DORIS | SELECTDB Webinar Real-Time

# 「实时分析」系列直播－数据更新

7月31日 周四 19:30-20:30

观看直播

# 实时分析场景-数据更新

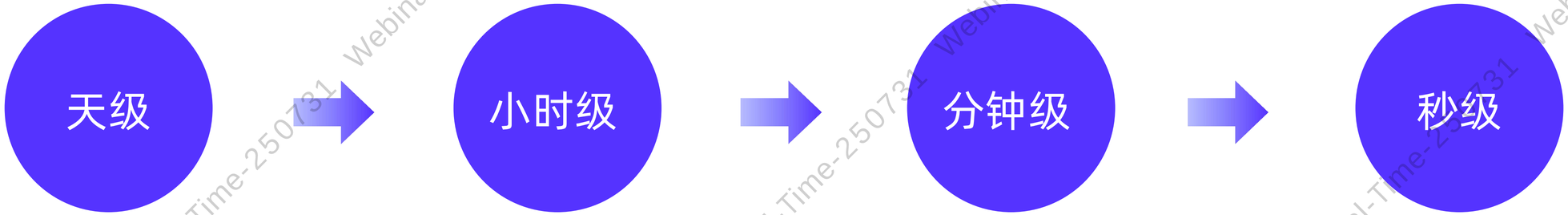
**张晨-SelectDB资深研发工程师，Apache Doris Committer**

# 目录

# 目录

# 分析性能越来越快

**大数据量的分析延迟越来越低**




**大部分 OLAP 系统的数据新鲜度还停留在天级或者小时级别**

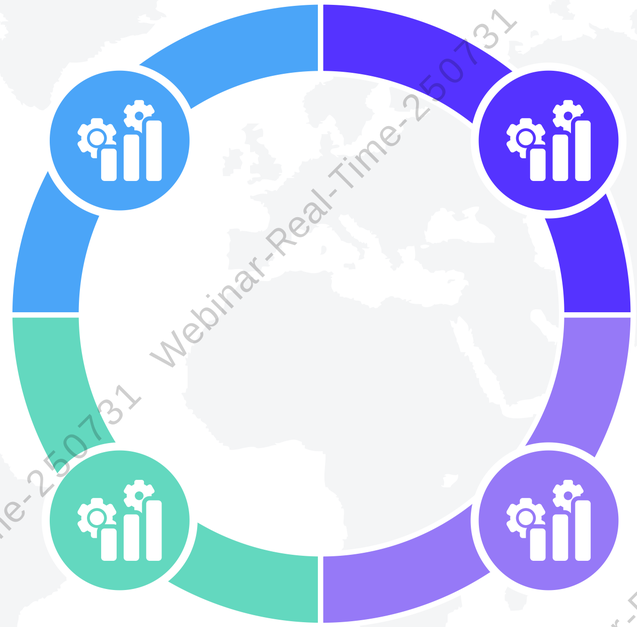
# 业务决策加速

## 从 T+1 到实时决策的跃迁

传统 T+1 数据更新模式导致决策滞后，实时数据流使业务响应速度提升 95%，抢占市场先机。

## 动态数据驱动的商业闭环

订单状态/用户标签实时更新构建敏捷反馈系统，决策周期从小时级压缩至秒级，重塑商业逻辑。



## CDC技术赋能实时同步

通过变更数据捕获技术，业务库与数仓毫秒级同步，消除传统 ETL 的批次间隔瓶颈。

## 零延迟修正的数据可信度

错误数据秒级回补确保分析准确性，避免"过期数据陷阱"，保障决策引擎的高可靠性。

# 目录

# 目录

# 表模型

**主键模型**
**Unique Key**

- 支持行级别 UPSERT，为实时更新而生
- 支持 Merge-on-Write（MoW）模式，查询效率接近明细表，兼顾高效更新和查询性能
- 支持部分列更新，灵活更新部分字段
- 适合频繁变更、实时性强的业务场景

