柏睿数据
BorayData

# 数据库性能优化文档

## BorayData RapidsDB Optimization Manual

# RapidsDB

## 全内存分布式分析型数据库

威讯柏睿数据科技（北京）有限公司

*Intelligent Data, Enabling Future!*

北京市朝阳区望京东园七区保利国际广场 19 号楼

电话：+01064700868

RapidsDB Rapids StreamDB Rapids ParallelAI AI work flow AI Rapids APU

# 版权声明

本文档所涉及的软件著作权、版权和知识产权已依法进行了相关注册、登记，由威讯柏睿数据科技（北京）有限公司合法拥有，受《中华人民共和国著作权法》、《计算机软件保护条例》、《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

柏睿数据 BorayData

# 免责声明

本文档包含的柏睿数据公司的版权信息由柏睿数据公司合法拥有，受法律的保护，柏睿数据公司对本文档可能涉及到的非柏睿数据公司的信息不承担任何责任。在法律允许的范围内，您可以查阅，并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经柏睿数据公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将视为侵权，柏睿数据公司具有依法追究其责任的权利。

本文档中包含的信息如有更新，恕不另行通知。您对本文档的任何问题，可直接向威讯柏睿数据科技（北京）有限公司告知或查询。

未经本公司明确授予的任何权利均予保留。

通讯方式如下：

威讯柏睿数据科技（北京）有限公司

北京市朝阳区望京东园七区保利国际广场 19 号楼(T1)503

电话：+010-64700868

网址：http://www.boraydata.cn

# 商标声明


是威讯柏睿数据科技（北京）有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由柏睿数据公司合法拥有，受法律保护。未经柏睿数据公司书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯柏睿数据公司商标权的，柏睿数据公司将依法追究其法律责任。

# 版本声明

本文档涉及产品为柏睿全内存分布式分析型数据库RapidsDBv4.2.3版本以及列存储引擎rpdsqlv2.0.5版本。

| 版本 | 发布日期 | 生效日期 | 作者 | 审核人 | 批准人 |
|---|---|---|---|---|---|
| V1.0 | 2020-10-29 | 2020-10-29 | 王文凯 | 阿尔曼 | 莫明勋 |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

*使用说明：*

*1、 使用时需将“封面”和“修订历史”更新为本文档的封面和修订历史信息，同时删除本说明文字。*

*2、 版本号V1.0对应页脚的版本号001，V2.0对应002以此类推。*

*3、 无审核人和批准人时，则对应单元格填写“/”。*

# 一、 项目背景

## 1.1 需求背景

某央企部门存储业务数据采用的底层数据源为 Oracle 数据库， 随着各种业务数据量递增，业务需求类型多样，业务逻辑越来越复杂，底层的 Oracle 数据库已经不能满足在秒级响应查询结果的需求，不能满足行政人员对结果进行快速处理的要求，在十亿级数据量下，简单的 SQL 查询往往都需要几分钟甚至十几分钟才能查询出结果。

柏睿数据的 RapidsDB 内存数据库是一款 OLAP 型数据库，并且基于全内存分布式结构，其数据读取性能远远超过 Oracle 数据库，能满足在十亿级数据量下，查询结果秒级响应。

## 1.2 业务场景

某央企部门基于 RapidsDB 内存式数据库开发了大数据对比平台，平台的开发目的就是能在在复杂的业务场景下，秒级响应查询结果。目前内存式数据库中共存有 19 个数据集，约 9.37 亿条记录；数据分析模型 27 个，包括 26 个多表关联和 1 个 SQL 分析。对 27 个业务模块进行分类，目前婚姻登记 5 个、社会救助 10 个、脱贫攻坚 11 个、殡葬服务 1 个。其中儿童福利、养老服务、慈善社工、流浪乞讨救助和残疾人福利 5 大类目前还未添加相应的业务查询模块。

主要大表数据的记录数如下：

- ✧ 婚姻 XXX 数据表：6.85 亿
- ✧ 残疾 XXX 数据集：3755 万
- ✧ 大重病 XX 数据集：441 万

- ✧ 逝者 XXX 数据集：2495 万
- ✧ 志愿者 XX 数据集：1.35 亿
- ✧ 建档立卡 XX 数据集：8580 万
- ✧ 低保救助人员信息数据集：4303 万

部分业务场景如下：

- ➢ 统计各地区低保救助(不包含特困救助),与建档立卡重合总人数；
- ➢ 统计各个地区结婚登记、离婚登记补发结婚登记证撤销婚姻登记证补发离婚登记证的男女人数或者总人数；
- ➢ 统计各地区临时救助与建档立卡重合人次、人数；
- ➢ 统计各地区未脱贫、未低保救助的重病、残疾的老年人数量；
- ➢ 统计各地区未脱贫人员中残疾人数,可通过条件筛选查看残疾类别、残疾等级以及老年人、未成年人的情况；
- ➢ 统计各地区建档立卡中未纳入低保救助或特困救助的总人数；

