**[Apache Doris](http://doris.apache.org/master/zh-CN/)**

http://doris.apache.org/master/zh-CN/getting-started/data-model-rollup.html#rollup

PB(1024TB)级数据查询

**基本概念**

在 Doris 中，数据以表（Table）的形式进行逻辑上的描述。  
一张表包括行（Row）和列（Column）。Row 即用户的一行数据。Column 用于描述一行数据中不同的字段。

Column 可以分为两大类：Key 和 Value。从业务角度看，Key 和 Value 可以分别对应维度列和指标列。

Doris 的数据模型主要分为3类:

* Aggregate 聚合,导入的数据会根据key值唯一,聚合其它value

| **ColumnName** | **Type** | **AggregationType** | **Comment** |
| --- | --- | --- | --- |
| user\_id | LARGEINT |  | 用户id |
| date | DATE |  | 数据灌入日期 |
| city | VARCHAR(20) |  | 用户所在城市 |
| age | SMALLINT |  | 用户年龄 |
| sex | TINYINT |  | 用户性别 |
| last\_visit\_date | DATETIME | REPLACE | 用户最后一次访问时间 |
| cost | BIGINT | SUM | 用户总消费 |
| max\_dwell\_time | INT | MAX | 用户最大停留时间 |
| min\_dwell\_time | INT | MIN | 用户最小停留时间 |

* Uniq 主键模型,key唯一value全部替换

| **ColumnName** | **Type** | **AggregationType** | **Comment** |
| --- | --- | --- | --- |
| user\_id | BIGINT |  | 用户id |
| username | VARCHAR(50) |  | 用户昵称 |
| city | VARCHAR(20) | REPLACE | 用户所在城市 |
| age | SMALLINT | REPLACE | 用户年龄 |
| sex | TINYINT | REPLACE | 用户性别 |
| phone | LARGEINT | REPLACE | 用户电话 |
| address | VARCHAR(500) | REPLACE | 用户住址 |
| register\_time | DATETIME | REPLACE | 用户注册时间 |

* Duplicate 导入数据不变

| **ColumnName** | **Type** | **SortKey** | **Comment** |
| --- | --- | --- | --- |
| timestamp | DATETIME | Yes | 日志时间 |
| type | INT | Yes | 日志类型 |
| error\_code | INT | Yes | 错误码 |
| error\_msg | VARCHAR(1024) | No | 错误详细信息 |
| op\_id | BIGINT | No | 负责人id |
| op\_time | DATETIME | No | 处理时间 |

## ROLLUP 在多维分析中是“上卷”的意思，即将数据按某种指定的粒度进行进一步聚合

在 Base 表之上，我们可以创建任意多个 ROLLUP 表。这些 ROLLUP 的数据是基于 Base 表产生的，并且在物理上是**独立存储**的。

| **ColumnName** | **Type** | **AggregationType** | **Comment** |
| --- | --- | --- | --- |
| user\_id | LARGEINT |  | 用户id |
| date | DATE |  | 数据灌入日期 |
| timestamp | DATETIME |  | 数据灌入时间，精确到秒 |
| city | VARCHAR(20) |  | 用户所在城市 |
| age | SMALLINT |  | 用户年龄 |
| sex | TINYINT |  | 用户性别 |
| last\_visit\_date | DATETIME | REPLACE | 用户最后一次访问时间 |
| cost | BIGINT | SUM | 用户总消费 |
| max\_dwell\_time | INT | MAX | 用户最大停留时间 |
| min\_dwell\_time | INT | MIN | 用户最小停留时间 |

在此基础上，我们创建一个 ROLLUP：

| **ColumnName** |
| --- |
| user\_id |
| cost |

该 ROLLUP 只包含两列：user\_id 和 cost。则创建完成后，该 ROLLUP 中存储的数据如下：

| **user\_id** | **cost** |
| --- | --- |
| 10000 | 35 |
| 10001 | 2 |
| 10002 | 200 |
| 10003 | 30 |
| 10004 | 111 |