**聚合模型**
**Aggregate Key**

- 根据 Key 列聚合数据，Doris 存储层保留聚合后的数据
- 从而可以减少存储空间和提升查询性能
- 通常用于需要汇总或聚合信息（如总数或平均值）的情况

**明细模型**
**Duplicate Key**

- 不支持更新，仅支持追加写入，允许重复key
- 适合全量追加场景，如日志采集、埋点数据

# 不同表模型对更新能力的支持

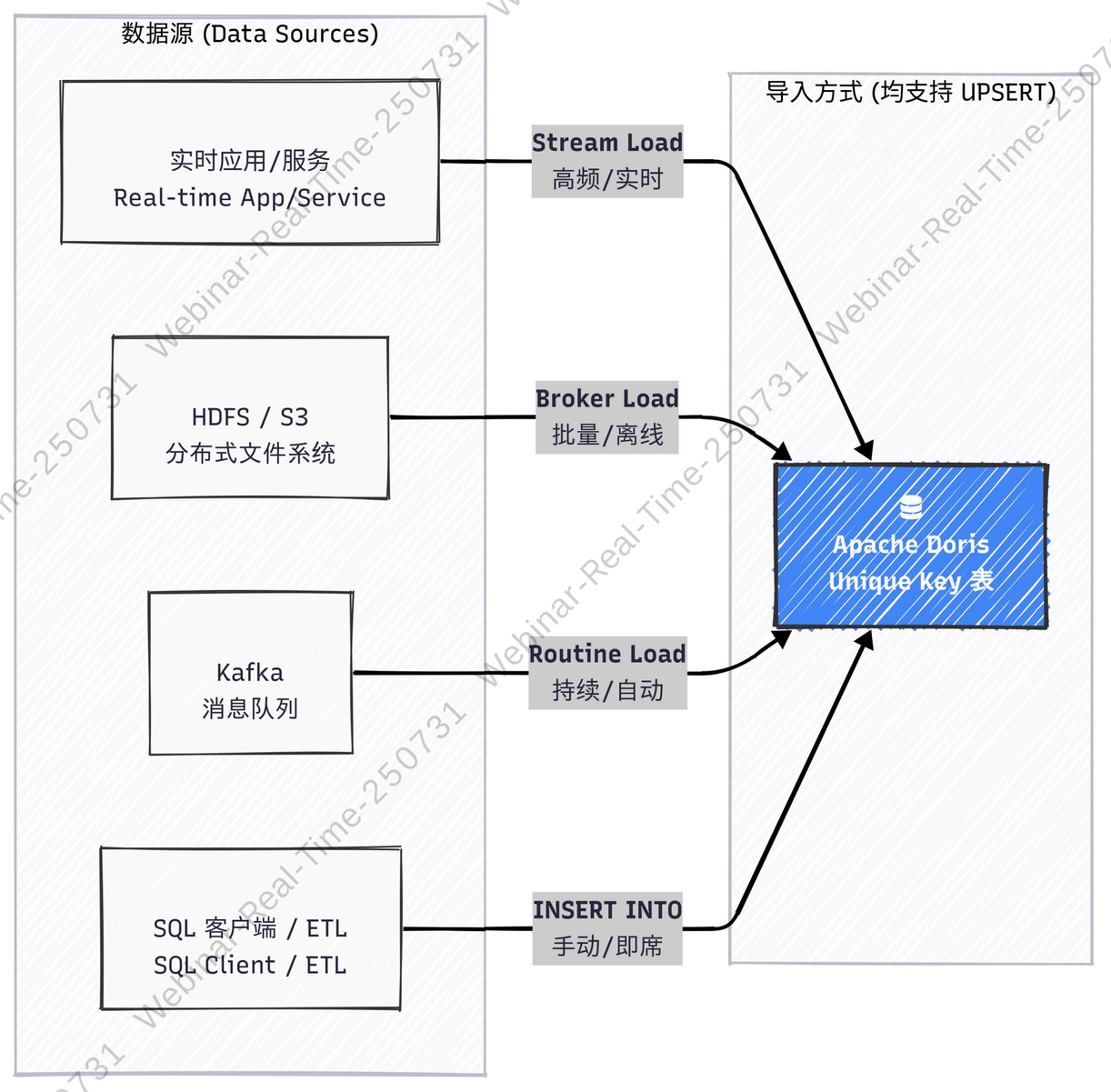
| | Unique Key | Aggregate Key | Duplicate Key |
|---|---|---|---|
| 通过导入进行更新 | 支持 | | |
| 通过导入进行部分列更新 | 支持 | | |
| 条件更新 | 支持 | | |
| 通过导入进行删除 | 支持 | | |
| UPDATE 语句 | 支持 | | |
| DELETE 语句 | 支持 | 支持（仅 Key 列） | 支持 |

# 更新方式-通过导入进行更新

支持多种导入方式，均支持 UPSERT 语义（对 Unique Key 表）

- Stream Load：适合批量或实时数据导入
- Broker Load：支持分布式文件导入
- Routine Load：自动持续拉取消息队列数据
- INSERT INTO：交互式写入

特别说明：需要开启 MoW 模式，并配置 Partial Update 开关，才能支持部分列更新



# 更新方式-通过 DML 语句更新

## UPDATE 语句

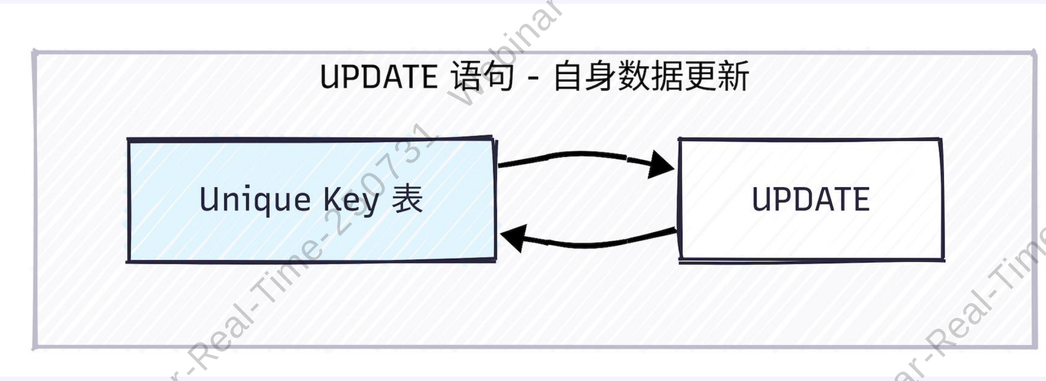
- 先根据 where 条件扫描数据，进行加工之后写回表中
- 常见使用方式

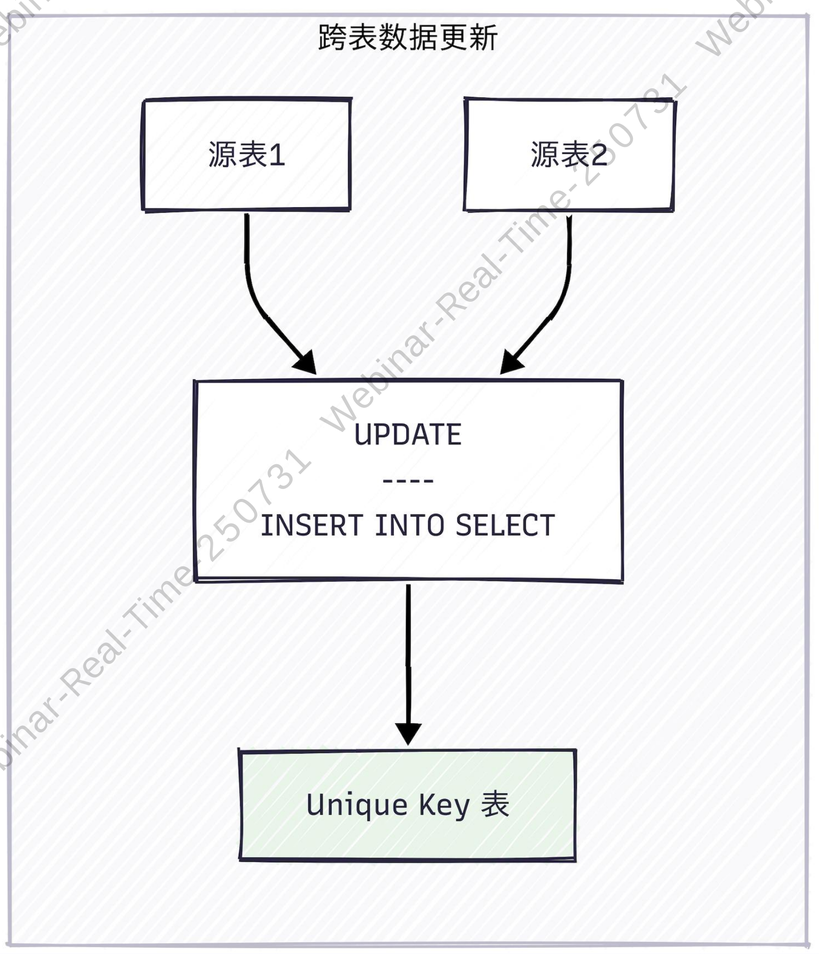
```
UPDATE tbl_name
SET v1 = 1
WHERE k1=1 and k2=2;
```

```
UPDATE t1
SET t1.c1 = t2.c1,
    t1.c3 = t2.c3 * 100
FROM t2 INNER JOIN
     t3 ON t2.id = t3.id
WHERE t1.id = t2.id;
```

