Databend

Databend

聚合索引读取流程

RinChanNOW

Databend

Content

Databend

Part 1

什么是聚合索引

Databend

背景：物化视图

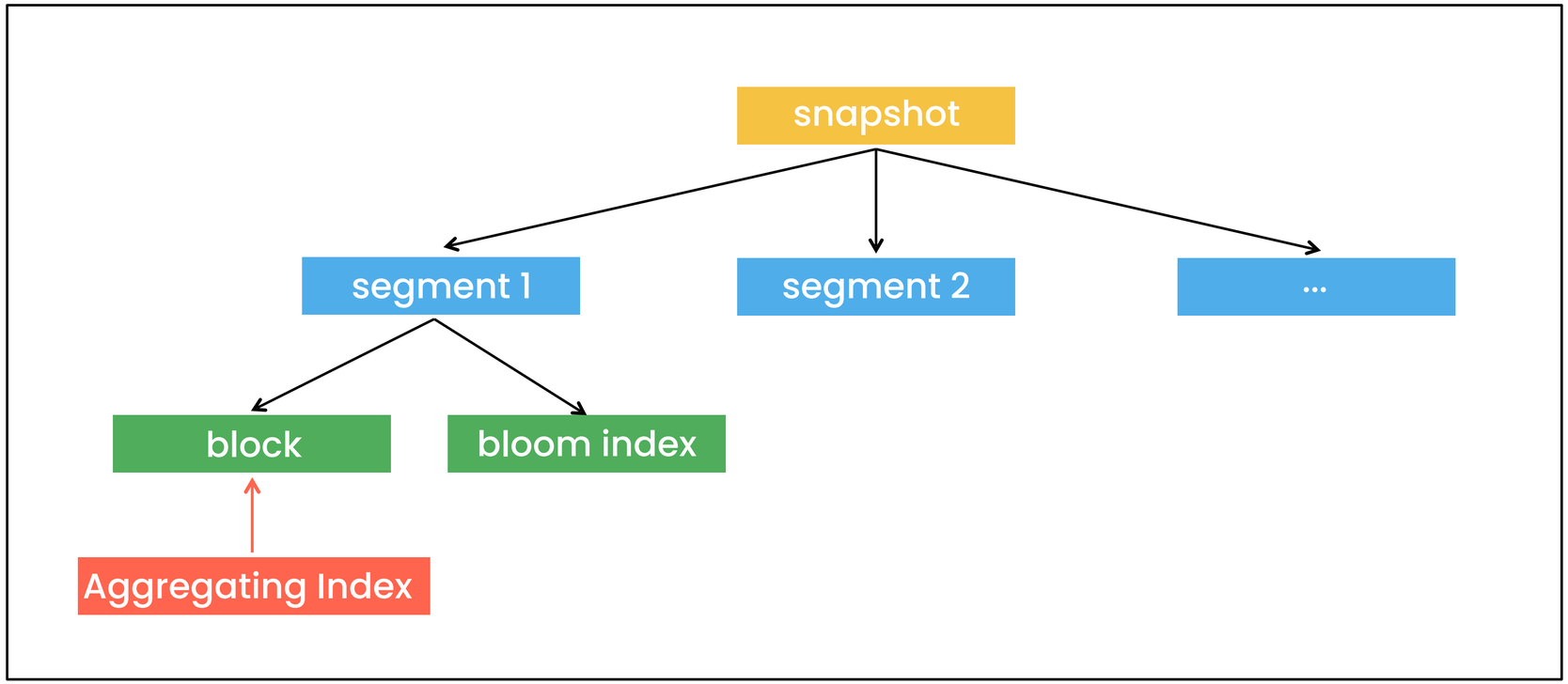
海量原始数据导入到数据库后，随着数据的增长，实时分析的压力会越来越大 ，大部分原因在于扫描和计算的数据变多。一些近似的查询 虽然可以复用数据的 cache 来减少 IO 的开销，但 cache 存在时效性和容量的局限，也无法解决增量计算的问题。

业界有许多 OLAP 数据库采用了预聚合模型来优化查询性能，也有一些实时流数据库用物化视图来管理聚合状态。

数据库	预聚合方式
ClickHouse	Aggregate MergeTree、Projection
Doris	类似 Aggregate MergeTree，但易用性更强
Firebolt	Aggregating Index 索引，类似 Projection
Snowflake	Aggregating Index 索引，但是以物化视图的方式暴露，异步刷新