# 二、 设计优化

## 2.1 优化目的

在服务器内存资源有限、数据量十亿级级别、复杂业务逻辑下，查询结果秒级响应还是有难度的，需要从表结构设计、SQL 优化等方面进行设计优化。

## 2.2 理解业务的关键因素

在进行表结构设计之前，必须明白涉及的业务特征，主要注意以下几方面：

- 数据的加载方式：零星插入、批量加载、并发插入、数据更新频率；
- 数据入库速度对比查询是否更加重要；

- 查询是否主要处理与整个数据集相关的一小部分行(0.1%或者更少)还是涉及整个数据表或者大部分子集；
- 哪张表或表的哪些列用来 join；
- 哪些列用来过滤；

## 2.3 行存储和列存储

最重要的考虑因素是在创建表时表的存储形式，RapidsDB 支持两种存储，基于内存的行存储和基于磁盘的列存储。

行存储有利于数据查找和并发更新，保存数据在内存中，特定行的所有列都聚集在内存中，查找某特定行性能较高，数据存储在无锁索引中，拥有高并发性的高性能。行存储可以创建多个索引，以此来满足多个类型的查询，事务类型 OLTP 的工作负载涉及到频率较高更新适合行存储表。

列存储适用于顺序扫描，数据以列存在磁盘上，当执行顺序扫描涉及到少部分列字段时，可拥有数据压缩和较好的性能。因为列存储只有一个单个索引，对比行存储，通用性较差。

一般情况下，如果你的业务尽可能和以下情形匹配，那么列存储是高效的：

- 大量的行会执行顺序扫描(百万行或者表的 5%)；
- 聚合产生在表的部分列上(如少于 10 列)；
- 更新和删除较少，但是此操作会影响大批数据行；

列存需要考虑真实环境，数据更新较少还是大批量的更新，频繁的单行事务性更新不能很好在列存上执行。

### 2.3.1 选择表的存储方式

使用行存储或者是列存储，可以参考以下指南：

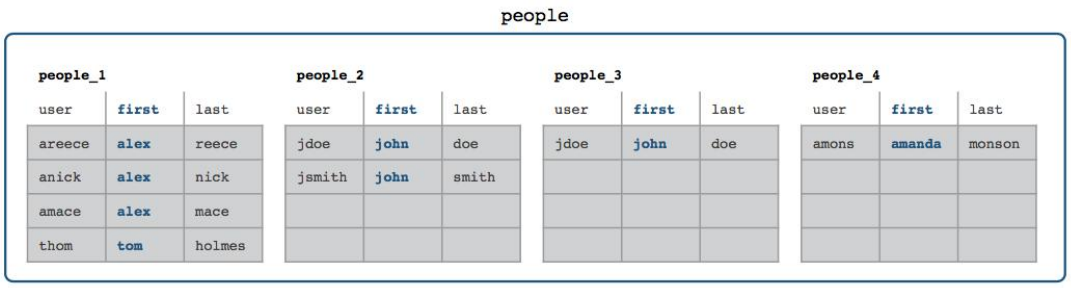
- 是否需要强制的唯一约束？那么使用行存储并设置合适的主键；
- 是否有很多查询在不同列上选择过滤？那么使用行存储并创建多个索引；
- 是否在数据中有查询点？那么使用行存储；
- 是否需要高并发更新和删除，那么使用行存储；
- 聚合大量数据只用来更新或者批量删除？那么使用列存储有较好的性能；

## 2.4 Shard keys

第二点需要考虑的是，当插入数据时，需要考虑表的 shard key 选择。通过分布式集群，存储在叶子节点的各个分区上，shard key 是表的列集合，用来控制表里的数据行如何分区，用来决定分区负责的一系列行。RapidsDB 通过从在 shard key 中的列到分区键 ID 来计算 hash 值，因此，表数据行拥有相同 shard key 的存储在同一个分区。

例如，下列表的 shard key 为 firstname，相同 firstname 值的数据行在同一个分区。

```
create table people (
    username varchar(24),
    firstname varchar(24),
    lastname varchar(24),
    shard key (firstname)
);
```




### 2.4.1 选择 Shard key

Shard key 在表创建时指定，且一旦指定，就不能修改，当创建时，考虑以下两点：

- 数据在分区中均匀分布;
- 列上的分区数据经常过滤或者 join;

首先，由于系统将统一使用其资源，因此均匀分布数据使容量规划变得更加容易，当数据分布不规则或者倾斜，查询性能将面临巨大压力，这回减缓分区查询，多分区查询的运行速度不会超过所涉及的最慢分区速度。

再次，当优化器精准的定位使用了那些分区，查询会显著提升并且使用更少的资源，如果查询过滤与 shard key 匹配，则处理单个分区，对高并发来说，可以大幅度的降级系统资源使用，相似的，多表连接如果拥有相同的 shard key 将会在本机叶子节点执行而不是围绕整个集群分配数据，结果就是显著提供并发性能。

多表连接如果有不同的 shard key 或者分布式连接，基本会很慢而且使用更

多资源。

### 2.4.2 选择行存储的 key

如果要定义索引，在行式表上，会使用 shard key 高效查找指定行。

对行式索引有两种类型的存储：无锁的 skiplists 和无锁的 hash 表。

默认情况下，索引存储为 skiplists,类似于其他数据库中的 B-tree 索引，主要用来存储为有序数据优化的一种数据结构，它在越来越小的有序列表集合中存储行，查询可以通过使用不同大小的列表进行二进制搜索来快速查找数据，并且可以通过迭代最大的列表来快速扫描数据范围。对于多列索引，查询过滤器必须匹配索引列列表的前缀，才能利用索引。

