GoldenDB 分布式数据库应用开发规范

2024 年 03 月

Version：1.5

修订记录

版本号	修订说明	变更人	变更日期
1.0	初版建立	曹健	2023-09
1.1	1、细化“2.2.2.表命名规范”，补充各类场景。 2、细化和优化“2.2.3.列命名规范”，补充各类场景。 3、优化“2.2.5.约束命名规范”。 4、新增“2.2.6.序列命名规范”。 5、优化“3.2.2.主键”。 6、修改“3.2.6.3.禁止大表/高频变更表使用复制表（分布式）”。 7、补充“3.4.2.日期和时间类型”。 8、细化“4.2.4.批处理必须避免大事务,必须分批提交”描述。 9、优化“4.3 DDL 规范”内容，新增“4.3.6.【建议】表重组”，新增“4.3.7.【规则】DDL 执行失败处理”。 10、新增“4.6.UPDATE 规范”。 11、优化“4.7.DELETE 规范”。 12、其他优化。	曹健	2023-10
1.2	1 修改为分布式的开发规范，集中式的规范转到《GoldenDB 集中式开发规范 v1.0》 2 修改索引命名规范、修改了约束命名 3 修改 3.5.6.1 组合索引中索引列的先后顺序规则 4 增加了 4.2 读写分离 4 修改了 4.8.7.5 or 子句的列 5 增加了 5.2 dbtool 卸数 6 增加了 5.3 增量卸数 7 增加了 4.5.14 增加了禁止大表不带条件直接查询 8 删除了 3.5.5 中禁止使用 char 类型 9 增加了 1.1 中各类用户的具体权限 10 3.4.3 必须为 not null 改为建议为 not null	宋中蓉	2023-10
	1 增加了第一章 采用分布式数据库基本原则 2 修改了 4.2.3 表必备字段改为表设计应考虑的字段 3 增加了 5.10 truncate 操作规范，在 truncate 后进行数据写入是一定要做 analyze table	宋中蓉	2023-12
1.3	1 增加了 5.4.9 限制一次性添加分区的个数	宋中蓉	2024-2
	2 4.1.1 增加了数据库连接（池）配置参考	宋中蓉	2024-2
1.4	1 增加 5.2 dbtool 卸数中关于生僻字的注意事项	宋中蓉	2024-2
1.5	1 修改 4.3.1.分片规则，大表规则：10K 改为 8K。 2 修改 4.5.3.分区表注意事项，限制分区数分区数 100 改为 200。 3 去掉 4.6.6.5.禁止组合索引的索引字段过长。 4 去掉 5.5.9.【建议】select 存在 exists 等值关联子查询时，	黄悦悦	2024-3

	exists 子查询结果集较小时(<2000)，建议改为 in 子查询。 5 prepStmtCacheSize=50 默认值 50 改为 500。 6 部分内容不支持改为不允许。 7 5.5.37.【禁止】不允许@变量赋值。		

目录

第一章 采用分布式数据库基本原则

1.1. 基本原则

优先使用集中式数据库，当经过调研、测试认证后集中式无法满足要求时才采用分布式数据库，且采用分布式数据库时要考虑到复杂 SQL 的解决方案。

第二章 用户和权限规范

2.1. 数据库用户和权限

- 应用程序严禁使用实例用户和 DBA 权限的用户来访问数据库。

应用程序使用的账号权限：Alter,, Create,Delete, Drop, Execute, Index, Insert, Lock Tables, Select, Show View, Update

- 除应用用户以外增加跑批专用账号

拥有的权限：Create, Delete, Drop, Insert, Select, Update

- 增加开发人员操作权限账号

拥有权限：Create, Delete, Drop, Insert, Select, Trigger, Update

- 增加开发人员使用的查询账号

拥有的权限：Select

- 增加监控账号

拥有的权限：Select,Replication Client, Replication Slave, Show Databases,process

2.2. NAS 盘权限

在多个 loadserver 的情况下，需要赋子 NAS 盘目录的权限是 770。

第三章 数据库对象设计

3.1. 数据库对象命名通用规范

命名规范是指数据库对象如数据库(SCHEMA)、表(TABLE)、列名(COLUMN)、索引(INDEX)、约束(CONSTRAINTS)等的命名约定。

实践中需遵循以下原则:

1. 对象名做到见名知意，使用具有意义的英文单词或者英文单词宿写。

2. 单词之间用下划线“_”分隔。

3. 命名只能使用英文字母、数字、下划线，并以英文字母开头。

4. 所有数据库对象使用小写字母，禁止使用大写。例子:’CARD’，’carD’，其中 CARD，carD 禁止使用。

5. 对象名长度限制:

   - 数据库名不允许超过 8 个字符长度。
   - 表名和列名长度不允许招过 40 个字符，如果过长请用缩写。
   - 索引名和约束名长度不允许超过 64 个字符。
   - 其它对象名的长度不允许超过 40 个字符。

6. 严禁使用汉语拼音或者拼音首字母缩写，约定俗成的除外。

7. 禁止使用 GoldenDB 关键字单独作为对象名，如 date、rank、order 等。

3.2. 数据库对象命名详细规范

3.2.1. 数据库命名规范

1. 数据库名应尽可能和所服务的业务模块名一致。不同的项目使用不同的数据库名，不允许不同的项目使用相同的数据库名。

2. 数据库名必须简洁，不允许数据库名超过 8 个字符。

3.2.2. 表命名规范

同一个中心的表使用相同的前缀，表名称需要表达该表数据的业务含义。原则上命名以词根组成为准， 超长可以只选择部分关键词根表示，约定规则如下:

1. 所有表的表名以【中心应用名】开头。例如:贷款余额档 ldctc_bal。
2. 历史表以”_hist”作为后缀；
3. 明细表以”_detail”或者”_dtl”作为后缀；
4. 参数表以”_param”作为后缀；
5. 关联关系表以“_rel”作为后缀；
6. 映射表以“_map”作为后缀；
7. 临时表以“_tmp”作为后缀；
8. 中间表以“_mid” 作为后缀；
9. 程序需要访问的备份表以“_copy”作为后缀；
10. 程序不会访问的备份表以“_backup_”+日期或者“_bak_”+日期作为后缀；
11. 只用于 ods 抽数的表以“_ods"作为后缀。

3.2.3. 列命名规范

1. 标识符字段:以”_id”后缀，如：user_id 表示标识符。
2. 序号列字段:以”_no”后缀，如:trx_seq_no 表示交易序号。
3. 代码字段：以”_code”后缀，如:dist_code 表示地区代码。
4. 日期字段（类型是 date，只有年月）：以”_date”作为后缀。如:open_date 表示开户日期。
5. 时间字段（类型是 time，只有时分秒）：以”_time” 作为后缀。如：close_time 表示关门时间。
6. 日期时间类型（类型是 datetime，包含年月日时分秒）：以“_dt”或者“_datetime”作为后缀。
7. 时间戳字段（类型是 datetime(6)，包含年月日时分秒毫秒微秒）：以“_ts”作为后缀。
8. 布尔值字段：以“_flag”作为后缀。

3.2.4. 索引命名规范

1. 唯一索引以“uniq _字段名”方式命名。如果有多个多个字段组成的唯一索引，使用“uniq_字段名 1_字段名 2”来命名。
2. 辅助（二级）索引以“idx _字段名“命名。如果有多个索引，则使用 idx _字段名 1_字段名 2 “
3. 全局索引以“gidx _字段名”命名。
4. 全局唯一索引以“guniq _字段名”命名。

