标记列 Sign（插入的时候指定为 1，删除的时候指定为 -1），后台 Compaction 时会将主键相同、Sign 相反的行进行折叠，也即删除。来消除 ReplacingMergeTree 的限制。

第四：为了解决 CollapsingMergeTree 乱序写入情况下无法正常折叠问题，**VersionedCollapsingMergeTree** 表引擎在建表语句中新增了一列 Version，用于在乱序情况下记录状态行与取消行的对应关系。主键相同，且 Version 相同、Sign 相反的行，在 Compaction 时会被删除。

第五：ClickHouse 通过 **SummingMergeTree** 来支持对主键列进行预先聚合。在后台 Compaction 时，会将主键相同的多行进行 sum 求和，然后使用一行数据取而代之，从而大幅度降低存储空间占用，提升聚合计算性能。

第六：**AggregatingMergeTree** 也是预先聚合引擎的一种，用于提升聚合计算的性能。与 SummingMergeTree 的区别在于：SummingMergeTree 对非主键列进行 sum 聚合，而 AggregatingMergeTree 则可以指定各种聚合函数。

MergeTree 的建表语法：

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name (
    name1 [type] [DEFAULT|MATERIALIZED|ALIAS expr],
    name2 [type] [DEFAUErEMAMLERLALLIZED|ALIAS expr],
    省略...
) ENGINE = MergeTree()
    [PARTITION BY expr]
    [ORDER BY expr]
    [PRIMARY KEY expr]
    [SAMPLE BY expr]
    [SETTINGS name=value, 省略...]
```

介绍一下其中的几个关键选项：

```
1、PARTITION BY：分区键。指定表数据以何种标准进行分区。分区键既可以是单个列字段，也可以通过元组的形式使用多个列字段，同时它也支持使用列表达式。
2、ORDER BY：排序键，用于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。
3、PRIMARY KEY：主键。声明后会依照主键字段生成一级索引。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。
4、SETTINGS：index_granularity 选项表示索引的粒度，默认值为 8192。MergeTree 索引在默认情况下，每间隔 8192 行数据才生成一条索引。
5、SAMPLE BY：抽样表达式，用于声明数据以何种标准进行采样。
```

注意 settings 中的重要参数：

```
1、index_granularity 默认是 8192
2、index_granularity_bytes 默认 10M，需要通过 enable_mixed_granularity_parts 来开启
```

2.3. ClickHouse 工作原理

MergeTree 表引擎的内部工作细节！最终就是告诉你：为什么 clickhouse 做查询分析，那么快?

ClickHouse 从 OLAP 场景需求出发，定制开发了一套全新的高效列式存储引擎，并且实现了数据有序存储、主键索引、稀疏索引、数据 Sharding、数据 Partitioning、TTL、主备复制等丰富功能。这些功能共同为 ClickHouse 极速的分析性能奠定了基础。

2.3.1. 数据分区

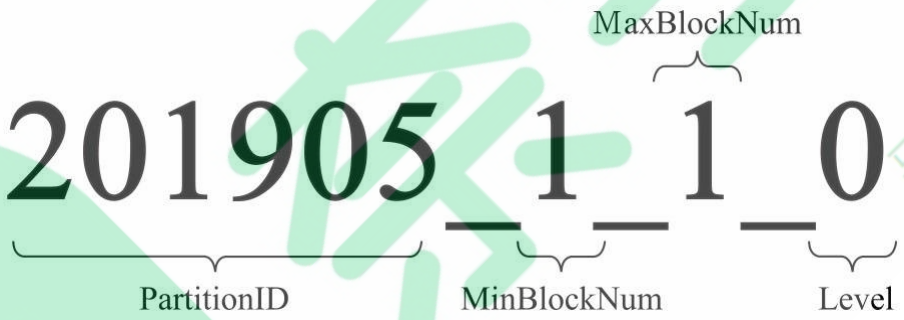
关于表分区目录结构：MergeTree 表的分区目录物理结构：