哈希表用来做快速查找，是由哈希函数在相关列上建立索引的存储桶的稀疏数组，查询可以精确匹配行是通过检查存储桶的哈希值，但是不能很快速的扫描表的子集，对于多列索引，查询过滤器必须匹配所有索引列才能利用索引，由于这种限制，不建议使用哈希索引，只有在特定数据集和工作负载上使用。

另外需要考虑的是添加索引会增加额外的开销，每增加一个索引会使用内存存储额外的数据结构，每行平均占用 40bytes，所以会轻微减缓插入数据的速度，这是由于额外的数据结构需要被更新所导致的。

每个行存储表最多可以有一个主键和多个可选的辅助键，扫描主键会快于扫描辅助键。

### 2.4.3 选择列存储的 key

列存储拥有一个索引，集群存储排序键，列存储将使用此排序键的所有行划分为逻辑段，包含多行数据，在一个段内的数据存储在磁盘的段文件上包含了多行的相同字段。这又两种重要功能，一种是单独扫描每一列，能够只扫描执行具

有高度局部性的查询所需的列；另外一种列存储趋向于列压缩，例如：重复和相似的列很容易被压缩在一起。

# 三、 优化命令

在 SQL 编程中，会有一些经常使用且性能较好的语法建议，参照如下：

1. 读取数据时，只选取所需要的列，不要每次都 SELECT *;
2. 慎用 in 和 not in，可以 exists 和 not exists 代替;
3. 尽量避免在 where 字句中对字段进行表达式操作;
4. 尽量避免在 where 字句中进行 null 值判断;
5. 索引的使用应遵循索引使用原则，如索引的使用应尽量满足一下几个原则：
   - 全值匹配
   - 最左前缀
   - 不在索引上做任何操作，会导致对应索引失效
   - 不使用索引中范围条件有变的列
   - 尽量使用覆盖索引
   - 使用不等于或 NOT IN 时会变成全表扫描
   - IS NULL、NOT NULL 无法使用索引，即索引失效
   - LIKE 关键字尽量后匹配，即’a%’，如过无法满足应尽量使用覆盖索引
   - 字符串不加单引号，容易引起索引失效(如整型会隐式类型转换)
   - 少用 OR，OR 并不会使用单值索引，应尽量使用 UNION

# 四、 优化原则

## 4.1 慎用中文 JOIN

多表关联中，应避免使用中文字段值作为多表关联条件；

## 4.2 查找热查询

在正在执行的热查询语句上执行查询分析，可以使用命令 SHOW PLANCACHE。

例如：你可以执行如下查询语句查看平均比较耗时的查询语句，你也可以修改语句以此来过滤数据库，查找指定类型的查询，查询执行时间最长的或者占用内存最高的语句。

```
SELECT DATABASE_NAME, QUERY_TEXT, COMMITS, EXECUTION_TIME,
EXECUTION_TIME/COMMITS AS AVG_TIME_MS, AVERAGE_MEMORY_USE
FROM INFORMATION_SCHEMA.PLANCACHE
WHERE QUERY_TEXT LIKE 'SELECT%'
ORDER BY AVG_TIME_MS DESC;
```

## 4.3 检查查询是否使用索引

一个影响查询性能的重要因素是检查查询语句是否添加合适的索引。可以使用 EXPLAIN 检查查询语句是否使用了索引。下面举例说明索引能够提高查询效率。

- 举例如下：

```
rapids> create table t (a int, b int);
```

- 假定执行如下查询：

```
rapids> select * from t where a=3;
```

- EXPLAIN 显示了当前查询执行中使用了全表扫描，如果 t 表中的部分值等于 3，那全表扫描不是必须的。

```
rapids>  explain select * from t where a=3;
+-----------------------------------------------+
| EXPLAIN                                       |
+-----------------------------------------------+
| Gather partitions:all alias:remote_0          |
| Project [t.a, t.b]                            |
| Filter [t.a = 3]                              |
| TableScan test.t table_type:sharded_rowstore  |
+-----------------------------------------------+
```

- 如果我们增加了索引，可以看到在列 a 上执行了索引范围扫描。

```
rapids>  alter table t add index (a);
Query OK, 0 rows affected (2.60 sec)
Records: 0  Duplicates: 0  Warnings: 0
rapids>  explain select * from t where a=3;
+-------------------------------------------------------------------------------+
| EXPLAIN                                                                       |
+-------------------------------------------------------------------------------+
| Gather partitions:all alias:remote_0                                          |
| Project [t.a, t.b]                                                            |
| IndexRangeScan test.t, KEY a (a) scan:[a = 3] table_type:sharded_rowstore     |
+-------------------------------------------------------------------------------+
3 rows in set (0.01 sec)
```