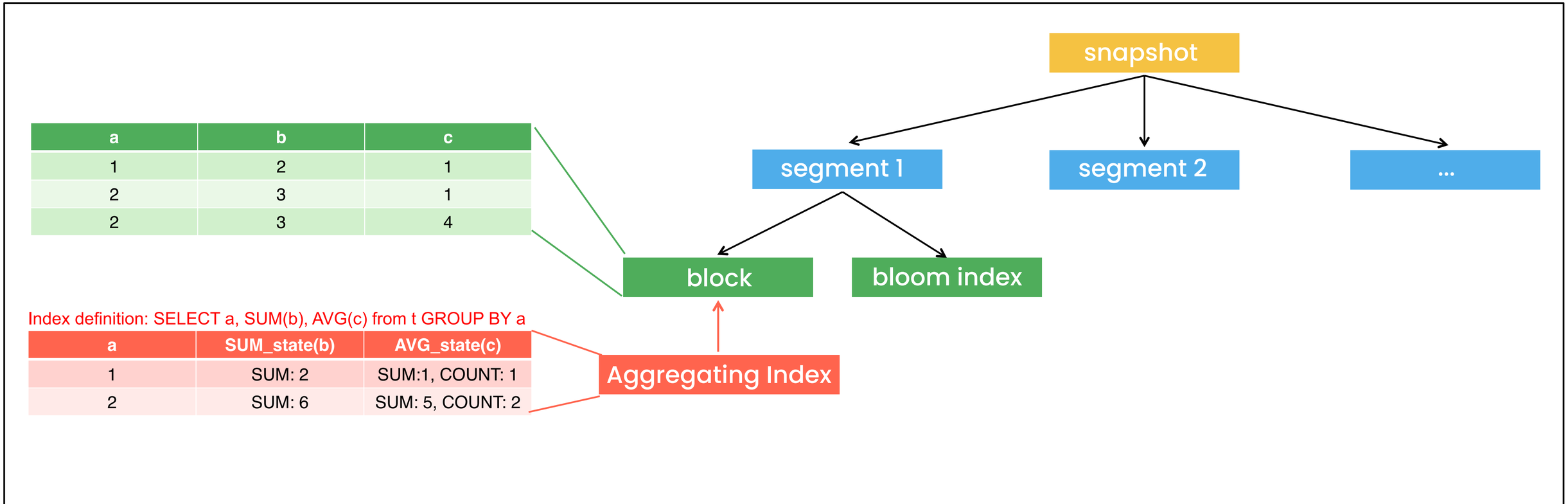
聚合索引

基于 Serverless 的设计理念，databend-query 会尽量少地维护运行时的状态。因此 Databend 选择采用聚合索引的方式来对数据进行预聚合，并通过 SQL 命令的方式对外部（上层应用）暴露异步刷新索引的接口。



聚合索引

类似于 Bloom Index，聚合索引底层存储中的 Block 是一一对应的关系。每一个聚合索引块的存储内容为对对应 Block 进行预聚合计算的结果。对于聚合函数，聚合索引中存放的是聚合的中间状态（即序列化后的 State 结构体，用于正式查询时进行合并）。



a	b	c
1	2	1
2	3	1
2	3	4

a	SUM_state(b)	AVG_state(c)
1	SUM: 2	SUM:1, COUNT: 1
2	SUM: 6	SUM: 5, COUNT: 2

相关命令

- **创建聚合索引：** CREATE AGGREGATING INDEX index_name AS SELECT ...;
- **删除聚合索引：** DROP AGGREGATING INDEX index_name;
- **刷新聚合索引：** REFRESH AGGREGATING INDEX index_name LIMIT n;