```
[root@bigdata05 20190710_20190711_1_5_1]# pwd
/var/lib/clickhouse/data/nxdb1/nx_smt_table/20190710_20190711_1_5_1
[root@bigdata05 20190710_20190711_1_5_1]# ll
total 48
-rw-r----- 1 root root  34 Aug 14 21:22 a.bin
-rw-r----- 1 root root  48 Aug 14 21:22 a.mrk2
-rw-r----- 1 root root  34 Aug 14 21:22 b.bin
-rw-r----- 1 root root  48 Aug 14 21:22 b.mrk2
-rw-r----- 1 root root 300 Aug 14 21:22 checksums.txt
-rw-r----- 1 root root  85 Aug 14 21:22 columns.txt
-rw-r----- 1 root root  34 Aug 14 21:22 date.bin
-rw-r----- 1 root root  48 Aug 14 21:22 date.mrk2
-rw-r----- 1 root root  10 Aug 14 21:22 default_compression_codec.txt
-rw-r----- 1 root root  34 Aug 14 21:22 name.bin
-rw-r----- 1 root root  48 Aug 14 21:22 name.mrk2
-rw-r----- 1 root root   8 Aug 14 21:22 primary.idx
[root@bigdata05 20190710_20190711_1_5_1]#
```

关于这些文件的解释：

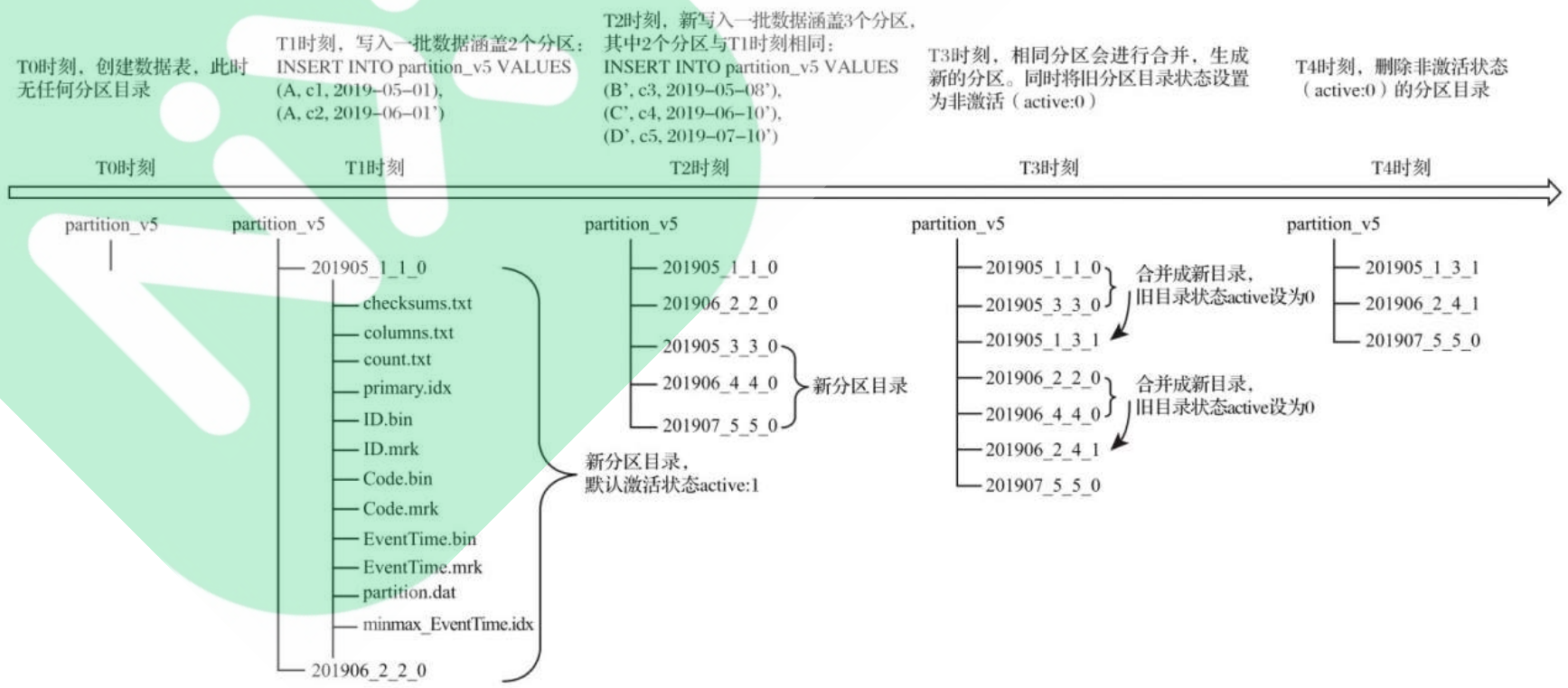
1、分区目录：20190710_20190711_1_5_1，一个分区可能会有多个不同的目录，该目录下存储该分区的数据及其他各种形式的数据。后台会执行合并，把相同分区的多个目录合并到一个分区。
2、checksums.txt：校验文件。使用二进制格式存储。它保存了余下各类文件(primary.idx、count.txt等)的 size 大小及 size 的哈希值，用于快速校验文件的完整性和正确性。
3、columns.txt：列信息文件，使用明文格式存储。用于保存此数据分区下的列字段信息
4、count.txt：计数文件，使用明文格式存储。用于记录当前数据分区目录下数据的总行数
5、primary.idx：一级索引文件，主键索引文件
6、xxx.bin：数据文件，使用压缩格式存储，默认为 LZ4 压缩格式，用于存储某一列的数据，每一列都对应一个该文件，如列 date 为 date.bin
7、xxx.mrk2：列字段标记文件，如果使用了自适应大小的索引间隔，则标记文件以 .mrk2 命名，否则以 .mrk 命名。它建立 primary.idx 稀疏索引与 xxx.bin 数据文件之间的映射关系，先通过主键索引找到数据的偏移量，然后去 xxx.bin 数据文件中找到真实数据
8、... 还有二级索引 和 分区键相关信息文件等等

关于表分区命名规则：分区的命名规则：PartitionID_MinBlockNum_MaxBlockNum_Level

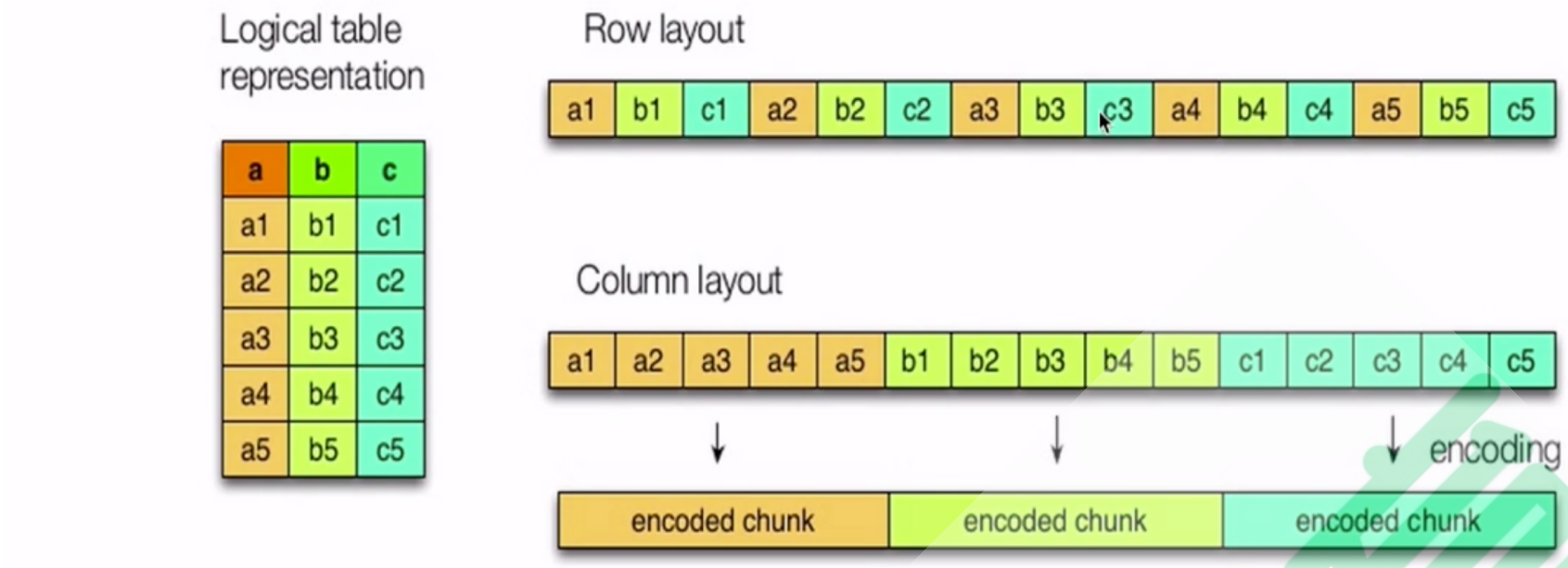