3.2.5. 约束命名规范

1. 主键约束无须命名。
2. 唯一约束以“uniq_字段名”方式命名。

3.2.6. 序列命名规范

序列（sequence）全部以“seq_表名”命名。

第四章 数据库设计规范

1. 数据节点单实例总数据空间建议不超过 1TB。

2. 数据节点单实例表数量不超过 1 千张。

3. 业务使用的数据集群是否分片以及分片数可按下表整理。

序号	租户名	租户分片模式	分片数	数据量
1	业务中心 1	单分片		316G
2	业务中心 2	多分片	2	1.4T

4.1. 数据库设计规范

1. 单实例的数据库不应该超过 10 个，允许单实例多数据库。

2. 统一使用 utf8mb4 字符集和 utf8mb4_bin 排序规则。

4.2. 表设计规范

4.2.1. 存储引擎及字符集

统一使用 innodb 存储引擎， ufimb4 字符集和 ut8mb4_ bin 排序规则。

4.2.2. 主键

1. 所有的表必须有一个主键。

2. 所有需要同步到其他平台的表必须有一个业务主键或者业务唯一索引。

3. 主键的选择原则如下：

- 参数表全部用业务主键作为主键。
- 如果是主要以业务主键进行查询而且不会发高并发 INSERT 的表，使用分片键 + 业务主键作为主键。
- 如果该表（比如流水表，日志表，历史表）有高并发 INSERT 操作，使用自增列 id 做主键。
- 禁止使用自动更新属性 （ON UPDATE CURRENT_TMESTANP(6)）的时间戳作为主键，使用时间戳作为单列主键可能导致主键冲突，会导致无法删余数据。

4.2.3. 表设计应考虑的字段

1. 除了参数表外，所有的表必须要有对应的业务字段 id 字段。

2. 在在设计时应考虑包含两个字段：create_ts(创建时间戳)， update_ts(修改时间戳)且非空。对表的记录进行更新的时候，包含对 update_ts 字段的更新。并且定义如下：

```
create_ts datetime(6) not null default CURRENT_TMESTAMP(6) comment ‘创建时间戳’,
update_ts datetime(6) not null default CURRENT_TMESTAMP(6) on update CURRENT_TIMESTAMP(6) comment '修改时间戳',
```

3. 根据实际业务需要设置记录创建柜员号，创建柜员机构，修改柜员号，修改柜员机构等。

4.2.4. 范式与冗余平衡原则

表设计应遵循范式与冗余平衡原则，一般遵守第三范式，但在需要对多张表进行关联查询，为了提高查询的效率，可以做适当的冗余。但冗余字段必须遵守:

1. 非频繁修改字段；

2. 非超长字段，如 text 类型字段。

4.2.5. 表的生命周期管理

需要和业务人员确定表的生命周期，联机交易系统必须保留尽可能短时间的数据，项目组需要制定数据清理规则，避免联机交易用的数据库容量不断增加。

4.2.6. 禁止事项

4.2.6.1. 禁止在表中建立预留字段

1. 预留字段的命名很难做到见名识义。

2. 无法确认存储的数据类型，所以无法选择合适的类型。

3. 对预留字段类型的修改，并不会比新增列效率更高。

4.2.6.2. 禁止在数据库存储图片、文件等二进进制数据

1. 将图片、文件等二进制数据存放到数据库中，不仅数据写入量大，日志写入量也大，会造成数据库空间耗用非常严重。

2. 另外这些数据的存储很消耗数据库主机的 CPU 资源。这些数据的处理不是 GoldenDB 的优势。

3. 请使用其它方式存储，GoldenDB 只保存文件存储路径。

4.2.6.3. 禁止大表/高频变更表使用复制表

禁止原因：一个表的设计容量大于 300 万行，该表称为大表。禁止大表采用复制表模式。复制表在每个分片上均有相同的全量数据，适合用于被其他表进行本地关联查询。如果表的规模较大，并没有达到数据分散存储的目的，无法通过增加机器数量扩展分布数据库的存储和计算性能。对复制表的变更，如插入、更新和删除操作，需要在每个分片执行成功，整个事务才能成功，如果分片数量较大，则更新效率低下。只允许更新较少的表使用复制表，禁止经常更新的表使用复制表模式。

优化方法：

- 随着业务增长记录行数会增加的表，应避免采用复制表模式。
- 如果表的规模较小，可以采用单节点表；大表应采用哈希、范围或列表等分片模式，将数据打散到多个分片。

原 SQL 示例

```
--汇率表采用复制表，但每分钟需要更新一次
create table exchange rate(currency， rate，...)
distributed by duplicate(g1，g2，g3);
--每次更新需要在多个分片上同同时执行
update exchange_rate set rate=... where currency='cny';
```

优化 SQL 示例

```
--汇率表改用单节点复制表（只使用 g1 一个分片）
create table exchange_ rate(currency， rate，...)
distributed by duplicate(g1);
update exchange_rate set rate=... where currency='cny';
```

4.3. 分片设计规范

数据分片：把数据库横向扩展到多个物理节点上的一种分布式技术。可以理解为将表数据按照特定的分片规则水平切分成若干片段(shard)，使这些数据片段分布在不同物理节点上。