一些限制：

- 只能将简单的查询语句作为聚合索引的定义。（不支持子查询、多表查询等）
- 目前支持的聚合函数有限，在充分测试后会逐步开放更多聚合函数。

Databend

Part 2

使用聚合索引优化查询

Databend

查询改写

创建了聚合索引之后，便可以通过聚合索引来优化查询了。

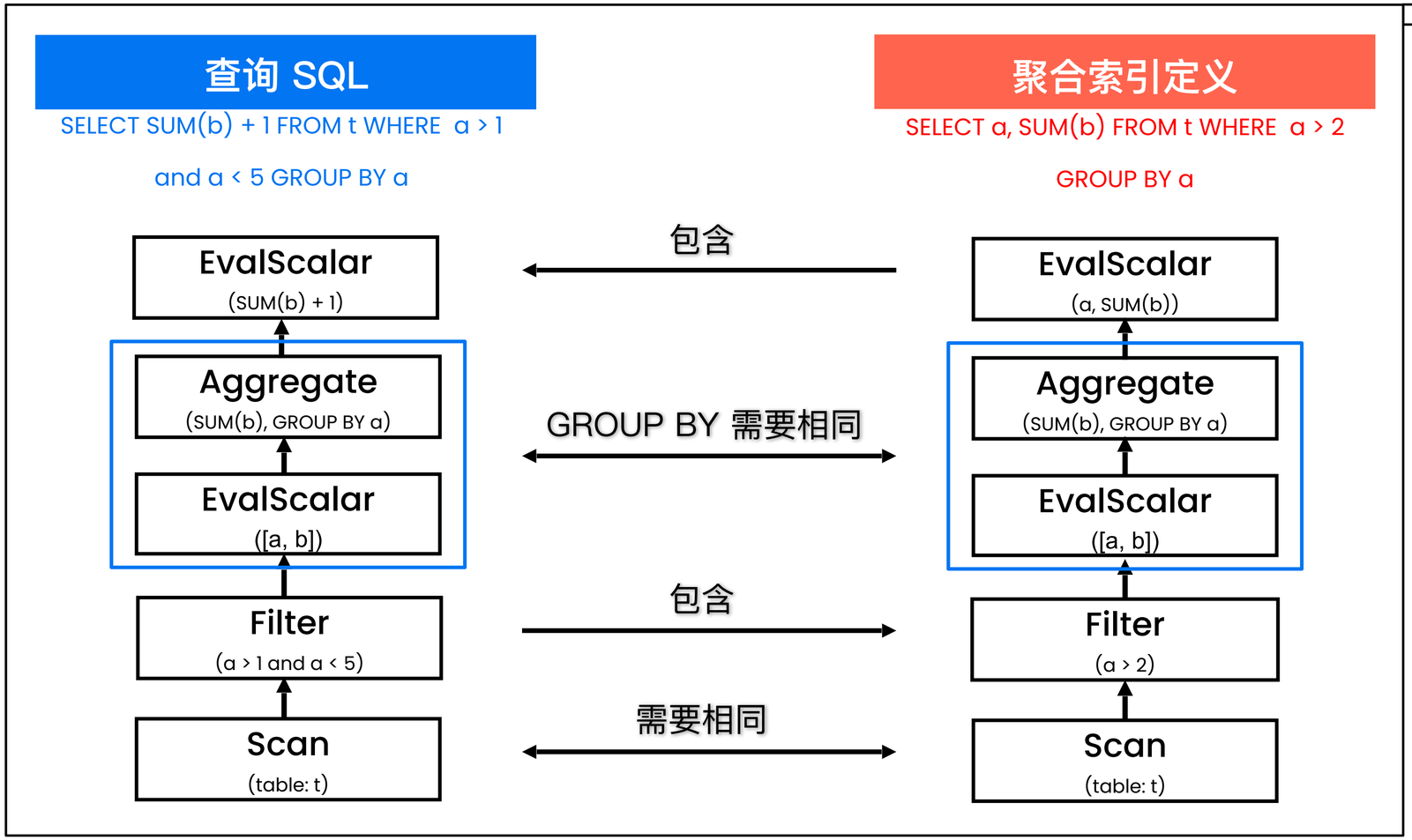
利用聚合索引进行查询主要分为两个步骤：

1. 查询计划改写。对原始查询计划进行分析，找出能够使用的聚合索引，改写底层数据读取计划。

2. 查询执行改写。根据被改写的查询计划，生成的查询流水线也需要进行相应的改动。

查询计划改写

实现上，Databend 利用 RBO 框架（基于规则）对原始计划进行分析，找出与查询 SQL 匹配的聚合索引。



匹配规则：

- 查询输出列需要包含在索引输出中。
- 聚合 GROUP BY 列需要相同。
- 索引聚合函数需要包含查询聚合函数。
- 索引输出列需要包含 GROUP BY 列。
- 查询过滤条件需要包含索引过滤条件。
- 底层数据表需要相同。

- 使用第一个能匹配上的聚合索引。将聚合索引信息下推到底层 Scan 计划中。

查询执行改写

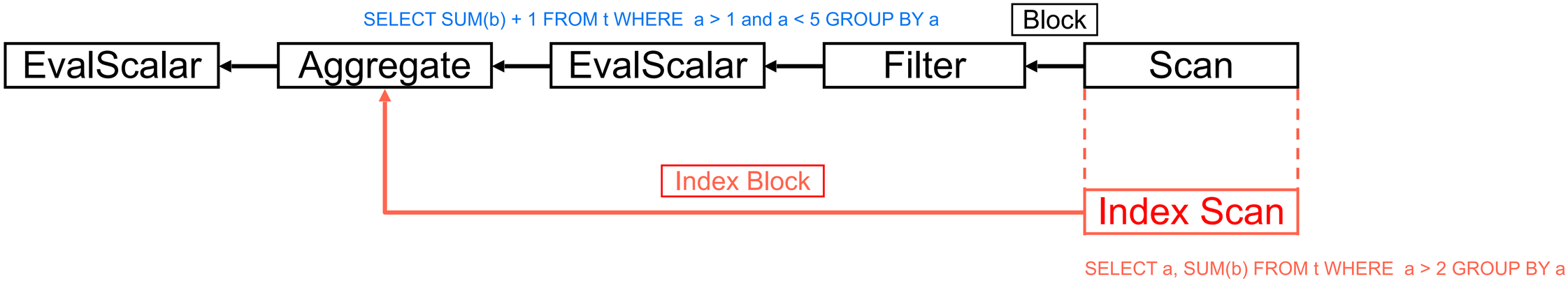
- 方案一：构造一个全新的 Source 读取聚合索引 Block，通过 Union 汇总 Table Source 读取的 Block。
- 方案二：改写现有 Table Source，支持聚合索引读取。

