高级使用指南 (Palo 2)

1 数据表的创建和导入相关

1.1 合理的表模式

> **Note**
>
> 合理设计表模式，将会极大的提高查询的性能。

Palo 中使用类似 前缀索引 的结构来提高查询性能。

数据在 Palo 内部组织为一个个 Data Block。每个 Data Block 的第一行的前几列会被用作这个 Data Block 的索引，在数据导入时被创建。

考虑到索引大小等因素，Palo 最多使用一行的前 36 个字节作为索引，并且遇到 VARCHAR 类型则会中断并截止。并且 VARCHAR 类型最多只是用字符串的前 20 个字节。

下面举例说明：

- 表1 的 Schema 为:

Field	Type	Agg
k1	int	
k2	bigint	
k3	char(24)	
k4	int	
pv	bigint	SUM

前3列的长度和为(4+8+24=)36，正好 36 个字节，所以只有前 3 列被用作索引。

- 表2 的 Schema 为:

Field	Type	Agg
k1	int	
k2	bigint	
k3	varchar(30)	
k4	int	
pv	bigint	SUM

前2列的长度和为(4+8=)12，还没有达到 36 个字节。而第 3 列类型为 varchar，所以前3列被用作索引。其中 k3 只截取前 20 字节。

- 表3 的 Schema 为:

Field	Type	Agg
k3	varchar(30)	
k1	int	
k2	bigint	
k4	int	

pv	bigint	SUM

该表第一列即为 varchar，因此只有第一列被用作索引。并且直接取第一列的前 20 个字节（因为是 VARCHAR 类型）作为索引。

- 表4 的 Schema 为:

Field	Type	Agg
k1	bigint	
k2	bigint	
k3	datetime	
k4	bigint	
k5	bigint	
pv	bigint	SUM

前4列的长度和为(8+8+8+8=)32，如果加上第5列（8个字节），则会超过 36 个字节。所以只有前 4 列被用作索引。

我们以 表2 和 表3 为例，进一步说明表模式对查询性能的影响。

两个表的 Schema 除了列顺序不同，其他完全一样。

如下查询:

```
SELECT * from tbl WHERE k1 = 12345;
```

在 表2 的查询性能会明显优于 表3。因为在 表2 可以运用到 k1 索引，而 表3 因为只有 k3 作为索引，因此等同于全表扫描。

Note

- 尽量将 频繁使用 、选择度高 的列放在前面。
- 尽量不要将 VARCHAR 类型放在前几列。
- 尽量使用整型作为索引列。

1.2 修改 Schema

使用 ALTER TABLE 命令可以修改表的 Schema，包括如下修改：

- 增加列
- 删除列
- 修改列类型
- 改变列顺序

以下举例说明。

原表 test_tbl 的 Schema 如下:

```
+-------+------------+-------+
| Field | Type       | Agg   |
+-------+------------+-------+
| k1    | int        |       |
+-------+------------+-------+
| k2    | bigint     |       |
+-------+------------+-------+
| k3    | char(24)   |       |
+-------+------------+-------+
| k4    | int        |       |
+-------+------------+-------+
| pv    | bigint     | SUM   |
+-------+------------+-------+
```

我们新增一列 uv，类型为 BIGINT，聚合类型为 SUM，默认值为 0:

```
ALTER TABLE test_tbl ADD COLUMN uv BIGINT SUM DEFAULT '0';
```

提交成功后，可以通过以下命令查看:

```
SHOW ALTER COLUMN;
```

当作业状态为 FINISHED，则表示作业完成。新的 Schema 已生效。

可以使用以下命令取消当前正在执行的作业:

```
CANCEL ALTER COLUMN FROM test_db.test_tbl;
```

Note

请使用 'HELP ALTER TABLE' 查看更多详细信息。

1.3 创建 Rollup

使用 ALTER TABLE 命令可以创建或删除 Rollup。

以下举例说明。

原表 test_tbl 的 Schema 如下:

```
+-------+------------+-------+
| Field | Type       | Agg   |
+-------+------------+-------+
| k1    | int        |       |
+-------+------------+-------+
| k2    | bigint     |       |
+-------+------------+-------+
| k3    | char(24)   |       |
+-------+------------+-------+
| k4    | int        |       |
+-------+------------+-------+
| pv    | bigint     | SUM   |
+-------+------------+-------+
```