- 但是，如果同时对列 a 和 b 的过滤，其实没有减少对表的扫描，可以看到，查询依然使用 a=3 进行扫描。

```
rapids> explain select * from t where a=3 and b=4;
+-------------------------------------------------------------------------------+
| EXPLAIN                                                                       |
+-------------------------------------------------------------------------------+
| Gather partitions:all alias:remote_0                                          |
| Project [t.a, t.b]                                                            |
| Filter [t.b = 4]                                                              |
| IndexRangeScan test.t, KEY a (a) scan:[a = 3] table_type:sharded_rowstore     |
+-------------------------------------------------------------------------------+
```

- 增加索引在列 a 和 b 上，可以明显看到查询扫描该表

```
rapids> alter table t add index (a, b);
Query OK, 0 rows affected (1.88 sec)
Records: 0  Duplicates: 0  Warnings: 0
rapids> explain select * from t where a=3 and b=4;
+-----------------------------------------------------------------------------------------+
| EXPLAIN                                                                                 |
+-----------------------------------------------------------------------------------------+
| Gather partitions:all alias:remote_0                                                    |
| Project [t.a, t.b]                                                                      |
| IndexRangeScan test.t, KEY a_2 (a, b) scan:[a = 3 AND b = 4] table_type:sharded_rowstore |
+-----------------------------------------------------------------------------------------+
3 rows in set (0.01 sec)
```

- 然而，在列 b 上的过滤依然没能匹配此索引，因为查询必须匹配索引列列表的前缀才能使用索引。

```
rapids>  explain select * from t where b=4;
+-----------------------------------------------+
| EXPLAIN                                       |
+-----------------------------------------------+
| Gather partitions:all alias:remote_0          |
| Project [t.a, t.b]                            |
| Filter [t.b = 4]                              |
| TableScan test.t table_type:sharded_rowstore  |
+-----------------------------------------------+
```

### 4.3.1 分组和排序索引

- 另一个需要考虑的查询分组和排序的索引是否能提高性能。重新创建表 t，

```
rapids>  create table t (a int, b int);
Query OK, 0 rows affected (0.32 sec)
```

- 考虑如下查询

```
rapids> explain select a, sum(b) from t group by a;
+-----------------------------------------------------------------------------------------+
|EXPLAIN                                                                                  |
+-----------------------------------------------------------------------------------------+
| Project [remote_0.a, `sum(b)`] est_rows:2                                               |
| HashGroupBy [SUM(remote_0.`sum(b)`) AS `sum(b)`] groups:[remote_0.a]                    |
| Gather partitions:all est_rows:2 alias:remote_0                                         |
| Project [t.a, `sum(b)`] est_rows:2                                                      |
| HashGroupBy [SUM(t.b) AS `sum(b)`] groups:[t.a]                                         |
| TableScan test.t table_type:sharded_rowstore est_table_rows:2 est_filtered:2            |
+-----------------------------------------------------------------------------------------+
```

执行以上需要 hash 分组，此处为每个 a 的组构建了一个 hash 表。

- 当在 a 上添加索引后，此处会使用 streaming 分组，因为扫描 a 上的索引会执行索引元素的分组。

```
rapids> alter table t add index (a);
Query OK, 0 rows affected (1.54 sec)
Records: 0  Duplicates: 0  Warnings: 0
rapids> explain select a, sum(b) from t group by a;
+-----------------------------------------------------------------------------------------+
| EXPLAIN                                                                                 |
+-----------------------------------------------------------------------------------------+
| Project [remote_0.a, `sum(b)`] est_rows:2                                               |
| StreamingGroupBy [SUM(remote_0.`sum(b)`) AS `sum(b)`] groups:[remote_0.a]               |
| GatherMerge [t.a] partitions:all est_rows:2 alias:remote_0                              |
| Project [t.a, `sum(b)`] est_rows:2                                                      |
| StreamingGroupBy [SUM(t.b) AS `sum(b)`] groups:[t.a]                                    |
| TableScan test.t, KEY a (a) table_type:sharded_rowstore est_table_rows:2 est_filtered:2 |
+-----------------------------------------------------------------------------------------+
```

- 相似的，对于排序，如没有索引，会执行 sort。

```
rapids> explain select * from t order by b;
+-----------------------------------------------------------+
| EXPLAIN                                                   |
+-----------------------------------------------------------+
| GatherMerge [remote_0.b] partitions:all alias:remote_0    |
| Project [t.a, t.b]                                        |
| Sort [t.b]                                                |
| TableScan test.t table_type:sharded_rowstore              |
+-----------------------------------------------------------+
```

- 如果有索引，会避免使用 sort。

```
rapids> alter table t add index (b);
Query OK, 0 rows affected (1.62 sec)
Records: 0  Duplicates: 0  Warnings: 0
rapids> explain select * from t order by b;
+-----------------------------------------------------------+
| EXPLAIN                                                   |
+-----------------------------------------------------------+
| GatherMerge [remote_0.b] partitions:all alias:remote_0    |
| Project [t.a, t.b]                                        |
| TableScan test.t, KEY b (b) table_type:sharded_rowstore   |
+-----------------------------------------------------------+
```