Databend 采用方案二

- 不需要预先确定读哪些 Index Block，在读取运行时进行检测。

- 无需创建 Union，简化流水线。

查询执行改写



在执行原本的 Scan 时，在读取某一个 Block 时，先检测是否存在相应的 Index Block。如果存在，则读取 Index Block，并跳过后续表达式计算与条件过滤环节直接通过 Aggregate 算子进行聚合。

Databend

Part 3

性能测试

Databend

测试环境

- OS: Ubuntu 18.04
- CPU: Intel Xeon Gold 5218R @ 80x 3.433GHz
- RAM: 92 GB
- Disk: 9 TB

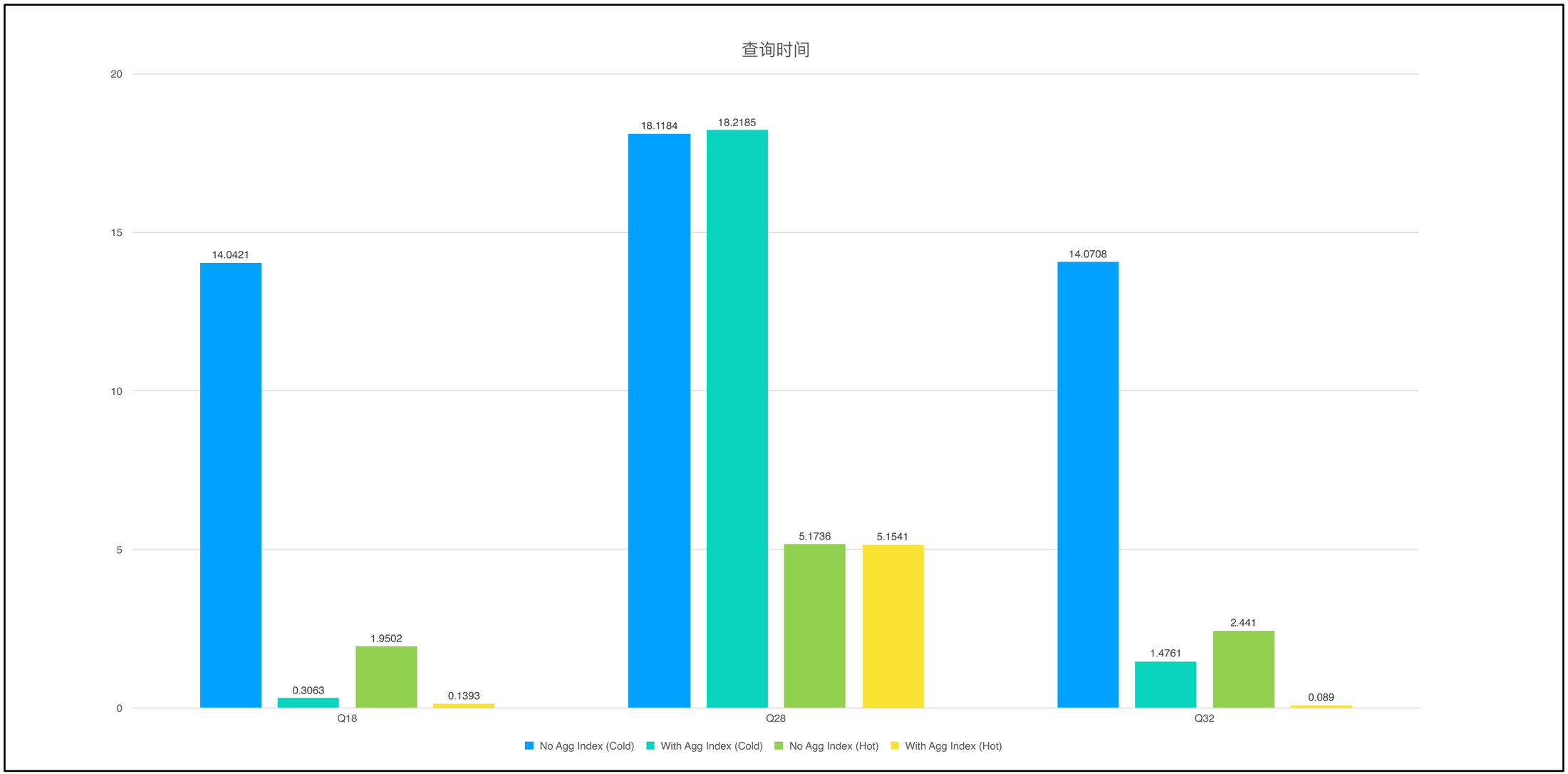
- Dataset: hits (70 GB)
- Storage format: Parquet
- Storage backend: Disk