## INSERT INTO SELECT 语句

- 效果与 UPDATE 语句类似
- 能使用隐藏列(__DORIS_DELETE_SIGN__, __DORIS_SEQUENCE_COL__ 实现更丰富的语义控制



# 删除方式-通过导入进行删除

与更新类似，所有的导入方式都支持在导入过程中增加一列隐藏列__DORIS_DELETE_SIGN__来对数据进行标记删除

Stream Load：–H "delete: label_c3=1"

Broker Load：DELETE ON label_c3=true

Routine Load：

COLUMNS(k1, k2, k3, v1, v2, label),

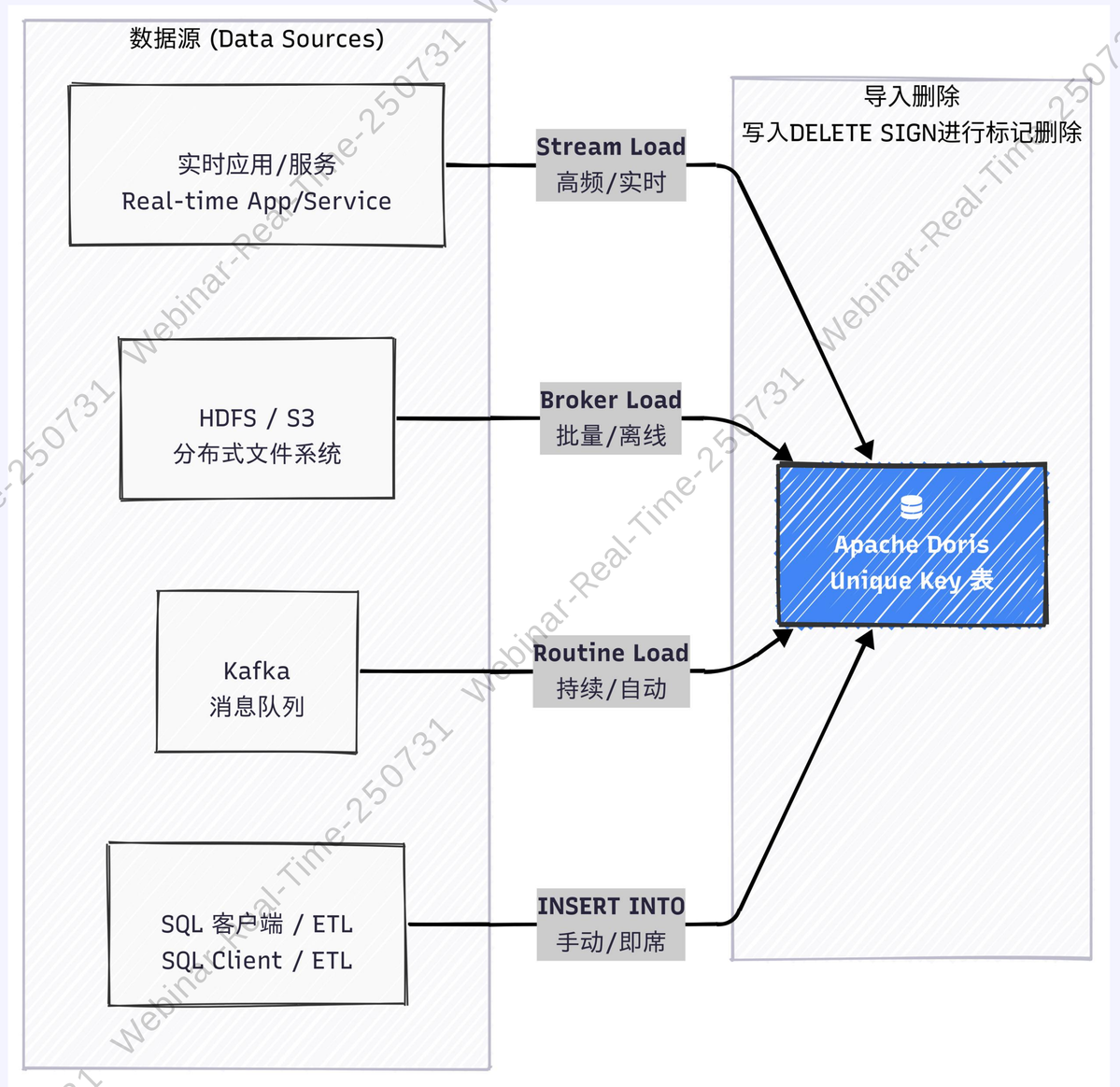
[DELETE ON label=true]

INSERT INTO：

insert into example_table (id, __DORIS_DELETE_SIGN__)

values (1, 1);

注：后边会展开讲解 delete sign 的原理。



# 通过DELETE语句进行删除

| | 主键模型 | 聚合模型 | 明细模型 |
|---|---|---|---|
| 实现方式 | Delete Sign | Delete Predicate | Delete Predicate |
| 限制 | 无 | 删除条件只能用于 Key 列 | 无 |
| 删除性能 | 一般 | 快 | 快 |