### 4.3.2 扇区与单分区查询

Rapids 的分布式架构使的你可以最大限度利用众多服务器的 CPU，特别是针对百万行的扫描，需要很高的查询性能。然而，事务查询会选择较少的行，这

不太理想， 然后，最好每个查询只涉及一个分区，看如下表：

```
rapids> create table urls (
        domain varchar(128),
        path varchar(8192),
        time_sec int,
        status_code binary(3),
        ...
        shard key (domain, path, time_sec)
        );
```

- 下面的查询是单个分区的，因为在 shard key 的所有列上都有相同的过滤，所以结果集只匹配在单个分区上，在 explain 中，Gather partitions:single 表明在使用单个分区执行计划。

```
rapids> explain SELECT status_code FROM urls WHERE domain = 'youtube.com'
        AND path = '/watch?v=euh_uqxwk58' AND time_sec = 1;
+-------------------------------------------------------------------------------+
| EXPLAIN                                                                       |
+-------------------------------------------------------------------------------+
| Gather                                                                        |
| partitions:single                                                             |
| Project                                                                       |
| [urls.status_code]                                                            |
|IndexRangeScan test2.urls, SHARD KEY domain (domain, path, time_sec) scan:     |
|[domain = "youtube.com" AND path = "/watch?v=euh_uqxwk58" AND time_sec = 1]    |
+-------------------------------------------------------------------------------+
3 rows in set (0.66 sec)
```

- 但是下面的查询没有在 time_sec 上过滤，查询就扫描了所有分区，执行计划：Gather partitions:all。

```
rapids> explain SELECT status_code FROM urls WHERE  domain = 'youtube.com'
        AND path = '/watch?v=euh_uqxwk58';
+-------------------------------------------------------------------------------+
| EXPLAIN                                                                       |
+-------------------------------------------------------------------------------+
| Project[urls.status_code]                                                     |
| Gather partitions:all                                                         |
| Project [urls.status_code]                                                    |
| IndexRangeScan test2.urls, SHARD KEY domain (domain, path, time_sec) scan:    |
| [domain = "youtube.com" AND path = "/watch?v=euh_uqxwk58"]                    |
+-------------------------------------------------------------------------------+
```

为了弥补，我们可以将表 urls 切分，很容易重新查询，也会产生单个分区，但是这样可能是的一些 domain 占据大量页存因此产生数据倾斜，选择 shard key 经常需要平衡，我们尽可能使用最少的限制确保数据不会倾斜，本例中的建议是

使用 shard key(domain, path)。

### 4.3.3 分布式连接

- 查询架构允许你可以在任何表上运行任意 SQL 查询，而不考虑数据分发，

  通常可以通过优化模式来提高性能，使得在查询执行期间数据移动最小化。

```
rapids> CREATE TABLE lineitem(
        l_orderkey int not null,
        l_linenumber int not null,
        ...
        primary key(l_orderkey, l_linenumber)
        );
        CREATE TABLE orders(
        o_orderkey int not null,
        ...
        primary key(o_orderkey)
        );
```

- 执行计划

```
rapids> explain SELECT Count(*) FROM LINEITEM JOIN ORDERS ON o_orderkey = l_orderkey ;
+-----------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                               |
+-----------------------------------------------------------------------------------------------------------------------+
| Project [CAST(COALESCE($0,0) AS SIGNED) AS `Count(*)`] est_rows:1                                                     |
| Aggregate [SUM(remote_0.`Count(*)`) AS $0]                                                                            |
| Gather partitions:all est_rows:1 alias:remote_0                                                                       |
| Project [`Count(*)`] est_rows:1 est_select_cost:12,000,000                                                            |
| Aggregate [COUNT(*) AS `Count(*)`]                                                                                    |
| HashJoin [r1.o_orderkey = LINEITEM.L_ORDERKEY]                                                                        |
| HashTableBuild alias:LINEITEM                                                                                         |
| Project [LINEITEM_1.L_ORDERKEY] est_rows:6,001,215                                                                    |
| TableScan tpch.LINEITEM AS LINEITEM_1, PRIMARY KEY (L_ORDERKEY, L_LINENUMBER)                                         |
|          table_type:sharded_rowstore est_table_rows:6,0                                                               |
| TableScan r1 storage:list stream:yes table_type:reference est_table_rows:1,500,000 est_filtered:1,500,000             |
| Broadcast [ORDERS.O_ORDERKEY AS o_orderkey] AS r1 distribution:direct est_rows:1,500,000                              |
| TableScan tpch.ORDERS, PRIMARY KEY (O_ORDERKEY) table_type:sharded_rowstore est_table_rows:1,500,000                  |
|          est_filtered:1,500,000                                                                                       |
+-----------------------------------------------------------------------------------------------------------------------+
```