Databend

测试语句

```
-- Q18
CREATE AGGREGATING INDEX i_q18 AS
    SELECT UserID, extract(minute FROM EventTime) AS m, SearchPhrase, COUNT(*)
    FROM hits
    GROUP BY UserID, m, SearchPhrase;
REFRESH AGGREGATING INDEX i_q18;
SELECT UserID, extract(minute FROM EventTime) AS m, SearchPhrase, COUNT(*)
FROM hits
GROUP BY UserID, m, SearchPhrase
LIMIT 10;
-- Q28
CREATE AGGREGATING INDEX i_q28 AS
    SELECT REGEXP_REPLACE(Referer, '^https?://(?:www\.)?([^/]+)/.*$', '\1') AS k, AVG(length(Referer)) AS l, COUNT(*) AS c, MIN(Referer)
    FROM hits
    WHERE Referer <> ''
    GROUP BY k;
REFRESH AGGREGATING INDEX i_q28;
SELECT REGEXP_REPLACE(Referer, '^https?://(?:www\.)?([^/]+)/.*$', '\1') AS k, AVG(length(Referer)) AS l, COUNT(*) AS c, MIN(Referer)
FROM hits
WHERE Referer <> ''
GROUP BY k
LIMIT 25;
-- Q32
CREATE AGGREGATING INDEX i_q32 AS nWidth)
    FROM hits
    SELECT WatchID, ClientIP, COUNT(*) AS c, SUM(IsRefresh), AVG(Resolutio
    GROUP BY WatchID, ClientIP;
REFRESH AGGREGATING INDEX i_q32;
SELECT WatchID, ClientIP, COUNT(*) AS c, SUM(IsRefresh), AVG(ResolutionWidth)
FROM hits
GROUP BY WatchID, ClientIP
LIMIT 10;
```

测试结果


查询时间
20
15
10
5
0
14.0421
0.3063
1.9502
0.1393
18.1184
18.2185
5.1736
5.1541
14.0708
1.4761
2.441
0.089
Q18
Q28
Q32
No Agg Index (Cold)
With Agg Index (Cold)
No Agg Index (Hot)
With Agg Index (Hot)