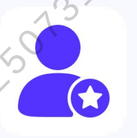
说明：

1. 主键模型会将要删除的 key（主键）的重新写入（带 delete sign 标记），因此删除速度取决于删除的数据量
2. 明细模型和聚合模型的删除操作只是记录一个 runtime filter，在查询时进行过滤，因此删除操作很快，速度与被删除数据量无关
3. 两种方式均不是立刻删除物理数据，要在 compaction 阶段对物理数据进行实际的删除

# 性能最佳实践

## UPDATE 语句不建议进行并发操作

- 并发的 UPDATE 在涉及到相同key时，无法保证隔离性。
- 只有在业务自己确认并发的 UPDATE 不涉及相同 Key 时才可以使用。

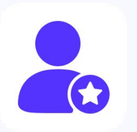


## 不建议使用 INSERT INTO 进行高频更新

- 每条 INSERT INTO 都会带来额外的事务成本。
- 不建议超过 100次/s的 INSERT INTO 高频写入。
- 如有需求考虑使用 Group Commit

## 明细表高频 delete 会严重影响查询性能

- 每一条delete都是个runtime filter，在查询时将被删除的数据过滤掉。
- 高频delete会引入严重的查询开销。



## 删除分区数据使用 Truncate Partition

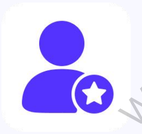
- 不建议使用 delete  删除分区数据。
- Truncate Partition 效率最高

# 目录

# 主键模型(Unique Key Model)能力图谱

## 整行 Upsert

- 基础的整行 Upsert 能力
- 支持高达十几万 TPS 的吞吐和秒级延迟



## 部分列更新

- 直接通过导入来更新部分字段
- 支持上万的 TPS 和高并发更新

## 条件更新

- 按照用户给定的 Sequence 值来决定 key 的更新顺序
- 适用于数据乱序到达时的强一致需求

## 复杂的UPDATE语句支持

- 复杂的 where 条件和多表 join 支持
- 仅支持单并发的更新任务

## 条件删除

- 通过额外的 Delete Sign 列来标识要删除的key列
- 适用于批量删除 key

## 复杂的DELETE语句支持

- 复杂的 where 条件和多表 join 支持
- 仅支持单并发的更新任务

# MoW vs MoR

| | Merge on Write | Merge on Read |
|---|---|---|
| **导入速度** | 导入过程中进行数据去重，小批量实时写入相比 MoR 约有 10%-20% 的性能损失，大批量导入（例如千万级/亿级数据）相比 MoR 约有 30%-50% 的性能损失 | 与明细表接近 |
| **查询速度** | 与明细表接近 | 需要在查询期间进行去重，查询耗时约为 MoW 的 3-10 倍 |
| **谓词下推** | 支持<br>能大大减少查询时需要 IO 的数据量<br>可以配合倒排索引使用，大幅提升查询性能 | 不支持 |
| **资源消耗** | • **导入资源消耗**：相比明细表和 MoR，约额外消耗 10%-30% 的 CPU。<br>• **查询资源消耗**：与明细表接近，无额外资源消耗。<br>• **Compaction 资源消耗**：相比 Duplicate Key，消耗更多内存和 CPU，具体取决于数据特征和数据量 | • **导入资源消耗**：与明细表相近，无额外资源消耗。<br>• **查询资源消耗**：相比明细表和MoW，查询时额外消耗更多的 CPU 和内存。<br>• **Compaction 资源消耗**：相比明细表，需更多内存和 CPU，具体数值取决于数据特征和数据量。 |