该 blocknum 在该表内全局累加，每次创建一个新的分区目录的时候，就会累加 1。Level 是分区被合并过的次数计数，合并一次则加1。

关于分区的合并规则：



2.3.2. 列式存储

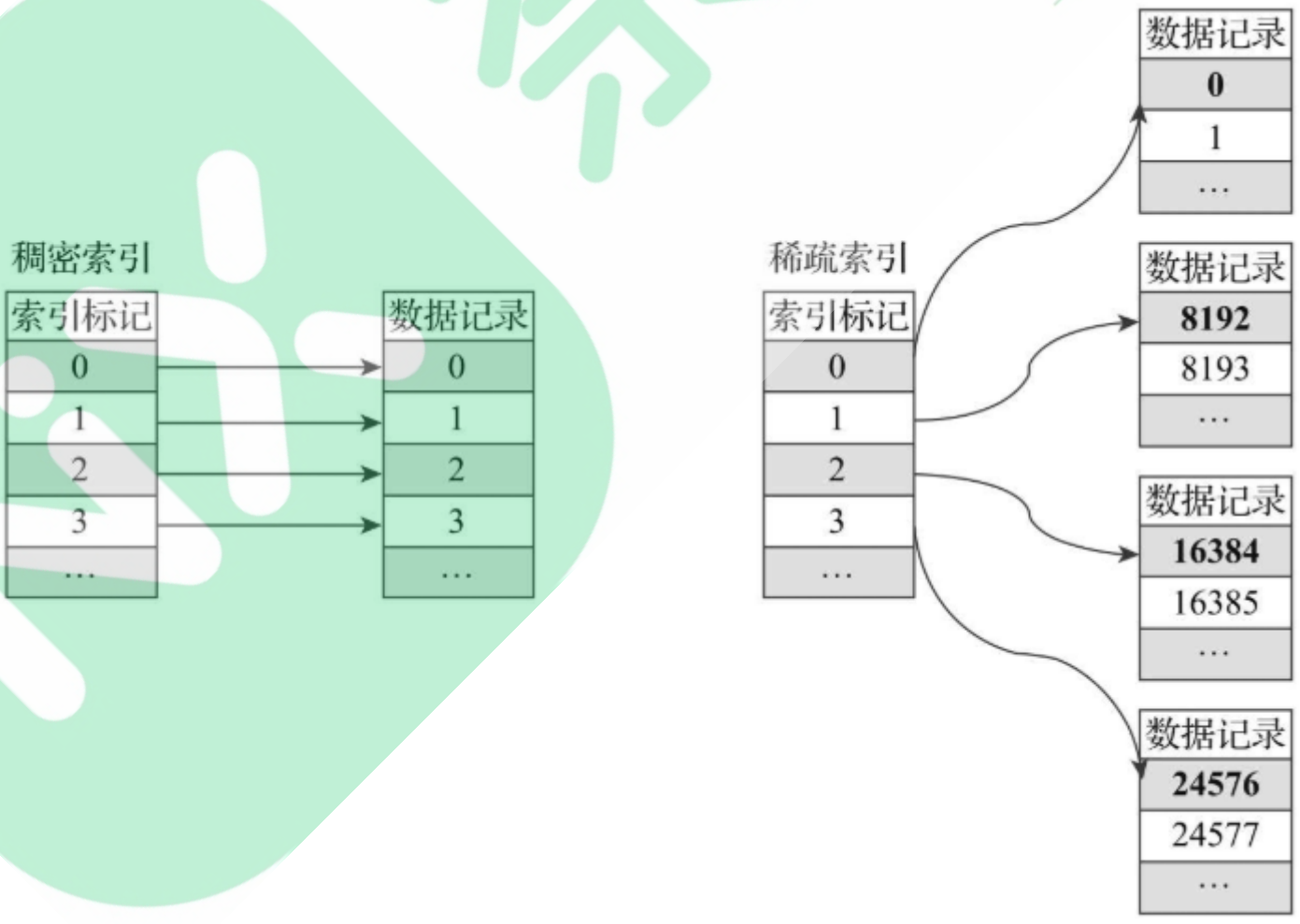


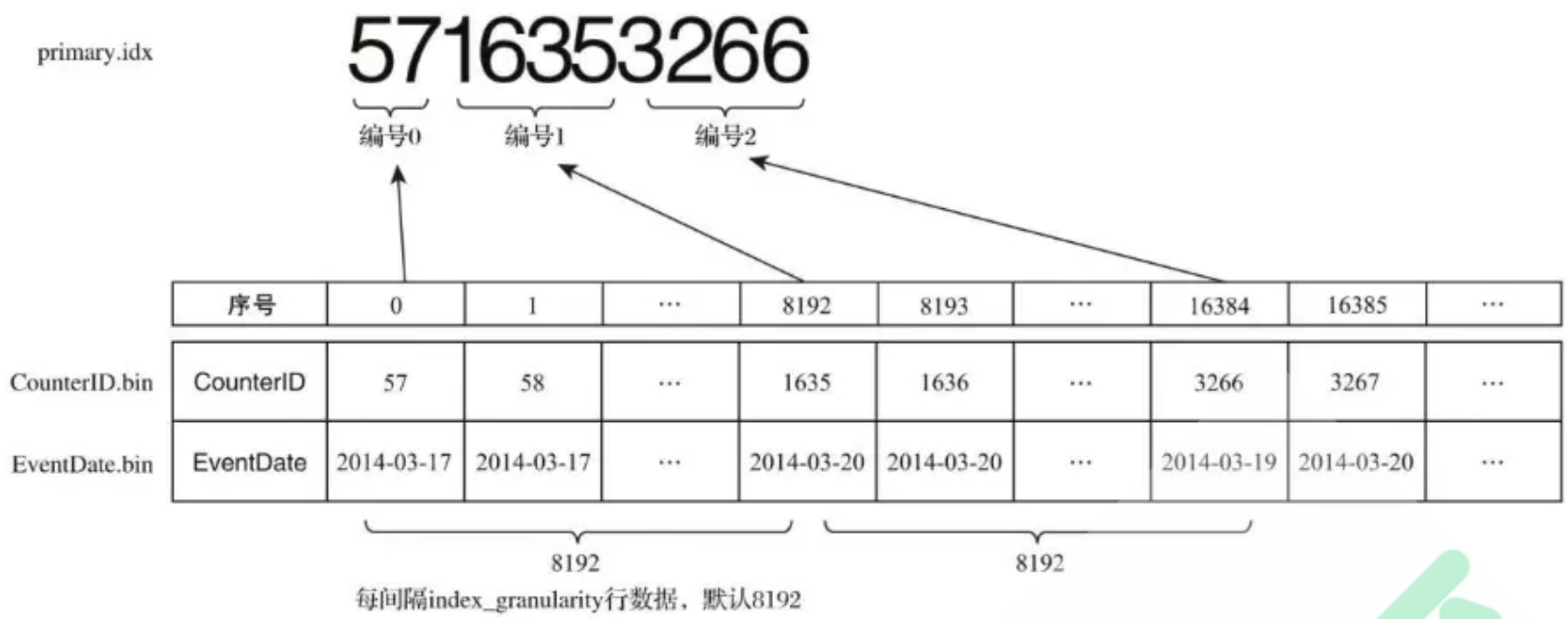
相比于行式存储，列式存储在分析场景下有着许多优良的特性。

- **分析场景中往往需要读大量行但是少数几个列。**在行存模式下，数据按行连续存储，所有列的数据都存储在一个block中，不参与计算的列在IO时也要全部读出，读取操作被严重放大。而列存模式下，只需要读取参与计算的列即可，极大的减低了IO cost，加速了查询。
- 同一列中的数据属于同一类型，**压缩效果显著。**列存往往有着高达十倍甚至更高的压缩比，**节省了大量的存储空间，降低了存储成本。**
- **更高的压缩比意味着更小的data size**，从磁盘中读取相应数据耗时更短。
- **自由的压缩算法选择。**不同列的数据具有不同的数据类型，适用的压缩算法也就不尽相同。可以针对不同列类型，选择最合适的压缩算法。
- **高压缩比**，意味着同等大小的内存能够存放更多数据，系统cache效果更好。

2.3.3. 一级索引

关于一级索引：MergeTree 的主键使用 PRIMARY KEY 定义，待主键定义之后，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 文件内。一级索引是稀疏索引，意思就是说：每一段数据生成一条索引记录，而不是每一条数据都生成索引，如果是每一条数据都生成索引，则是稠密索引。稀疏索引的好处，就是少量的索引标记，就能记录大量的数据区间位置信息，比如不到 24414 条标记信息，就能为 2E 条数据提供索引（算法：200000000 / 8192）。在 ClickHouse 中，一级索引常驻内存。总的来说：一级索引和标记文件一一对齐，两个索引标记之间的数据，就是一个数据区间，在数据文件中，这个数据区间的所有数据，生成一个压缩数据块。