我们在此基础上创建一个 Rollup，选择列：k1, k2, pv

```
ALTER TABLE test_tbl ADD ROLLUP(k1, k2, pv);
```

提交成功后，可以通过以下命令查看:

```
SHOW ALTER ROLLUP;
```

当作业状态为 FINISHED，则表示作业完成。

可以使用以下命令取消当前正在执行的作业:

```
CANCEL ALTER ROLLUP FROM test_db.test_tbl;
```

Note

请使用 'HELP ALTER TABLE' 查看更多详细信息。

1.4 Table 与 上卷表（Rollup）的关系

Table 是 Palo 中数据对外唯一的可查询对象，通过 CREATE TABLE 语句创建。

Rollup 可以理解为 Table 的一个物化索引结构。物化 是因为其数据在物理上是独立存储的。而 索引 的意思是，Rollup 存在的目是用于加速在这个 Table 上的某类查询响应。

Rollup 附属于 Table，一个 Table 可以有多个 Rollup。在创建 Table 时会默认生成一个 Base Rollup，该 Rollup 包含 Table 的所有列。后续的其他 Rollup 在此基础上创建。并且通常其他 Rollup 的列数要少于 Base Rollup。

Note

Rollup 通过指定某些常用的维度组合，对指标列进行聚合，能够极大地减少数据量，从而加速查询。

示例: pv_fact 表包含 (k1, k2, k3, k4, v1) 5列，其中前4列为维度列，v1 为指标列，聚合方式为 SUM。

我们在此基础上创建两个 Rollup:

```
ALTER TABLE pv_fact ADD ROLLUP rollup1 (k1, k3, v1);
ALTER TABLE pv_fact ADD ROLLUP rollup2 (k2, k4, v1);
```

当查询模式满足一定条件时，系统将会命中对应的 Rollup 并返回查询结果，从而提高查询的性能。

- 查询命中 rollup1:

```
select k1, sum(v1) from pv_fact group by k1;
```

- 查询命中 base:

```
select k1, k2, sum(v1) from pv_fact group by k1, k2;
```

- 查询命中 rollup2:

```
select k2, sum(v1) from pv_fact group by k2;
```

Note

- 查询时不可直接指定 Rollup。是否命中由系统内部进行判断，无需用户干预。
- 用户可以通过 “EXPLAIN your_sql;” 语句来查看 Sql 的执行计划。
- 导入时只需要按照 Table 的所有列来导入，Rollup 的数据系统会根据其包含的列自动生成。
- ALTER TABLE 命令为异步操作。提交后，使用 SHOW ALTER ROLLUP 查看进度。详见 ‘HELP SHOW ALTER’。

2 数据表的查询

2.1 内存限制

- 为了防止用户的一个查询可能因为消耗内存过大，将集群搞挂，所以查询进行了内存控制，默认控制为落在没有节点上的执行计划分片使用不超过 2GB 内存。
- 用户在使用时，如果发现报 memory limit exceeded 错误，一般是超过内存限制了。
- 遇到内存超限时，用户应该尽量通过优化自己的 sql 语句来解决。
- 如果确切发现 2GB 内存不能满足，建议联系 Palo同学 进行协助。

显示查询内存限制:

```
mysql> show variables like "%MEM_LIMIT%";
+---------------+------------+
| Variable_name | Value      |
+---------------+------------+
| MEM_LIMIT     | 2147483648 |
+---------------+------------+
1 row in set (0.00 sec)
```

执行一个超内存的查询，会得到超内存的报错:

```
mysql> show variables like "%MEM_LIMIT%";
+---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| MEM_LIMIT     | 100000 |
+---------------+--------+
1 row in set (0.00 sec)

mysql> select * from customer order by c_custkey limit 100;
ERROR:
Memory limit exceeded
```

2.2 查询超时

当前默认查询时间设置为最长为 300 秒，如果一个查询在 300 秒内没有完成，则查询会被 Palo 系统 cancel 掉。用户可以通过这个参数来定制自己应用的超时时间，实现类似 wait(timeout) 的阻塞方式。