注：自 2.1 版本起，MoW 已经成为主键模型的默认实现。

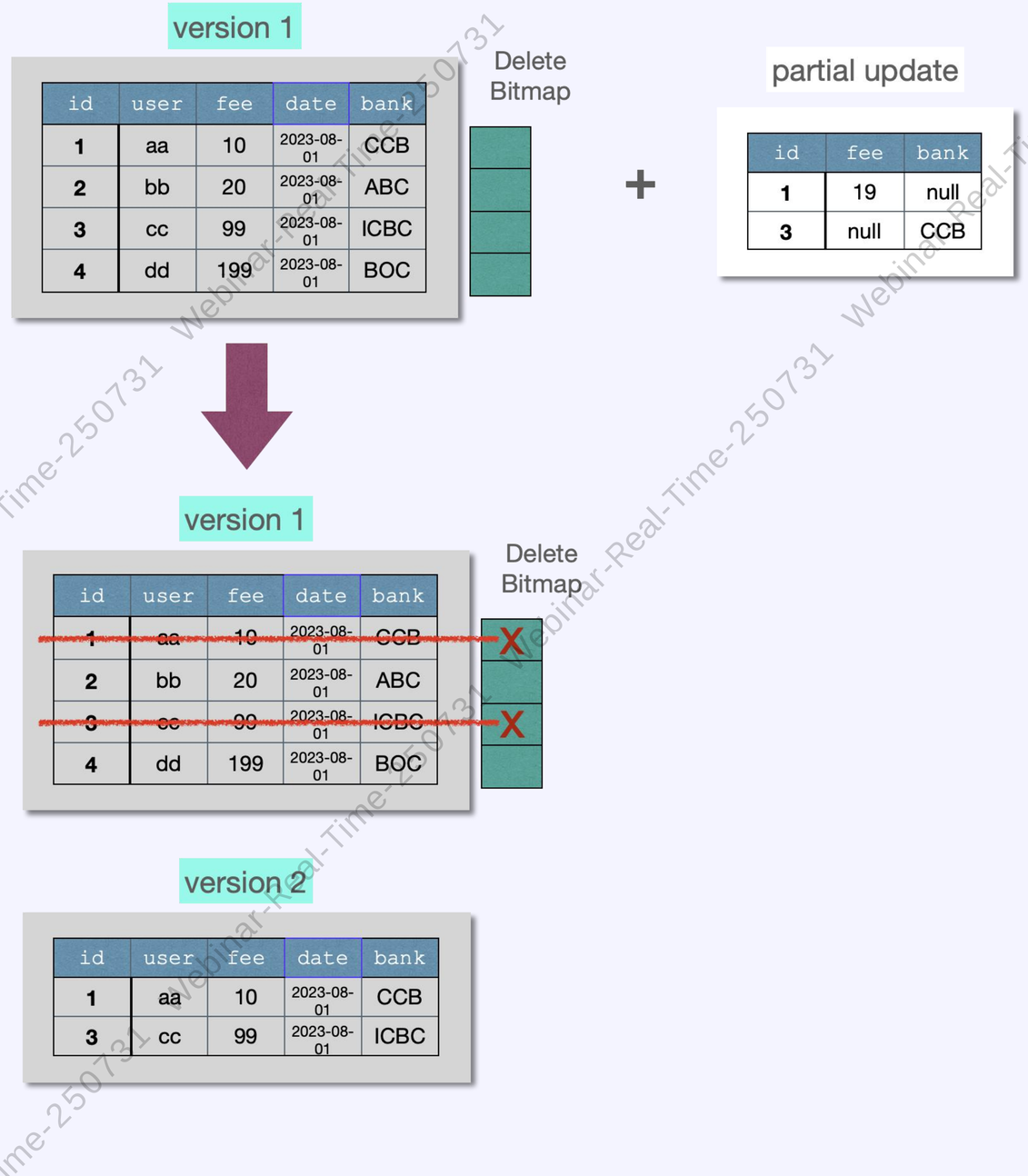
# 部分列更新

**列更新广泛存在于多种场景中**

- 数据修正
- 实时标签
- 宽表拼接
- …

**Doris 2.0 在 MoW 基础上，通过整行数据补齐，实现了高吞吐高并发的部分列更新能力支持**

- 该功能最好配合 nvme SSD 使用
- 高频更新，兼具极速的分析性能



# 部分列更新性能对比

## 影响性能的因素

- 硬盘类型：SSD/HDD
- 硬盘接口：NVMe/SATA
- CPU和内存是否充足
- 表的列数
- 是否开启行存
- 行存数据的膨胀率
- 导入的并发数
- 导入的频率
- 存量数据大小
（影响page cache命中率）

## 参考性能

### 配置

- 16 core CPU
- 64 GB 内存
- 400 GB SATA SSD X 6
- 机器数量 3 台

### 表 Schema 及导入参数

- 列数：200
- 数量：5000 万行
- 副本数：1
- 导入并发：5
- 导入间隔：5s

| 开启行存 | 写入性能 |
| --- | --- |
| 是 | 约 20000 行/ 秒 |
| 否 | 约 6000 行/ 秒 |

# Apache Doris Delete Sign 工作原理

## 1 原始数据表

| ID | Name | Age | __DORIS_DELETE_SIGN__ |
|---|---|---|---|
| 1 | Alice | 25 | 0 |
| 2 | Bob | 30 | 0 |
| 3 | Carol | 28 | 0 |
| 4 | David | 35 | 0 |

注：0 表示正常数据

## 2 执行删除操作

```
DELETE FROM table
WHERE ID = 2;
```

🚀 不直接删除数据行，仅标记

## 3 标记删除后的数据表

| ID | Name | Age | __DORIS_DELETE_SIGN__ |
|---|---|---|---|
| 1 | Alice | 25 | 0 |
| 2 | Bob | 30 | 1 |
| 3 | Carol | 28 | 0 |
| 4 | David | 35 | 0 |

注：1 表示已删除数据

## 4 查询过程中的过滤

```
SELECT * FROM table;
```

↓ 内部自动添加过滤条件

```
SELECT * FROM table
WHERE __DORIS_DELETE_SIGN__ = 0;
```

## 5 查询结果

| ID | Name | Age |
|---|---|---|
| 1 | Alice | 25 |
| 3 | Carol | 28 |
| 4 | David | 35 |

✗ Bob 的记录被过滤掉

## 6 Compaction 物理删除

🔄 后台 Compaction 进程：

- 定期扫描数据文件
- 合并多个数据文件
- 过滤掉 __DORIS_DELETE_SIGN__ = 1 的记录
- 生成新的数据文件，删除旧文件

⚡ 批量处理，高效清理

## 7 物理删除后的文件

| ID | Name | Age |
|---|---|---|
| 1 | Alice | 25 |
| 3 | Carol | 28 |
| 4 | David | 35 |

✅ Bob 的记录完全消失

## 🎯 Delete Sign 机制优势

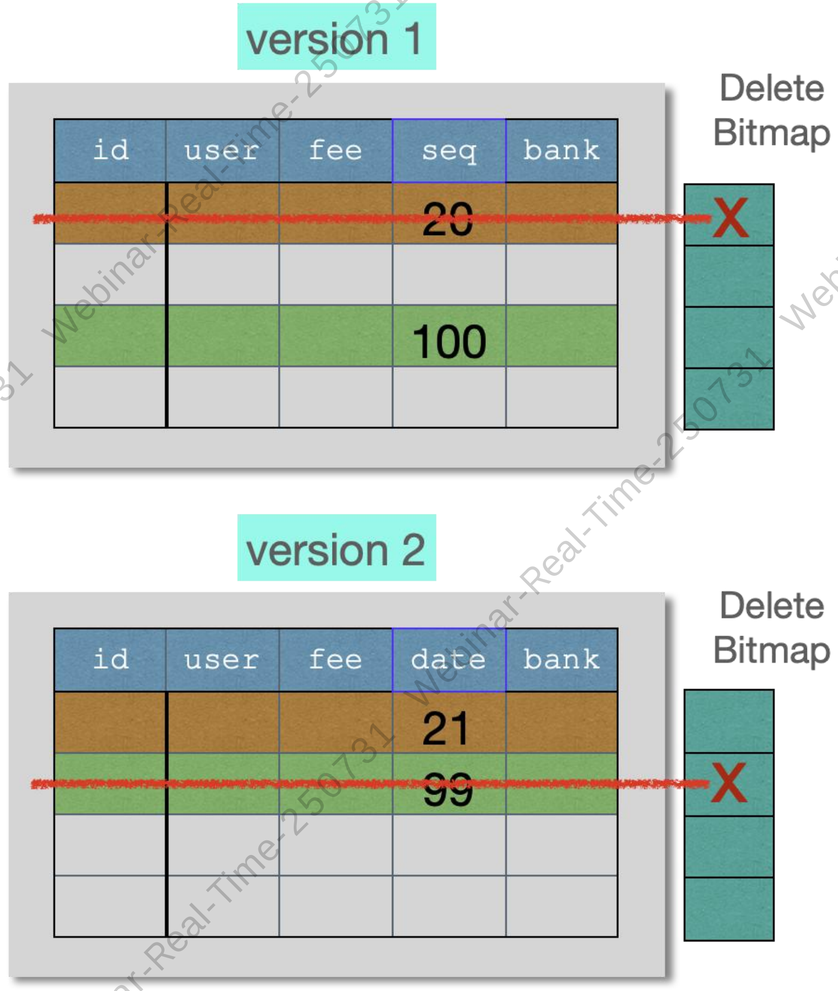
- **快速删除响应**
  只需标记，无需立即移动数据
- **事务一致性保证**
  标记操作是原子的，保证数据一致性
- **批量处理优化**
  Compaction 过程批量处理，提高效率
- **MVCC 支持**
  配合版本号实现多版本并发控制
- **存储空间回收**
  后台自动清理，释放存储空间
- **高并发友好**
  读写分离，不影响查询性能

