Doris使用规范（最佳实践）

> 💡 说明：这里一些原则非强制，但是遵守会取得更好的效果

第一部分：字符集规范

【强制】数据库字符集指定utf-8，并且只支持utf-8。

命令规范

1. 【建议】库名统一使用小写方式，中间用下划线（_）分割，长度62字节内
2. 【建议】表名称大小写敏感，统一使用小写方式，中间用下划线（_）分割，长度64字节内

第二部分：建表规范

1. 【强制】确保每个tablet大小为1-3G之间。举例：假设表内单分区数据量在100G，按天分区,bucket数量100个。
2. 【强制】 5 亿以上的数据必须设置分区分桶策略
    a. 维度表：缓慢增长的，可以使用单分区，在分桶策略上使用常用查询条件（这个字段数据分步相对均衡）分桶，
        i. 100M以内：1 buckets
        ii. 100M-1G： 3-5 个 Buckets
        iii. 大于1G-3G： 5-7个 buckets
        iv. 3-5G： 7-10 个 buckets
    b. 事实表
        i. 没有办法分区的，数据又缓慢增长的：单个tablet数据量保持在1-3G；比如5亿数据大小在20G，bucket数量给20个
        ii. 没有办法分区的，数据又较快增长的，没办法按照时间动态分区，可以适当放大一下你的bucket数量，按照你的数据保存周期（180天）数据总量，来估算你的bucket数量应该是多少，建议还是单个bucket大小在1-3G。
            1. 避免数据倾斜的问题
                a. 一个是对分桶字段进行加盐处理，业务上查询的时候也是要同样的加盐策略，这样能利用到分桶数据剪裁能力
                b. 另外一个是数据随机分桶，这种缺点是没办法利用数据分桶剪裁能力，数据分布会很均匀

3. 【建议】 1000w-2 亿以内数据为了方便可以不设置分区，直接用分桶策略。（不设置其实Doris内部会有个默认分区）
    a. 参考上面第二点
4. 【强制】 2000kw 以内数据禁止使用动态分区（动态分区会自动创建分区，而小表用户客户关注不到，会创建出大量不使用分区分桶）
    a. 参考上面第二点
5. 【强制】对于有大量历史分区数据，但是历史数据比较少，或者不均衡，或者查询概率的情况，使用如下方式将数据放在特殊分区。
    a. 对于历史数据，如果数据量比较小我们可以创建历史分区（比如年分区，月分区），将所有历史数据放到对应分区里
    b. 创建历史分区方式 `FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR`

```
(
 PARTITION p00010101_1899 VALUES [('0001-01-01'), ('1900-01-01')),
 PARTITION p19000101 VALUES [('1900-01-01'), ('1900-01-02')),
...
 PARTITION p19000104_1999 VALUES [('1900-01-04'), ('2000-01-01')),
 FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR,
 PARTITION p30001231 VALUES [('3000-12-31'), ('3001-01-01')),
 PARTITION p99991231 VALUES [('9999-12-31'), (MAXVALUE))
)
```

6. 【强制】如果分桶字段存在30%以上的数据倾斜，则禁止使用Hash分桶策略，改使用random分桶策略
    a. 参考上面第二点事实表部分
7. 【建议】前缀索引的第一个字段一定是最长查询的字段，并且需要是高基字段。这里面选取分区分桶外最长查询且高基数的列
    a. 前缀索引（36位）：第一个字段查询性能最好，前缀索引碰见varchar类型的字段，会自动截断前20个字符
        i. Int（4）+ Int（4） + varchar(50)，前缀索引长度只有28
        ii. Int（4） + varchar(50) + Int（4），前缀索引长度只有24
        iii. varchar(10) + varchar(50)，前缀索引长度只有30
    b. 最常用的查询字段如果能放到前缀索引里尽可能放到前前缀索引里，如果不能，可以放到分桶字段里
        i. 分桶字段注意事项：这个一般是数据分布比较均衡的，也是经常使用的字段，最好是高基数字段

good case： UNIQUE KEY(`user_id` , `age`) user_id最长被查询，且数据分布比较散

bad case： UNIQUE KEY(`age` , `user_id`) age是低基数列，且可能存在数据倾斜

8. 【强制】表的副本数必须为3
9. 【建议】前缀索引中的字段长度尽可能明确，因为Doris只有前36个字节能走前缀索引
10. 【强制】除了UNIQUE KEY和aggregate key要构建key的情况，否则不要基数（例如user_type）小于50的字段建立任何索引。因为Doris内置了字典类型优化。
    a. 已经有了低基数优化了
    b. Unique Key 是aggregate key 的一个特例，当aggregate key 的key 保持唯一其实就是Unqiue key 模型
11. 【强制】BloomFilter索引必须在查询条件是in或者=，并且是高基（5000以上）列上构建。
    a. 数据基数在一半左右
    b. 类似身份证号这种基数特别高并且查询是等值（=）查询，使用Bitmap索引能极大加速
    c. Bloomfilter 使用场景：
        i. 首先BloomFilter适用于非前缀过滤。
        ii. 查询会根据该列高频过滤，而且查询条件大多是 in 和 = 过滤。
        iii. 不同于Bitmap, BloomFilter适用于高基数列。比如UserID。因为如果创建在低基数的列上，比如 “性别” 列，则每个Block几乎都会包含所有取值，导致BloomFilter索引失去意义。
12. 【强制】bitmap索引必须在一定基数范围内构建，太高或者太低的基数都不合适
    a. 这种索引更多的适合正交查询
    b. 适用场景：