可以看到，ROLLUP 中仅保留了每个 user\_id，在 cost 列上的 SUM 的结果。那么当我们进行如下查询时:

SELECT user\_id, sum(cost) FROM table GROUP BY user\_id;

Doris 会自动命中这个 ROLLUP 表，从而只需扫描极少的数据量，即可完成这次聚合查询。

1. 获得不同城市，不同年龄段用户的总消费、最长和最短页面驻留时间

紧接示例1。我们在 Base 表基础之上，再创建一个 ROLLUP：

| **ColumnName** | **Type** | **AggregationType** | **Comment** |
| --- | --- | --- | --- |
| city | VARCHAR(20) |  | 用户所在城市 |
| age | SMALLINT |  | 用户年龄 |
| cost | BIGINT | SUM | 用户总消费 |
| max\_dwell\_time | INT | MAX | 用户最大停留时间 |
| min\_dwell\_time | INT | MIN | 用户最小停留时间 |

则创建完成后，该 ROLLUP 中存储的数据如下：

| **city** | **age** | **cost** | **max\_dwell\_time** | **min\_dwell\_time** |
| --- | --- | --- | --- | --- |
| 北京 | 20 | 0 | 30 | 10 |
| 北京 | 30 | 1 | 2 | 22 |
| 上海 | 20 | 1 | 200 | 5 |
| 广州 | 32 | 0 | 30 | 11 |
| 深圳 | 35 | 0 | 111 | 6 |

当我们进行如下这些查询时:

* SELECT city, age, sum(cost), max(max\_dwell\_time), min(min\_dwell\_time) FROM table GROUP BY city, age;
* SELECT city, sum(cost), max(max\_dwell\_time), min(min\_dwell\_time) FROM table GROUP BY city;
* SELECT city, age, sum(cost), min(min\_dwell\_time) FROM table GROUP BY city, age;

Doris 会自动命中这个 ROLLUP 表

### 前缀索引与 ROLLUP

#### [#](http://doris.apache.org/master/zh-CN/getting-started/data-model-rollup.html#%E5%89%8D%E7%BC%80%E7%B4%A2%E5%BC%95)前缀索引

不同于传统的数据库设计，Doris 不支持在任意列上创建索引。Doris 这类 MPP 架构的 OLAP 数据库，通常都是通过提高并发，来处理大量数据的。  
本质上，Doris 的数据存储在类似 SSTable（Sorted String Table）的数据结构中。该结构是一种有序的数据结构，可以按照指定的列进行排序存储。在这种数据结构上，以排序列作为条件进行查找，会非常的高效。

在 Aggregate、Uniq 和 Duplicate 三种数据模型中。底层的数据存储，是按照各自建表语句中，AGGREGATE KEY、UNIQ KEY 和 DUPLICATE KEY 中指定的列进行排序存储的。

而前缀索引，即在排序的基础上，实现的一种根据给定前缀列，快速查询数据的索引方式。

我们将一行数据的前 **36 个字节** 作为这行数据的前缀索引。当遇到 VARCHAR 类型时，前缀索引会直接截断。我们举例说明：

1. 以下表结构的前缀索引为 user\_id(8Byte) + age(4Bytes) + message(prefix 20 Bytes)。

| **ColumnName** | **Type** |
| --- | --- |
| user\_id | BIGINT |
| age | INT |
| message | VARCHAR(100) |
| max\_dwell\_time | DATETIME |
| min\_dwell\_time | DATETIME |

1. 以下表结构的前缀索引为 user\_name(20 Bytes)。即使没有达到 36 个字节，因为遇到 VARCHAR，所以直接截断，不再往后继续。

| **ColumnName** | **Type** |
| --- | --- |
| user\_name | VARCHAR(20) |
| age | INT |
| message | VARCHAR(100) |
| max\_dwell\_time | DATETIME |
| min\_dwell\_time | DATETIME |