# 条件更新（Sequence列）

**当数据乱序到达时，仍然能按照原始的更新顺序生效**

- 需要单独指定 Sequence Column
- 当数据乱序到达时，即使旧数据比新数据后写入 Doris，系统会保证旧数据不会覆盖新的数据

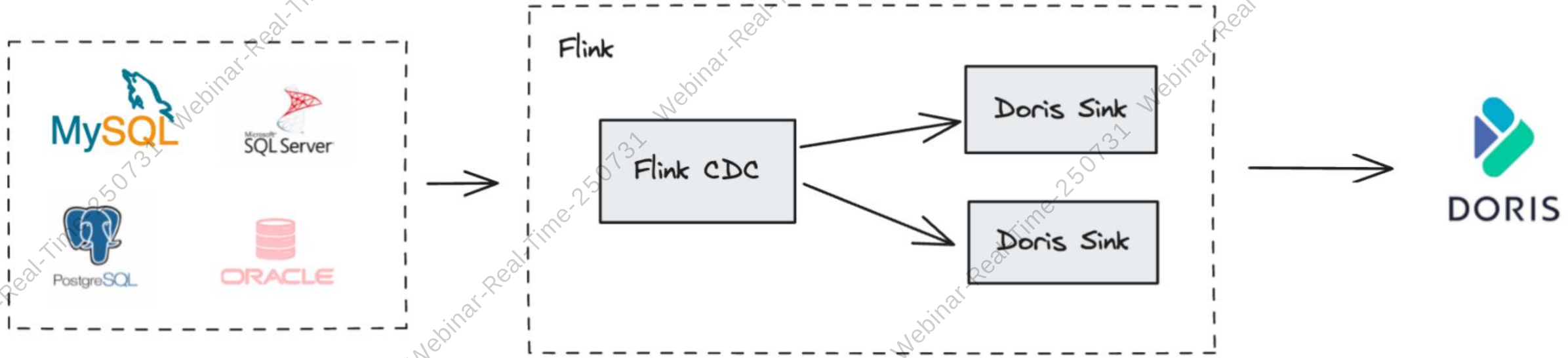
```
CREATE TABLE test.test_table
(
        user_id bigint,
        date date,
        group_id bigint,
        modify_date date,
        keyword VARCHAR(128)
)
UNIQUE KEY(user_id, date, group_id)
DISTRIBUTED BY HASH (user_id) BUCKETS 32
PROPERTIES(
         “function_column.sequence_col” = ‘modify_date‘,
         “replication_num" = “1”
);
```



# 目录

# CDC 数据实时同步

- 实时捕获数据库变更，写入 Unique Key 表
- 利用 seq 列保证写入顺序和数据一致性，使用 delete sign 实现对 binlog 删除记录的同步
- 支持全量 + 增量混合同步