优势：

- 无限扩展：数据分片是分布式数据库的基础技术，实现存储容量的横向可扩展(理论上可以无限扩展)
- 性能提升：单个分片上数据集变小，数据库的查询压力变小、查询更快，性能更好。数据被水平切分成多个部分，实现了查询并发执行，增加了系统的吞吐量
- 高可用：若部分分片出现宕机，影响面仅涉及这几个分片的服务，不会造成整个系统的瘫痪，提高了数据库的可用性。

4.3.1. 分片规则

表的大小主要有 3 个维度

- 表的用途(读写频率)
- 表的数据量(行数)
- 表的数据大小(单行平均大小)
  - 小表：数据量级别在 10 万以下、单行小于 1K、读频率高，写频率低。
  - 大表：数据量级别在百万以上或单行数据量大于 8K 或写频率高。

以下描述均指多分片实例模式下:

- 小表优先使用单节点存储的表，大表优先使用多节点分布存储的表。
- 需要进行关联查询的相关大表，优先使用相同的分片规则和分片键；
- 去范式化原则
- 单个分片表建议不超过 5000W 行
- 针对多节点分布存储的表
  - 在采用 HASH/RANGE/LIST 分发策略时，尽量保持各个 DB 节点上的数据量均衡；
  - **HASH 分片**：适用于大多数分布式表，可以将数据均匀地分布到预先定义的数据节点上，保证各数据节点的数据数量大致一致，一般情况下不需要关心分发字段值的具体含义。对于等值操作的表特别适合。
  - **RANGE 分片**：指定一个给定的列值或列值集合应该保存在哪个数据节点上，常用于时间字段上，比如数据按照自然月或天来分布存储。
  - **LIST 分片**：适用于含有一系列限定性值的场景。
  - **复制表**：适用于小表且读多写少的表，或者 JOIN 和子查询中使用的表。主要目的是减少节点间网络数据的传输，以提高查询的性能。复制表可以指定保留全量数据副本的数据分片数

目，可以是 1 个（单节点复制表），也可以是若干个数据分片，甚至在所有分片均保留一份全量数据（多节点复制表）

- **多级分片**：适用于通过多维属性对数据分布精确控制的场景。如以银行核心应用为例，法人、集团、公/私属性的客户信息精确管理的场景：不同法人的客户可以在物理上分开、对公和对私客户分开、对公客户可能属于不同的集团，可以按照集团属性分开。

4.3.2. 分片键设计规范

- 分片键的选择需要结合业务场景分析得出，需涵盖主体或者全部的业务库表的访问维度提炼得出，其目的是让大部分的业务语句可以直接下压到分片计算，减少跨分片的操作；
- 分片键确认后执行 DDL 建表语句时指定；
- goldendb 要求必须指定分片键，无法指定合适的分片键可以使用主键或是自增列。

4.3.3. 分片最佳实践示例

1. 某券商业务迁移 GoldenDB，业务表中存在特定相关属性确定值的数据范围，选择按照特定属性值进行 list 分片划分，如按照区域编码具体属性划分，不同区域间的数据分布不会存在干扰，区分度高，最大限度减少范围条件查询引起的跨分片操作。
2. 某国有大行业务迁移 GoldenDB，整体分布策略采用多级分片，按照应用分区号前两位(省分行号)进行一级分片，有些分行通过一级分片后，可以直接选定分片。有些分行通过一级分片路由后，还不能选定到单独的分片，需要使用应用分区号的中间三位(客户号)的范围进行二级分片。对于较大的表，在路由已经能够选定分片后，通过 partition 来确定需要检索的分区，提高查询效率。
3. 某股份银行核心账户体系中有介质号、主账户、子账户三层层数据任意一个层级的信息先开立时候，以当时归属的客户号进行数据分片，确定分片键值，并记录下来。其他层级数据在建立时，

除了新增和已建立的其他层级数据关联关系时，也沿用同一数据分片键值，不能再按照客户号确定分片。这样确保归属同一账户的卡号、主账号、子账号三层数据，不管何时开立，都在同一分片。后续该账户的相关交易，这三层数据联动的增删改查，都是同一个数据分片里面完成，减少跨节点事务，同时考虑负载均衡。