- 添加 shard key

```
rapids> explain SELECT Count(*) FROM LINEITEM JOIN ORDERS ON o_orderkey = l_orderkey ;
+-----------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                         |
+-----------------------------------------------------------------------------------------------------------------+
| Project [CAST(COALESCE($0,0) AS SIGNED) AS `Count(*)`] est_rows:1                                               |
| Aggregate [SUM(remote_0.`Count(*)`) AS $0]                                                                      |
| Gather partitions:all est_rows:1 alias:remote_0                                                                 |
| Project [`Count(*)`] est_rows:1                                                                                 |
| Aggregate [COUNT(*) AS `Count(*)`]                                                                              |
| NestedLoopJoin                                                                                                  |
| IndexRangeScan tpch.LINEITEM, PRIMARY KEY (L_ORDERKEY, L_LINENUMBER) scan:[L_ORDERKEY = ORDERS.O_ORDERKEY]      |
|      table_type:sharded_rowstore est_table_rows:6,001,215 est_filtered:6,001,215                                |
| TableScan tpch.ORDERS, PRIMARY KEY (O_ORDERKEY) table_type:sharded_rowstore est_table_rows:1,500,000            |
|       est_filtered:1,500,000                                                                                    |
+-----------------------------------------------------------------------------------------------------------------+
```

柏睿数据 BorayData

### 4.3.4 聚合上的连接

- 考虑如下情况

```
rapids> CREATE TABLE customer(
        c_custkey int not null,
        c_acctbal decimal(15,2) not null,
        primary key(c_custkey)
        );
        CREATE TABLE orders(
        o_orderkey int not null,
        o_custkey int not null,
        o_orderstatus varchar(20) not null,
        primary key(o_orderkey),
        key(o_custkey)
        );
rapids> select count(*) from orders;
        +----------+
        | count(*) |
        +----------+
        |   429786 |
        +----------+
rapids> select count(*) from orders where o_orderstatus = 'open';
        +----------+
        | count(*) |
        +----------+
        |     1000 |
        +----------+
rapids> select count(*) from customer;
        +----------+
        | count(*) |
        +----------+
        |  1014726 |
        +----------+
rapids> select count(*) from customer where c_acctbal > 100.0;
        +----------+
        | count(*) |
        +----------+
        |      988 |
        +----------+
```

- 说明表 customer 和表 orders 相当大，当我们过滤订单状态是 open 和账户余额大于 100 的数据时，能够匹配的行很少。因此，当我们使用过滤器连接 orders 和 customer 时，可以使用连接在聚合器上。解释器显示：各自有单独的 Gather 操作在 orders 表上和 customer 表上，还有个 HashJoin 在 Gather 之上。

```
rapids> explain SELECT o_orderkey FROM customer JOIN orders where c_acctbal > 100.0
        AND o_orderstatus = 'open' AND o_custkey = c_custkey;
+-----------------------------------------------------------------+
| EXPLAIN                                                         |
+-----------------------------------------------------------------+
| Project [orders.o_orderkey]                                     |
| HashJoin [orders.o_custkey = customer.c_custkey]                |
| |---TempTable                                                   |
| |   Gather partitions:all                                       |
| |   Project [orders_0.o_orderkey, orders_0.o_custkey]           |
| |   Filter [orders_0.o_orderstatus = "open"]                    |
| |   TableScan test3.orders AS orders_0, PRIMARY KEY (o_orderkey)|
| TableScan 0tmp AS customer storage:list stream:yes              |
| TempTable                                                       |
| Gather partitions:all                                           |
| Project [customer_0.c_custkey]                                  |
| Filter [customer_0.c_acctbal > 100.0]                           |
| TableScan test3.customer AS customer_0, PRIMARY KEY (c_custkey) |
+-----------------------------------------------------------------+
```

- 默认情况下，rapids 会执行此优化操作当每个 Gather 少于 120000 行时，这种限制可以被改变通过修改变量 max_subselect_aggregator_rowcount，我们也可以通过手动禁用此优化通过使用 hint 的 leaf_pushdown，通过强制使用 leaf_pushdown 优化器来执行尽可能多的任务在叶子节点。

```
rapids> explain SELECT WITH(LEAF_PUSHDOWN=TRUE) o_orderkey FROM customer JOIN orders
        where  c_acctbal > 100.0 AND o_orderstatus = 'open' AND o_custkey = c_custkey;
+-----------------------------------------------------------------------------------------+
|EXPLAIN                                                                                  |
+-----------------------------------------------------------------------------------------+
|Project [r0.o_orderkey]                                                                  |
| Gather partitions:all est_rows:11                                                       |
| Project [r0.o_orderkey] est_rows:11 est_select_cost:1955                                |
| Filter [customer.c_acctbal > 100.0]                                                     |
| NestedLoopJoin                                                                          |
| |---IndexSeek test3.customer, PRIMARY KEY (c_custkey) scan:[c_custkey = r0.o_custkey]   |
      est_table_rows:1013436 est_filtered:1092                                           |
| TableScan r0 storage:list stream:no                                                     |
| Repartition [orders.o_orderkey, orders.o_custkey] AS r0 shard_key:[o_custkey] est_rows: |
| Filter [orders.o_orderstatus = "open"]                                                  |
| TableScan test3.orders, PRIMARY KEY (o_orderkey) est_table_rows:92628 est_filtered:972  |
+-----------------------------------------------------------------------------------------+
rapids> set max_subselect_aggregator_rowcount=500;
rapids> explain SELECT o_orderkey FROM  customer JOIN orders where c_acctbal > 100.0
        AND o_orderstatus = 'open' AND o_custkey = c_custkey;
+-----------------------------------------------------------------------------------------+
| EXPLAIN                                                                                 |
+-----------------------------------------------------------------------------------------+
| Project [r0.o_orderkey]                                                                 |
| Gather partitions:all est_rows:11                                                       |
| Project [r0.o_orderkey] est_rows:11 est_select_cost:1955                                |
| Filter [customer.c_acctbal > 100.0]                                                     |
| NestedLoopJoin                                                                          |
| |---IndexSeek test3.customer, PRIMARY KEY (c_custkey) scan:[c_custkey = r0.o_custkey]   |
      est_table_rows:1013436 est_filtered:1092                                           |
| TableScan r0 storage:list stream:no                                                     |
| Repartition [orders.o_orderkey, orders.o_custkey] AS r0 shard_key:[o_custkey] est_rows: |
| Filter [orders.o_orderstatus = "open"]                                                  |
| TableScan test3.orders, PRIMARY KEY (o_orderkey) est_table_rows:92628 est_filtered:972  |
+-----------------------------------------------------------------------------------------+
```