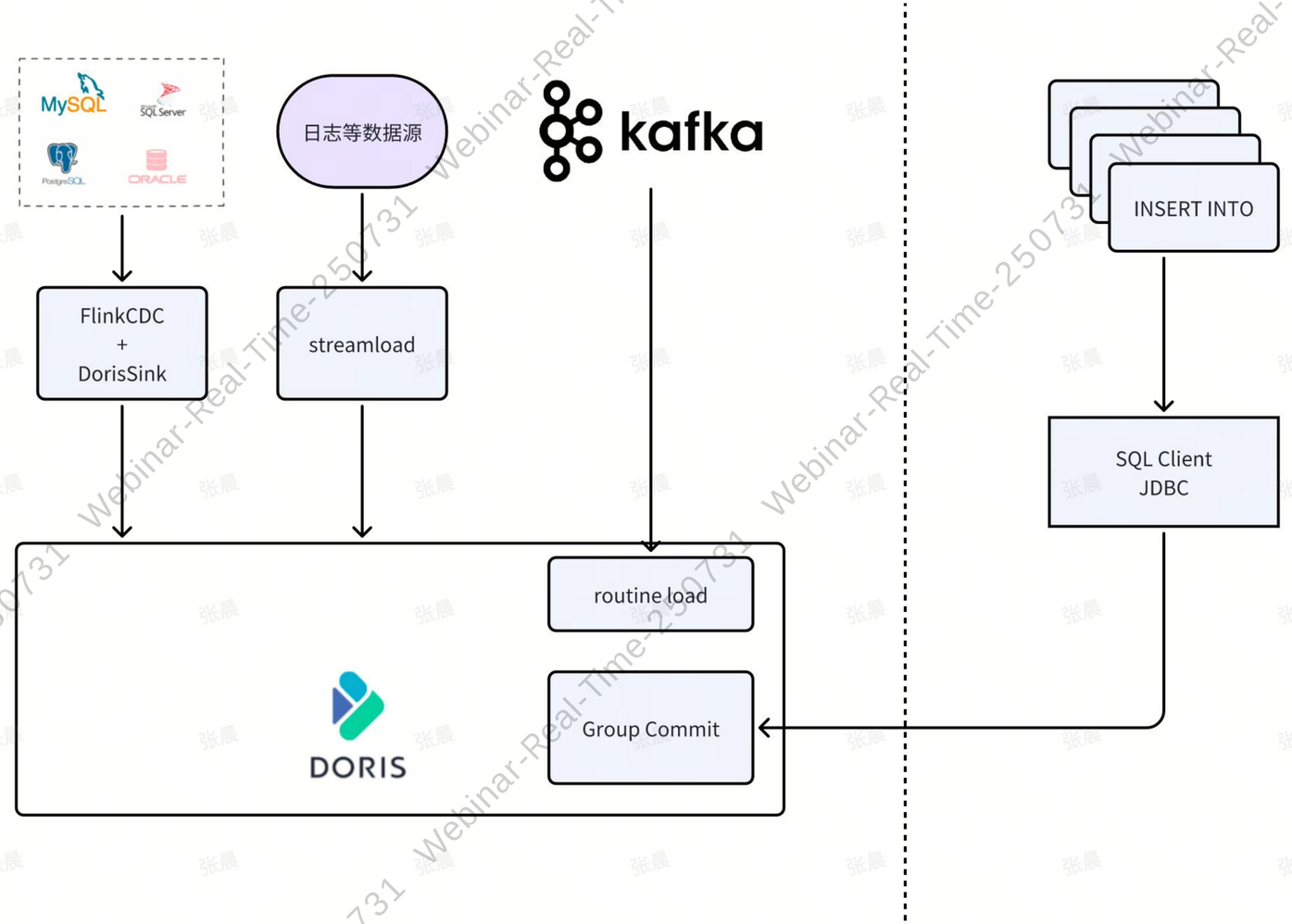


# 高频实时写入

数据实时写入的常见用法

（FlinkCDC/StreamLoad/RoutineLoad）

- 秒级延迟
- 写入即可见
- 高吞吐
- 极速查询




如果想使用 INSERT INTO 来实现高频实时写入，建议搭配 Group Commit 实现

# Apache Doris 部分列更新拼宽表场景

## 数据源

**用户基础信息**
- user_id
- name
- age
- gender
- register_time

**用户行为数据**
- user_id
- login_count
- page_views
- session_duration
- last_login_time

**交易数据**
- user_id
- total_orders
- total_amount
- avg_order_value
- last_order_time

**偏好数据**
- user_id
- categories
- brands
- price_range
- update_time

数据流

**Apache Doris**
部分列更新引擎
- 实时数据同步
- 增量列更新
- 高性能写入

统一宽表

### 用户宽表 (Wide Table)

| 用户ID | 基础信息 | 行为数据 | 交易数据 | 偏好数据 | | 更新时间 |
|---|---|---|---|---|---|---|
| 10001 | 张三,25,男 | 100次,500页 | 50单,¥5000 | 电子,苹果 | | 2024-07-30 |
| 10002 | 李四,30,女 | 80次,300页 | 30单,¥3000 | 服装,nike | | 2024-07-30 |