Databend

Part 4

总结

Databend

总结

Databend 通过索引的方式为每一个 Block 建立聚合索引，提前计算出聚合的结果来实现物化视图的功能。

查询时，通过读取聚合索引中预先计算好的聚合结果，减少实际聚合的计算量，提升查询性能。

未来工作

Databend 还能够继续提升查询改写的能力，以支持更多的查询语句。

Databend

Databend

聚合索引写入流程

AriesDevil

Databend

Content

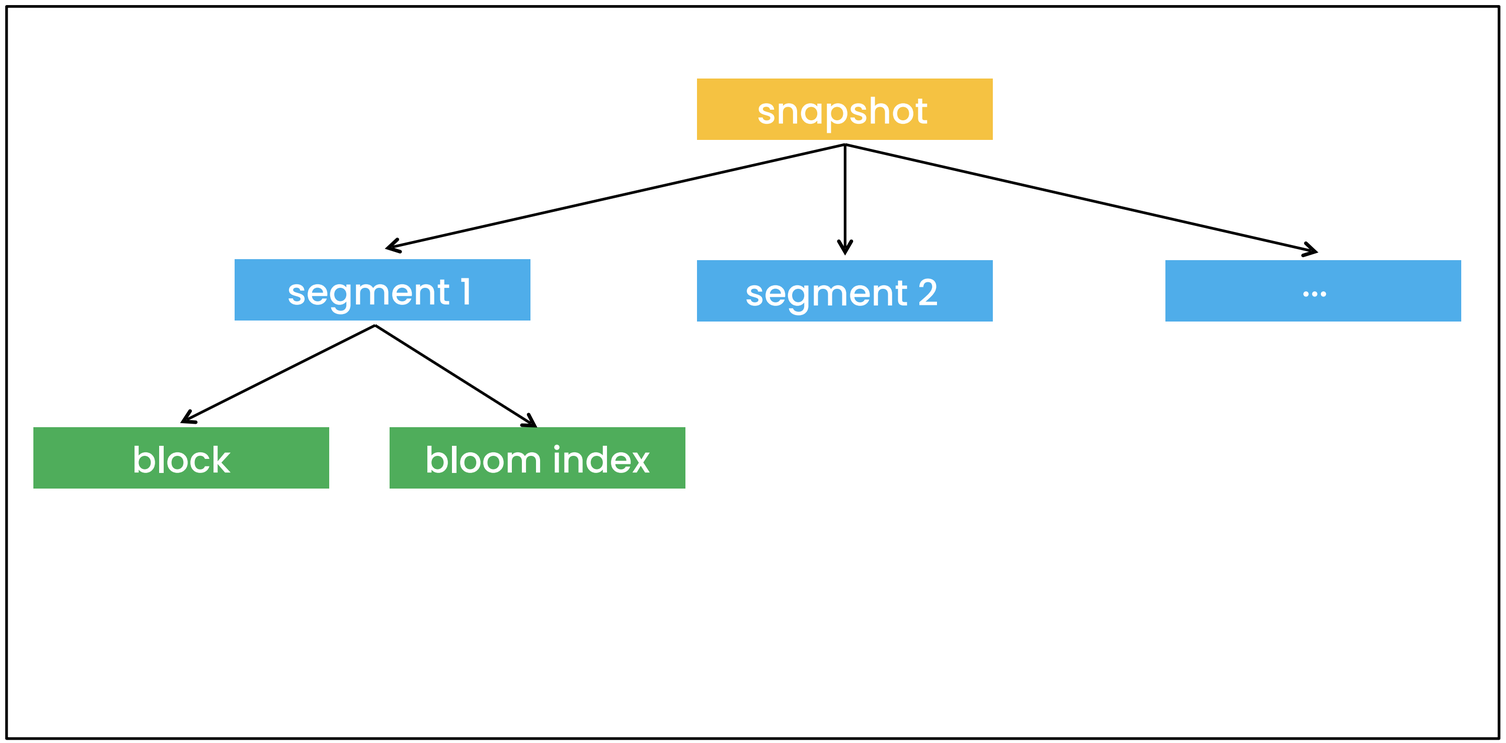
Databend

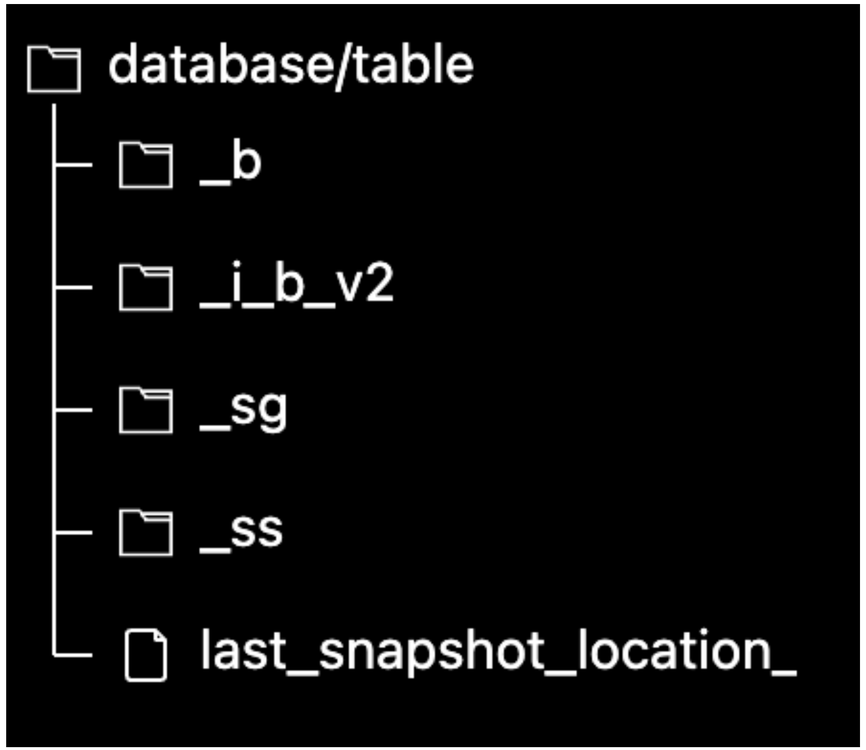
Part 1

Databend 存储结构

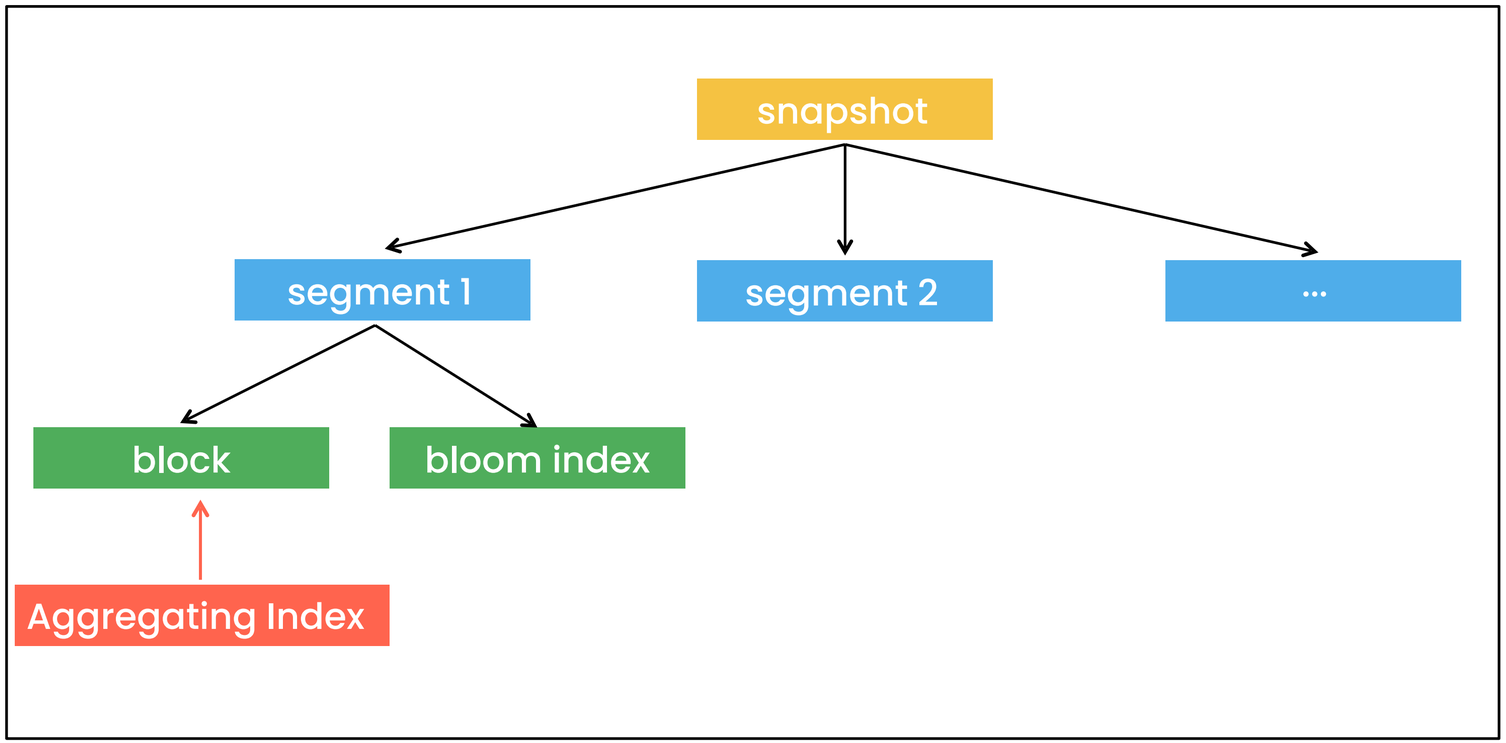
Databend

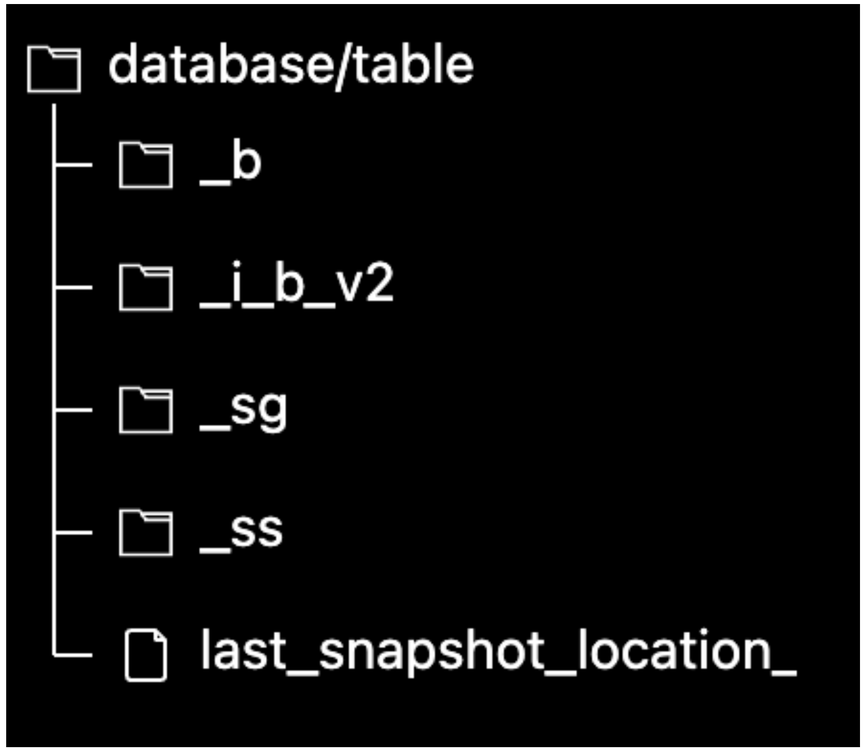
Databend 存储架构





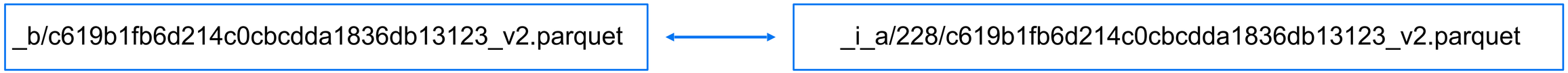
Databend 存储架构





Databend 存储架构

目录	含义
_b	用于存储真正的 block，以 parquet 或者 native(stawboat) 格式存储
_i_a	用户存储聚合索引，文件名以及后缀与 _b 下面的数据文件相同
_i_b_v2	block 文件的 bloom filter 索引，以 parquet 格式存储
_sg	segment 用户管理 blocks，JSON 格式，sg 文件包含 >=1 && <= 1000 个 blocks
_ss	snapshot 用户关联某个版本对应的 segments
last_snapshot_location_hint	指向最后一个 snapshot 存储的地址



Databend

Part 2

聚合索引的生成

聚合索引生成

- **生成聚合索引：**REFRESH AGGREGATING INDEX <index_name> [LIMIT n];

生成聚合主要分为三个步骤：

1. 原始 query 改写，对创建聚合索引的 sql 进行改写。
2. 改写底层数据读取计划。
3. 执行改写后的计划，裁剪数据列，写入索引文件。

Databend

Part 3

具体实现

Query 改写

改写规则：

- 改写原始聚合函数为聚合中间状态。