### 4.3.5 执行 ANALYZE

- ANALYZE 命令统计表的数据信息以此来方便精准优化查询，对优化复杂分析查询特别重要，最好是对所有的表执行此命令。

```
rapids> ANALYZE TABLE table_name;
```

# 五、 优化示例

## 5.1 建表（以婚姻表为例）

创建婚姻表的类型为列式表，SHARD KEY 为身份证号字段 ID_NUM，索引 ey 为地区字段 REGN。

```
CREATE TABLE `DWS_CIVIL_MARG_PN_DI` (
`PN_CD` varchar(32) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '主键标识' ,
`NAME` varchar(240) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '姓名' ,
`ID_TP` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '身份证件类型' ,
`ID_NUM` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '身份证件号码' ,
`NATION` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '国籍' ,
`SEX` varchar(20) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '性别' ,
`SEX_CD` varchar(2) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '性别代码' ,
`AGE` decimal(3,0) DEFAULT NULL COMMENT '年龄' ,
`ETHNIC` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '民族' ,
`REGN` varchar(100) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '地区' ,
`CITY` varchar(100) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '城市' ,
`HOUS_REG_TP` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '户籍类别' ,
`POLIT_STS` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '政治面貌' ,
`MARG_STS` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '婚姻状况' ,
`ASOC_CD` varchar(60) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '职业分类' ,
`DEGR_M` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '学历' ,
`BUS_TP` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '业务类型' ,
`BUS_CD` varchar(10) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '业务类型代码' ,
`LH_REAS` varchar(200) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '离婚原因' ,
`BL_REAS_CD` varchar(30) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '补领原因' ,
`SXXW_TY` varchar(100) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '失信行为' ,
`REGN_CD` varchar(100) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '行政区划代码' ,
`BATCH` varchar(9) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '批次' ,
`SOURCE_NUM` decimal(16,0) DEFAULT NULL COMMENT '业务编号' ,
`OP_DT` varchar(100) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '登记时间' ,
`DEPT_CD` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL COMMENT '登记部门' ,
UNIQUE KEY `pk` (`REGN`) UNENFORCED RELY ,
SHARD  KEY `ID_NUM` (`ID_NUM`) USING CLUSTERED COLUMNSTORE
) COMMENT '婚姻登记当事人基本信息数据集' AUTOSTATS_ENABLED=TRUE;
```

考虑到关于婚姻表查询的 sql 语句的 where 子句中经常出现对性别字段 SEX 和业务类型字段 BUS_TP，而这两个字段的值在表里均为字符型数值，而过滤条件中出现汉字，会大大延长查询时间，所以在创建婚姻表时，增加了对应的两个冗余字段 SEX_CD 和 BUS_CD，用数值型值代替字符型的值。

## 5.2 执行 ANALYZE

建好表并导入完数据后，需要执行 ANALYZE 命令来统计表的数据信息以此来方便精准优化查询，对优化复杂分析查询特别重要，最好是对所有的表执行此命令：

ANALYZE TABLE DWS_CIVIL_MARG_PN_DI;

## 5.3 SQL 优化

见第二大章：优化命令。

## 5.4 SQL 优化记录

其中各个 SQL 所涉及到的主要大表数据的记录数如下：

- ✧ 婚姻数据表：6.85 亿
- ✧ 残疾信息数据集：3755 万
- ✧ 大重病信息数据集：441 万
- ✧ 逝者信息数据集：2495 万
- ✧ 志愿者信息数据集：1.35 亿
- ✧ 建档立卡数据集：8580 万
- ✧ 低保救助人员信息数据集：4303 万
- ✧ ……

主要的测试 SQL 如下：

（1）

```
select REGN,AGE,SEX,count(*) from DWS_CIVIL_MARG_PN_DI t left join XZQH_LX a on a.dm =
substring(t.REGN_CD,1,2) where t.REGN != a.mc and t.BUS_TP in ('结婚登记','离婚登记','补发结婚证',
'补发离婚证','撤销结婚登记') and exists (select 1 from XZQH_LX a where substring(t.REGN_CD,1,2)=a.dm)
group by REGN,AGE,SEX  order by t.REGN limit 500;
```