新增交易

行为更新

偏好更新

## 核心优势

- 只更新变化的列，减少I/O开销
- 支持高并发实时写入
- 保证数据一致性
- 简化ETL流程，降低运维成本

## 应用场景

**用户画像**
实时更新
多维标签

**风控模型**
实时特征
快速响应

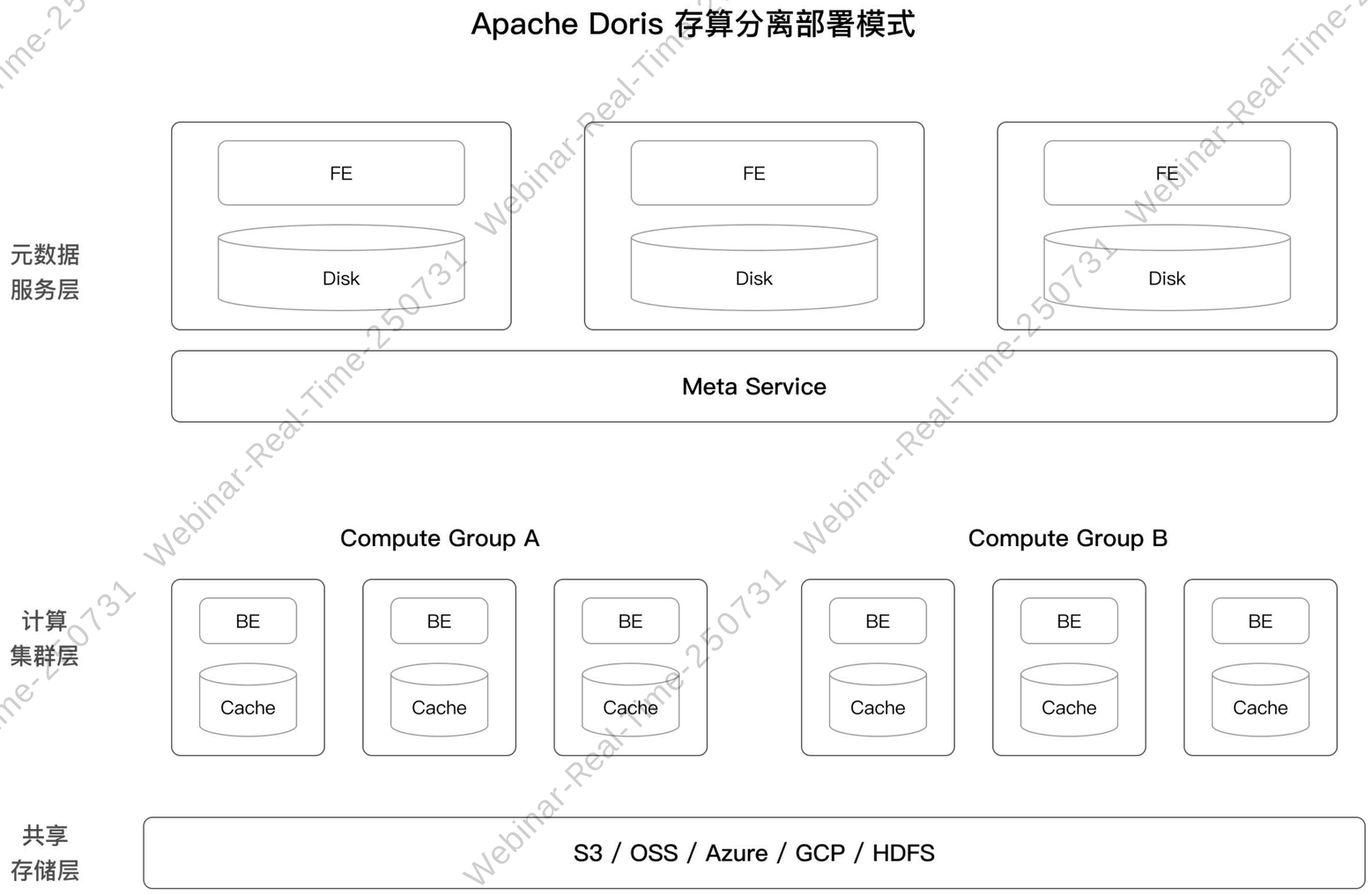
**推荐系统**
个性化
实时计算

# 目录

# 存算分离架构

## Doris 3.0 正式支持存算分离架构

- 引入了 MetaService
- BE 无状态化
- 更好的弹性
- 更好的读写负载隔离
- 更低的成本



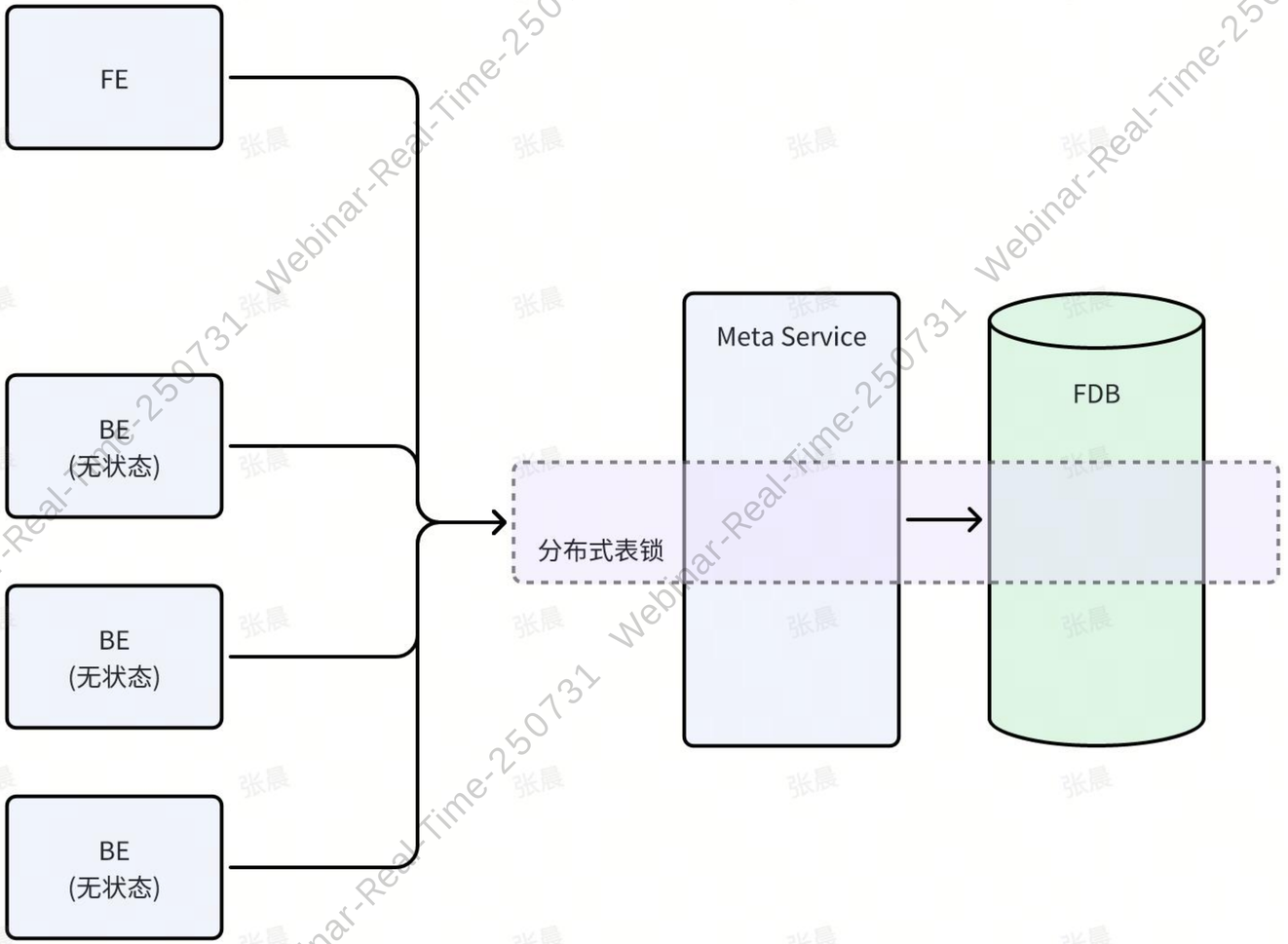
# 存算分离主键模型的挑战

## 主键模型的 MoW 实现对本地的数据状态高度依赖

- 因为要在写入时进行数据去重
- 去重操作需要在 BE 本地完成
- 写入并发，写入与 compaction 并发，写入与 Heavy Schema Change 都会出现写写冲突
- 存算一体架构下 BE 有状态，通过 BE 内存表锁进行互斥来解决冲突

## 存算分离架构下引入了分布式表锁来解决写冲突

- 每个 tablet 的 compaction 在提交时都会请求表锁
- 每个导入在提交时都会去请求表锁
- 因此当 tablet 数量特别多，导入频率也很高时，表锁互斥所带来的额外成本就会非常高



# 性能最佳实践

## 控制单表导入频率

- 不建议超过 60次/s
- 通过攒批，调整并发等方式来控制单表导入频率

## 通过分区分桶，控制需要更新的tablet数量

- 利用时间分区（按周/按天/按小时），减少单次导入更新分区数量
- 分桶数量根据数据量设置在 8-64 之间

## 调整Compaction策略减少表锁冲突

- 通过调整 compaction 策略来适当降低 compaction 的频率

## 升级到3.1版本

- 3.1 版本即将发布
- 针对分布式表锁的实现进行了大幅优化，显著提升了存算分离主键表的导入性能

# Thanks !

DORIS | SELECTDB

# 回放与演讲资料获取

请关注 SelectDB 公众号发送 20250731

## 联系我们

www.selectdb.com

400-092-6099

微信公众号

免费试用

在线咨询

加入社区