       适用于低基数的列上，建议在100到100,000之间，如：职业、地市等。 重复度过高则对比其他类型索引没有明显优势；重复度过低，则空间效率和性能会大大降低。 特定类型的查询例如count、or、and 等逻辑操作因为只需要进行位运算
    c. Bitmap 索引支持类型：
        - bitmap 索引支持的数据类型如下:
            - `TINYINT`
            - `SMALLINT`
            - `INT`
            - `BIGINT`
            - `CHAR`
            - `VARCHAR`

- `DATE`
- `DATETIME`
- `LARGEINT`
- `DECIMAL`
- `BOOL`

13. 【强制】亿级别以上数据，如果有模糊匹配，使用倒排索引或者是 NGram Bloomfilter
14. 【建议】如果某个范围数据在分区分桶和前缀索引中都不好设计，可以考虑引入倒排索引加速。
15. 【强制】单表物化视图不能超过6个
    a. 单笔物化视图是实时构建
    b. 在unique 模型上物化视图只能起到 Key 重新排序的作用，不能做数据的聚合，因为Unqiue模型的聚合模型是replace
16. 【建议】建议使用JSON数据类型代替字符串类型存放JSON数据的使用方式

第三部分：数据变更规范

1. 【强制】应用程序不可以直接使用delete后者update语句变更数据，使用CDC的upsert方式来实现。
    a. 低频操作上使用，比如 Update 几分钟更新一次
    b. 如果使用 Delete 一定带上分区条件
2. 【强制】DBA执行delete后者update语句时必须带分区条件
3. 【强制】禁止使用INSERT INTO tbl1 VALUES ("1"), ("a");这种方式写入数据。
4. 【建议】特殊大的ETL操作，简单单独在Session中设置超时时间
    a. `SELECT /*+ SET_VAR(query_timeout = 1*/ sleep(3);` 类似这样通过Hint方式去设置Session 会话变量，不要设置全局的系统变量

第四部分：数据查询规范

1. 【强制】in 中条件超过 2000 后，必须修改为子查询
2. 【强制】禁止使用REST API（Statement Execution Action）执行大量SQL查询，改接口仅仅用于集群维护。
3. 【建议】一次insert into select 数据超过1亿条后，建议拆分为多个insert into select语句执行，分成多个批次来执行。
    a. 如果真的是要这样执行，在集群资源相对空闲的时候可以通过调整并发度来加快的数据导入速度

```
1 set parallel_fragment_exec_instance_num = 8 或者 16 建议是你CPU内核的一半
```

```
2 insert into new_tbl select * from old_tbl
```

4. 【强制】query查询条件返回结果在5w条以上，使用JDBC Catalog或者OUTFILE方式导出。不然大量FE上数据传输将占用FE资源，影响集群稳定性
   a. 如果你是交互式查询，建议使用分页方式（offset limit），分页要加Order by
   b. 如果是数据导出提供给第三方使用，建议使用 outfile 或者 export 方式
5. 【建议】query查询如果有大量的数据传输需求，建议部署observer节点并在该该节点执行查询（私有化部署）
   a. 建议的方式是 1 FE(Follower) + 多个 OBserver（FE）方式，读写分析，所有的写连接Follower，所有的读连接Observer
6. 【建议】尽量不要使用OR 作为 JOIN条件
7. 【建议】大量数据排序（5亿以上）后返回部分数据，建议先减少数据范围在执行排序，否则大量排序会影响性能。

```
1 select /*+ set_var(parallel_fragment_exec_instance_num=8) */  krr.sequence_id
  , krr.event_occur_time_date , krr.event_occur_time , krr.partner_code ,
  krr.app_name , krr.event_id , krr.event_type , krr.success ,
  krr.final_decision , krr.final_score, krr.account_login , krr.account_mobile ,
  krr.dev_success , krr.policy_name , krr.policy_version , krr.sub_reason_codes
  , krr.policy_uuid , krr.reason_code , krr.score , krr.event_real_time ,
  krr.produce_time , krr.account_email from kunpeng_risk_record krr where
  krr.event_occur_time_date between '2023-10-01 00:00:00' and '2023-10-25
  23:59:59' and krr.partner_code = 'liveme' order by krr.sequence_id desc limit
  20;
```

例如将 `from table order by datatime desc limit 10` 优化为`from table where  datatime='2023-10-20' order by datatime desc limit 10`

8. 【强制】2个以上大于3亿的表 JOIN 使用 Colocate JOIN

   Colocate Join 的使用参照：Colocation Join - Apache Doris
9. 【强制】亿级别大表禁止使用select * 查询，查询时需要明确要查询的字段
   a. SQL Block方式禁止这种操作
   b. 如果是高并发点查，建议开启行存

```
1     表属性级别
2     "enable_unique_key_merge_on_write" = "true",
```

```
    "store_row_column" = "true"

    be.conf
    disable_storage_row_cache 是否开启行缓存， 默认不开启

    使用PrepareStatement模板
```

10. 【强制】亿级以上表数据查询必须带分区分桶条件