4.4. 列设计规范

4.4.1. 数据据类型和长度

1. 合理地设置列的类型和长度，可以节省 GoldenDB 的表空间。

2. 列所占的空间小，建立索引时所需要的空间也就越小，这样一页中所能存储的索引节点的数量也就越多，索引树层级就越少，在遍历时所需要的 IO 次数也就越少，索引的性能也就越好。

3. 能用数字类型存储，就不要使用字符类型。

4. 关联列的数据类型和长度必须完全相同，如果关联表之间相同的数据类型不一致，那么在通过关联列进行关联表之间的联合查询的时候，会导致执行计划出现问题，影响 SQL 语句的性能。

4.4.2. 日期和时间类型

1. 使用 DATE 类型表示日期，使用 DATETIME 类型来表示年月日时分秒。

- DATE 的范围是'1000-01-01’到'9999-12-31’，占用 3 个字节。所有只需要精确到天的字段全部使用 DATE 类型，而不应该使用 TIMESTAMP 或者 DATETIME 类型。
- 所有需要精确到时间（时分秒）的字段均使用 DATETIME，不允许使用 TIMESTAMP 类型，TIMESTAMP 也可以用来表示年月日时分秒，但最大日期只能支持到 2038 年 1 月 18 日。DATETIME 比 TIMESTAMP 类型只多占一个字节的存储空间，最大值是’9999-12-31 23.59.59[.9999]’。

- 所有需要精确到时间（时分秒毫秒微秒）的字段均使用 DATATIME(6)类型。

2. 年月日时分秒的时间格式统一使用格式: ‘YYYY-MM-DD HH:MM:SS’

4.4.3. 是否允许 NULL 和默认值

1. 除 DATE、 DATETIME、TEXT 类、 BLOB 和 JSON 类型字段允许为空外，其他数据类型建议为 NOT NULL，并设置默认值。各种数据类型的理论值设置如下:

- 字符类的默认值使用”，例子: user_name varchar(20) not null default”；
- 数字类的默认认是 0；
- 时间类的默认值是 ‘00:00:00’ ；

2. 关于日期和日期时间类的列的规范:

- 新增日期字段，原则不允许为 null，特殊情况需要和 dba 申请;
- 存量的日期字段，根据生产实际数据情况，设置是否允许为 null;
- 日期的字段如果属于主键，不允许为 null；
- create_ts、update_ts 原则不允许为 null，特殊情况需要和 dba 提出申请；
- 日期类和日期时间类都不给默认值。

3. 建议不使用 NULL 的原因

- NULL 列需要额外的更多的存储空间；
- NULL 只能采用 IS NULL 或者 IS NOT NULL 查询，使用!=/in/not in 查询是得不到期望的结果。

4.4.4. 列必须指定注释

用 comment 属性来描述列所代表的真正业务含义，如果是枚举值则需要将该字段中使用的内容都定义出来。例如:

```
‘record_sts’ char(i) NOT NULL DEFAULT ‘’ COMMET ‘记录状态： 0-正常（NORMAL） 1-正常结清（CLOSE）2-
```

```
受托协议到期终止（CLOSE） 3-受托协议转让终止（CLOSE） 4-受托协议协商
终止(CLOSE） 5-现金折让终止(CLOSE ） 6-其它原因结清（CLOSE)’
```

4.4.5. 禁止事项

1. 禁止一张表内的字段数超过 200。

2. 除大字段外，禁止单行超过 8092 字节。

3. 禁止日期和时间类型的定义为 VARCHAR。

4. 禁止数字类型定义为 VARCHAR。

6. 禁止不带小数的数字采用 DECIMAL(m，0)类型。

4.5. 分区表设计规范

4.5.1. 分区表的使用场景

分区表主要适用于如下场景：按日期访问和归档的表，比如历史表，流水表和明细表。查询时加上时间范围条件效率会非常高，同时对于不需要的历史数据能很容易的批量删除。

☑注意：按照日期进行数据清理的必须设计成分区表。

4.5.2. 使用 COLUMNS 分区

1. 只使用范围和列表分区方式。GoldenDB 的 HASH 分区对于范围查询没有比较好的帮助，同时 GoldenDB 只支持嵌套循环连接。

2. 使用 COLUMNS RANGE 和 COLUMN LIST 分区来代替传统的 RANGE 和 LIST 分区。LIST 分区只支持整数分区，需要通过额外的函数运算从而得到整数。COLUMNS 分区支持整形（TINYINT 到 BIGINT）、日期（DATE、DATETIME）、字符串（CHAR，VARCHAR，BINARY，VARBINARY）三大数据类型。