当我们的查询条件，是**前缀索引的前缀**时，可以极大的加快查询速度。比如在第一个例子中，我们执行如下查询：

SELECT \* FROM table WHERE user\_id=1829239 and age=20；

该查询的效率会**远高于**如下查询：

SELECT \* FROM table WHERE age=20；

所以在建表时，**正确的选择列顺序，能够极大地提高查询效率**。

#### [#](http://doris.apache.org/master/zh-CN/getting-started/data-model-rollup.html#rollup-%E8%B0%83%E6%95%B4%E5%89%8D%E7%BC%80%E7%B4%A2%E5%BC%95)ROLLUP 调整前缀索引

因为建表时已经指定了列顺序，所以一个表只有一种前缀索引。这对于使用其他不能命中前缀索引的列作为条件进行的查询来说，效率上可能无法满足需求。因此，我们可以通过创建 ROLLUP 来人为的调整列顺序。举例说明。

Base 表结构如下：

| **ColumnName** | **Type** |
| --- | --- |
| user\_id | BIGINT |
| age | INT |
| message | VARCHAR(100) |
| max\_dwell\_time | DATETIME |
| min\_dwell\_time | DATETIME |

我们可以在此基础上创建一个 ROLLUP 表：

| **ColumnName** | **Type** |
| --- | --- |
| age | INT |
| user\_id | BIGINT |
| message | VARCHAR(100) |
| max\_dwell\_time | DATETIME |
| min\_dwell\_time | DATETIME |

可以看到，ROLLUP 和 Base 表的列完全一样，只是将 user\_id 和 age 的顺序调换了。那么当我们进行如下查询时：

SELECT \* FROM table where age=20 and message LIKE "%error%";

会优先选择 ROLLUP 表，因为 ROLLUP 的前缀索引匹配度更高。

### [#](http://doris.apache.org/master/zh-CN/getting-started/data-model-rollup.html#rollup-%E7%9A%84%E5%87%A0%E7%82%B9%E8%AF%B4%E6%98%8E)ROLLUP 的几点说明

* ROLLUP 最根本的作用是提高某些查询的查询效率（无论是通过聚合来减少数据量，还是修改列顺序以匹配前缀索引）。因此 ROLLUP 的含义已经超出了 “上卷” 的范围。这也是为什么我们在源代码中，将其命名为 Materialized Index（物化索引）的原因。
* ROLLUP 是附属于 Base 表的，可以看做是 Base 表的一种辅助数据结构。用户可以在 Base 表的基础上，创建或删除 ROLLUP，但是不能在查询中显式的指定查询某 ROLLUP。是否命中 ROLLUP 完全由 Doris 系统自动决定。
* ROLLUP 的数据是独立物理存储的。因此，创建的 ROLLUP 越多，占用的磁盘空间也就越大。同时对导入速度也会有影响（导入的ETL阶段会自动产生所有 ROLLUP 的数据），但是不会降低查询效率（只会更好）。
* ROLLUP 的数据更新与 Base 表示完全同步的。用户无需关心这个问题。
* ROLLUP 中列的聚合方式，与 Base 表完全相同。在创建 ROLLUP 无需指定，也不能修改。
* 查询能否命中 ROLLUP 的一个必要条件（非充分条件）是，查询所涉及的**所有列**（包括 select list 和 where 中的查询条件列等）都存在于该 ROLLUP 的列中。否则，查询只能命中 Base 表。
* 某些类型的查询（如 count(\*)）在任何条件下，都无法命中 ROLLUP。具体参见接下来的 **聚合模型的局限性** 一节。
* 可以通过 EXPLAIN your\_sql; 命令获得查询执行计划，在执行计划中，查看是否命中 ROLLUP。
* 可以通过 DESC tbl\_name ALL; 语句显示 Base 表和所有已创建完成的 ROLLUP。

在这篇文档中可以查看 [查询如何命中 Rollup](http://doris.apache.org/master/zh-CN/getting-started/hit-the-rollup)

## [#](http://doris.apache.org/master/zh-CN/getting-started/data-model-rollup.html#%E8%81%9A%E5%90%88%E6%A8%A1%E5%9E%8B%E7%9A%84%E5%B1%80%E9%99%90%E6%80%A7)聚合模型的局限性