需要注意的是：ClickHouse 的主键索引与 MySQL 等数据库不同，它并不用于去重，即便 primary key 相同的行，也可以同时存在于数据库中。要想实现去重效果，需要结合具体的表引擎 ReplacingMergeTree、CollapsingMergeTree、VersionedCollapsingMergeTree 实现。这个在之前的表引擎介绍中讲过。

2.3.4. 二级索引

关于二级索引：又称之为跳数索引。目的和一级索引一样，是为了减少待搜寻的数据的范围。跳数索引的默认是关闭的，需要通过 `SET allow_experimental_data_skipping_indices = 1` 来开启，索引生成粒度由 granularity 控制，如果生成了二级索引，则会在分区目录下生成额外的：skp_idx_[Column].idx 与 skp_idx_[Column].mrk 文件。跳数索引的生成规则：按照特定规则每隔 granularity 个 index_granularity 条数据，就会生成一条跳数索引。比如 minmax 跳数索引，生成的是：granularity 个 index_granularity 条数据内的最大值最小值生成一条索引，如果将来需要针对构建二级索引的这个字段求最大值最小值，则可以帮助提高效率。跳数索引一共支持四种类型：minmax（最大最小）、set（去重集合）、ngrambf_v1（ngram分词布隆索引） 和 tokenbf_v1（标点符号分词布隆索引），一张数据表支持同时声明多个跳数索引。比如：