（2）

```
select b.REGN 按省统计, count(distinct(b.ID_NUM)) 人数统计 from ( select a0.ID_NUM,
a0.NAME,a0.SEX,IF(CHAR_LENGTH(a0.ID_NUM)=18,(YEAR(NOW())-SUBSTRING(a0.ID_NUM,7,4)) -
(SUBSTRING(a0.ID_NUM,11,4)-DATE_FORMAT(NOW(),'%M%D')>0),0) AS AGE,a0.REGN,
a1.DISAB_NUM,a1.DISAB_CLASS,a1.DISAB_LV,a2.SUBS_TP,a2.SUBS_AMOUNT,a2.GRANT_STS
from DWS_EXTNL_CPAD_TPBWD_QF a0 inner join DWS_EXTNL_CDPF_DISAB_QF a1 on
a0.ID_NUM=a1.ID_NUM left join DWS_CIVIL_DISAB_PN_DI a2 on a1.ID_NUM=a2.ID_NUM
where 1=1  and a1.ID_NUM not in (select a2.ID_NUM from DWS_CIVIL_DISAB_PN_DI a2) and
a0.SEX in ('男性')  ) b group by b.REGN having b.REGN is not null  limit 500;
```

（3）

```
select SOURCE_NUM,OP_DT,DEPT_CD,REGN_CODE,count(1) from (SELECT
T.SOURCE_NUM,T.OP_DT,T.DEPT_CD,T.REGN_CD,left(T.REGN_CD,2) AS REGN_CODE
FROM DWS_CIVIL_MARG_PN_DI T WHERE LEFT(T.ID_NUM,2) <> LEFT(T.REGN_CD,2)
GROUP BY SOURCE_NUM,DEPT_CD,OP_DT) a group by REGN_CODE;
```

（4）

```
WITH AAA AS(SELECT A.SOURCE_NUM,A.DEPT_CD,A.OP_DT,A.ID_NUM,A.`NAME`,
A.SEX,A.SEX_CD,A.REGN_CD FROM DWS_CIVIL_MARG_PN_DI_TEST1 A LEFT JOIN
DWS_EXTNL_CDPF_DISAB_QF_TEST B ON B.ID_NUM = A.ID_NUM LEFT JOIN
DWS_EXTNL_NHC_SROS_ILNS_QF C ON C.ID_NUM = A.ID_NUM LEFT JOIN
DWS_CIVIL_FUNRL_USER_RI D ON D.ID_NUM = A.ID_NUM WHERE B.DISAB_LV_CD = 1
OR B.DISAB_LV_CD = 2 OR (B.DISAB_LV_CD = 3 AND B.DISAB_CLASS_CD = 5 OR
B.DISAB_CLASS_CD = 6))SELECT A.ID_NUM,A.`NAME`,A.SEX,B.ID_NUM,B.`NAME`,B.SEX,
A.SOURCE_NUM,A.DEPT_CD,A.OP_DT FROM AAA A JOIN AAA B ON A.SOURCE_NUM =
B.SOURCE_NUM AND A.REGN_CD = B.REGN_CD AND A.OP_DT = B.OP_DT WHERE
A.SEX_CD = '1' AND B.SEX_CD = '2';
```

（5）

```
select b.REGN 按省统计, count(distinct(b.ID_NUM)) 低保建档立卡重合人数 from (
select a0.NAME,a0.ID_NUM,a0.SEX,a0.AGE,a0.NATION,a0.REGN,a0.CITY,a0.HOUS_REG_TP,
a0.WORK_ABIL,a0.DISAB_CLASS,a0.DISAB_LV,a0.HEALTH_STS,a0.DEGR,a0.STUDY_STS,
a0.WORK_STS,a0.MARG_STS,a0.SERIOUS_ILLNESS_TP,a0.IS_LOCAL_PER,a0.IS_SEC_OBJ,
a0.STATE,a0.ASST_CLASS,a0.ASST_STS,a0.POV_REAS,a1.IS_OUT_POV from
DWS_CIVIL_SOCLASST_DB_PN_MI a0 inner join DWS_EXTNL_CPAD_JDLK_QF a1 on
a0.ID_NUM=a1.ID_NUM  where 1=1  ) b group by b.REGN;
```

测试记录如下：

| SQL 序号 | 优化前 | 优化后 | 优化策略 |
|---|---|---|---|
| 1 | 1min50s | 8.5s | 1、表结构优化，适当新增特定字段索引<br>2、字典字段增加冗余 CODE<br>3、修改多表 JOIN 以中文作为关联条件的情况，改为字典字段 CODE<br>4、修改 SQL 调用函数，如获取日期字段年份，优化前以截取字符串的形式实现，优化后改为 YEAR()函数<br>5、多表关联时调整 ON 条件的先后顺序，使小表在前，大表在后 |
| 2 | 9min28s | 20.7s | |
| 3 | 5min6s | 2.3s | |
| 4 | 23.5s | 9.3s | |
| 5 | 76s | 17s | |
| 6 | 8min6s | 5.5s | |
| 7 | …… | …… | |