- 聚合索引生成的文件需要与原始数据文件一一对应，此处增加一个原始数据文件名的内部列。

```
SELECT a, SUM(b)
FROM t
WHERE  a > 2
GROUP BY a
```

```
SELECT a, SUM_STATE(b), _block_name
FROM t
WHERE  a > 2
GROUP BY a, _block_name
```

Databend

改写底层数据读取

- 根据改写 sql 后的逻辑计划生成物理计划
- 遍历物理计划，找到 DataSourcePlan 并改写
- 构建查询 pipeline

```rust
pub struct IndexMeta {
    pub table_id: MetaId,

    pub index_type: IndexType,

    pub created_on: DateTime<Utc>,
    pub dropped_on: Option<DateTime<Utc>>,
    pub updated_on: Option<DateTime<Utc>>,

    pub query: String,
}
```

Databend

裁剪列并持久化

- sink 收集到查询结果后，裁剪掉 block name 列，按照 block name 对数据进行分组
- 等待 pipeline 执行完成后，按照 storage type 写入文件
- 当前实现需要在 sink 中积累数据，可能导致 oom，建议使用 limit 参数少量多次

整体 pipeline

```
+-------------------------------------------------------------------+
| AggIndexSink × 1 processor                                        |
|   CompoundBlockOperator(Project) × 1 processor                    |
|     CompoundBlockOperator(Map) × 1 processor                      |
|       TransformFinalAggregate × 1 processor                       |
|         TransformSpillReader × 1 processor                        |