```
GRANULARITY = 你在创建二级索引索引的指定的
INDEX_GRANULARITY = 8192
GRANULARITY * INDEX_GRANULARITY
```

```
CREATE TABLE skip_test(
    ID String,
    URL String,
    Code String,
    EventTime Date,
    INDEX a ID TYPE minmax GRANULARITY 5,
    INDEX b (length(ID) * 8) TYPE set(2) GRANULARITY 5,
    INDEX c (ID, Code) TYPE ngrambf_v1(3, 256, 2, 0) GRANULARITY 5,
    INDEX d ID TYPE tokenbf_v1(256, 2, 0) GRANULARITY 5
) ENGINE= MergeTree()
    order by id;
```

关于跳数索引支持的多种类型的区别：

```
1、minmax：以 index_granularity 为单位，存储指定表达式计算后的 min、max 值；在等值和范围查询中能够帮助快速跳过不满足要求的块，减少IO。

2、set(max_rows)：以index granularity为单位，存储指定表达式的 distinct value 集合，用于快速判断等值查询是否命中该块，减少IO。

3、ngrambf_v1(n, size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)：将 string 进行 ngram 分词后，构建 bloom filter，能够优化 等值、like、in 等查询条件。

4、tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)：与 ngrambf_v1 类似，区别是不使用 ngram 进行分词，而是通过标点符号进行词语分割。