查看当前超时设置:

```
mysql> show variables like "%QUERY_TIMEOUT%";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| QUERY_TIMEOUT | 300   |
+---------------+-------+
1 row in set (0.00 sec)
```

修改超时时间:

```
mysql> set QUERY_TIMEOUT = 1;
Query OK, 0 rows affected (0.01 sec)
```

查询超时时的报错:

```
mysql> select * from customer order by c_address limit 10;
ERROR:
query execute timeout
```

Note

当前超时的检查间隔为 5 秒，所以小于 5 秒的超时不会太准确。这个未来会将精度提高到秒级别。

2.3 结果行数限制

当前正在实现中.....

2.4 broadcast join 和 shuffle join

系统默认实现 join 的方式，是将小表进行条件过滤后，将其广播到大表所在的各个节点上，形成一个内存 hash 表，然后流式读出大表的数据进行 hash join。但是如果当小表过滤后的数据量无法放入内存的话，此时 join 将无法完成，通常的报错应该是首先造成内存超限。

如果遇到上述情况，建议使用 shuffle 的 join 方式，也被称作 partitioned join。即将小表和大表都按照 join 的 key 进行hash，然后进行分布式的 join。这个对内存的消耗就会分摊到集群的所有计算节点上。

使用 broadcast join（默认）:

```
mysql> select sum(lo_ordtotalprice) from lineorder, customer where lo_custkey = c_custkey and
c_name = "Customer#001048579";
+-----------------------+
| SUM(lo_ordtotalprice) |
+-----------------------+
|            8996548708 |
+-----------------------+
1 row in set (6.84 sec)
```

使用 broadcast join（显式指定）:

```
mysql> select sum(lo_ordtotalprice) from lineorder join [broadcast] customer where lo_custkey =
c_custkey and c_name = "Customer#001048579";
+-----------------------+
| SUM(lo_ordtotalprice) |
+-----------------------+
|            8996548708 |
+-----------------------+
1 row in set (6.51 sec)
```

使用shuffle join:

```
mysql> select sum(lo_ordtotalprice) from lineorder join [shuffle] customer where lo_custkey =
c_custkey and c_name = "Customer#001048579";
+-----------------------+
| SUM(lo_ordtotalprice) |
+-----------------------+
|            8996548708 |
+-----------------------+
1 row in set (22.55 sec)
```

多表join:

```
mysql> select sum(lo_ordtotalprice) from lineorder join [broadcast] customer join [shuffle] part
on (lo_custkey = c_custkey and lo_partkey = p_partkey ) where c_name = "Customer#001048579";
+-----------------------+
| SUM(lo_ordtotalprice) |
+-----------------------+
|            9216548131 |
+-----------------------+
1 row in set (7.04 sec)

mysql> select sum(lo_ordtotalprice) from lineorder join [broadcast] customer join [broadcast]
part  on (lo_custkey = c_custkey and lo_partkey = p_partkey ) where c_name = "Customer#001
+-----------------------+
| SUM(lo_ordtotalprice) |
+-----------------------+
|            9216548131 |
+-----------------------+
1 row in set (9.18 sec)
```

2.5 failover 和 loadbalancing

第一种

自己在应用层代码进行重试和负载均衡。比如发现一个连接挂掉，就自动在其他连接上进行重试。应用层代码重试需要应用自己配置多个palo前端节点地址。

第二种

如果使用 mysql jdbc connector 来连接Palo，可以使用 jdbc 的自动重试机制:

```
jdbc:mysql://[host:port],[host:port].../[database][?propertyName1][=propertyValue1]
[&propertyName2][=propertyValue2]...
```

第三种

应用可以连接到和应用部署到同一机器上的 mysql proxy，通过配置 mysql proxy 的 failover 和 loadbalancing 功能来达到目的。

http://dev.mysql.com/doc/refman/5.6/en/mysql-proxy-using.html

Note

- 无论你是否使用 Palo，还是普通的 mysql，应用都需要对连接进行错误检测，并且出错后要进行重试。
- 第一种：在有 failover 时，需要重试其他节点。
- 第二种和第三种：failover 时，也只需要简单重试，不需要在应用层明确地选择重试节点。

Previous Next