|           TransformPartitionBucket × 1 processor                  |
|             TransformAggregateSpillWriter × 1 processor           |
|               TransformPartialAggregate × 1 processor             |
|                 CompoundBlockOperator(Filter) × 1 processor       |
|                   CompoundBlockOperator(Project) × 1 processor    |
|                     FillInternalColumnProcessor × 1 processor     |
|                       DeserializeDataTransform × 1 processor      |
|                         SyncReadParquetDataSource × 1 processor   |
+-------------------------------------------------------------------+
```

Databend

Part 4

后续规划

Databend

后续规划

- 当前改写 DataSourcePlan 需要加载整个 partitions 元信息，后续会改成流式读取过滤
- 索引删除或者对应table删除后的数据清理

Databend

产品介绍

Databend 是一个使用 Rust 研发、完全面向云架构的云原生数仓，提供极速的弹性扩展能力，致力于打造按需、按量的 Data Cloud 产品。

开源 Cloud Data Warehouse 明星项目

真正的存储、计算分离架构，高性能、低成本，按需按量使用

支持基于同一份数据的多租户读写、共享操作

完整的数据库支持，兼容 MySQL，ClickHouse 协议，SQL Over HTTP 等

高弹性 + 强分布式，致力于解决大数据分析成本和复杂度问题

Databend

Part 5

Q & A

Databend

Thank you!