5、bloom_filter([false_positive])：对指定列构建 bloom filter，用于加速 等值、like、in 等查询条件的执行。
```

2.3.5. 数据压缩

关于数据压缩：ClickHouse 的数据存储文件 column.bin 中存储是一列的数据，由于一列是相同类型的数据，所以方便高效压缩。在进行压缩的时候，请注意：一个压缩数据块由头信息和压缩数据两部分组成，头信息固定使用 9 位字节表示，具体由 1 个 UInt8（1字节）整型和 2 个 UInt32（4字节）整型组成，分别代表使用的压缩算法类型、压缩后的数据大小和压缩前的数据大小。每个压缩数据块的体积，按照其压缩前的数据字节大小，都被严格控制在 64KB ~ 1MB，其上下限分别由 min_compress_block_size（默认65536=64KB）与 max_compress_block_size（默认1048576=1M）参数指定。具体压缩规则：

原理的说法：每 8192 条记录，其实就是一条一级索引 一个索引区间 压缩成一个数据块。自适应压缩

如果：
a = 10kb
b = 你的一个8192条数据的大小
c = 1M

1、单个批次数据 size < 64KB：如果单个批次数据小于 64KB，则继续获取下一批数据，直至累积到size >= 64KB时，生成下一个压缩数据块。如果平均每条记录小于8byte，多个数据批次压缩成一个数据块

2、单个批次数据 64KB<= size <=1MB：如果单个批次数据大小恰好在 64KB 与 1MB 之间，则直接生成下一个压缩数据块。

3、单个批次数据 size > 1MB：如果单个批次数据直接超过 1MB，则首先按照 1MB大小截断并生成下一个压缩数据块。剩余数据继续依照上述规则执行。此时，会出现一个批次数据生成多个压缩数据块的情况。如果平均每条记录的大小超过 128byte,则会把当前这一个批次的数据压缩成多个数据块

总结：在一个 xxx.bin 字段存储文件中，并不是一个压缩块对应到一条一级索引！

总结：一个 [Column].bin 其实是由一个个的压缩数据块组成的。每个压缩块的大小在：64kb - 1M 之间。

2.3.6. 数据标记

关于数据标记：数据标记文件也与 .bin 文件一一对应。即每一个列字段 [Column].bin 文件都有一个与之对应的 [Column].mrk2 数据标记文件，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。它们分别表示在此段数据区间内，在对应的 .bin 压缩文件中，压缩数据块的起始偏移量；以及将该数据压缩块解压后，其未压缩数据的起始偏移量。每一行标记数据都表示了一个片段的数据（默认8192行）在 .bin 压缩文件中的读取位置信息。标记数据与一级索引数据不同，它并不能常驻内存，而是使用 LRU（最近最少使用）缓存策略加快其取用速度。

总结数据读取流程：**先根据一级索引，找到标记文件中的对应数据压缩块信息**（压缩块在 .bin 文件中的起始偏移量和未压缩之前该条数据的是偏移量）然后从 .bin 文件中，把压缩块加载到内存，解压缩之后，执行读取。

建立了 主键索引 到 数据文件的 映射！

2.3.7. 查询数据

```
select name from student where date = 201905;
```

指定分区 ==> 指定字段（xxx.bin ） ==> 根据一级索引(primary.idx)定位到 标记文件(name.mrk2)中的那一条记录 ==> 扫描对应字段的 mark 标记文件获取两个偏移量信息（当前要查找的数据，处于这个 .bin 数据文件中的那个 压缩数据块，这个压缩数据块在 .bin 文件的偏移量， 这个压缩数据块解压缩出来之后，要找的数据在当前这个压缩数据快的偏移量） ==> 根据第一个偏移量去 .bin 文件中定位到一个 压缩数据快 ==> 读取数据到内存执行解压缩 ==> 根据第二个偏移量去内存解压缩数据中找到对应的数据

关于分区 关于一级索引 关于二级索引 关于数据压缩 关于数据标记

提高数据查询效率的核心原则只有一个：谁做的辅助动作能快速的帮助我们去快速降低待搜寻的数据范围

分布式系统的核心思想：分而治之，必须提供一套架构方便用户的请求被快速的定位到某个单台服务器去处理。一般来说，这个服务器处理这个请求，都是很快的！

3. 本次课程总结

本次课程，主要的内容点，是通过一次课的时间，给大家讲解关于 ClickHouse 的表引擎详解和架构原理剖析

```
1、关于表引擎详解
    Log
    Integration
    Special
    MergeTree
2、关于架构原理
    数据分区
    列式存储
    一级索引
    二级索引
    数据压缩
    数据标记
    数据查询
```