3. COLUMNS 分区分为 RANGE COLUMNS 和 LIST COLUMNS。

4.5.3. 分区表注意事项

1. 主键、唯一键必须包含分区列。

分区表的 primary key 和唯一索引，必须包含用于分区的列。否则会报错，以主键为例：“ERROR 1503 (HY00):A RIMARY KEY must include all columns in the table’s partitioning function”。

2. 允许按天或者按月分区。

- 按天清理的使用天分区，统一命名为【pYYYYMMDD】
- 按月清理的使用月分区，统一命名为【pYYYYMM】
- 在开发测试环境下，为了避免在缺分区的情况下报错“Table has no partition for value from column_list”，可以建立 pmax values less than (maxvalue) 分区。投产时，必须在 DDL 中删除该分区。

3. 联机常用的 SQL 条件中必须要带上分区列作为查询条件。

SQL 条件中要带上分区条件的列，从而使查询定位到少量的分区上，否则就会扫描全部分区，可以通过 EXPLAIN 来查看某条 SQL 语句会落在哪些分区上，从而进行 SQL 优化。如果常用的 SQL 不能带上分区列作为查询条件来限制访问的分区，请不要使用分区表。否则会造成 IO 增大，CPU 使用率增多，SQL 响应时间变长，性能下降。

4. 采用统一的批处理框架对分区进行维护。

- 需要提前预分配分区，比如按日新建每日的分区。
- 过期的分区需要删余。

5. 限制分区数。不允许分区数超过 200。(确定按月还是按日)。

6. **按月分区的表投产时必须提前建好 1 年的分区，按天分区的表投产时必 须提前建立 1 个月的分区。**

7. 按月分区的表至少保留 3 个月，按天分区的表至少保留 7 天的分区。

4.6. 索引设计规范

4.6.1. 选择合适的索引字段

索引常被用来优化如下场合的 SQL:

- SELECT、UPDATE、DELETE 语句的 WHERE 条件中的列。
- 包含在 ORDER BY， GROUP BY， DISTINCT 的字段。
- 多表 join on 的字段。

索引字段的选择需要结合业务需求，评估出应用中作为查询条件出现比较频繁的字段，在此字段上建立单独或者组合索引。选择建立索引的字段，应该遵循以下的原则:

1. **高选择性。**选择性是指某列的不同值的数量占该表总数据量的百分比。选择性越高，通过索引查询返回的结果集越少，索引更为高效。在 OLTP 应用系统中，选择性应接近于 100%。

2. **数据分布均匀。**索引字段中，个别数据值占总数据量的百分率明显比其它数据值占总数据量的百分率高，表明该字段数据值分布不均，容易引起数据库选择错误索引，生成错误的查询执行计划。应该避免在数据值分布不均的字段上建立索引。

4.6.2. 索引类型选择

建立单列索引还是组合索引，选择方式如下:

- 如果列的选择性非常高(接近于 100%)，适合建立单列索引。
- 根据查询条件，如果单列选择性不高，但组合列选择性较高时，需建立组合索引。

如下 2 个例子，分别建立单列索引和组合索引。

例 1：查询卡主档表中账号为 12345678 的信息:

```
SELECT card_no，card_type expiry_date
FROM card
WHERE acct_no = '12345678' and open_date= ‘2018-01 -01’
```

由于每个 acct_no 在卡主档表中记录很少发生重复，所以适合使用单列索引(acct_no)。

例 2：查询卡号为 12345678 的某个月的交易明细数据:

```
SELECT card_type, amt
FROM card_detail
WHERE card_no = '12345678' and txn_date between '2020-01-01' and '2020-01-31'
```

由于每个卡号可能有大量的交易记录，使用建立组合素引(card_no ，txn_date)可以通过索引减少索引和数据页读取数，从而降低 I0，节省 CPU，降低 SQL 响应时间。使用单列索引(card_no)在高并发情况下会导致 IO 繁忙， CPU 使用率高， 响应时间偏高。

4.6.3. 唯一索引

GoldenDB 的分片表里默认唯一索引在没有分片字段情况下，无法保证全局唯一，只能保证分片内唯一。如果需要保证全局唯一，需要使用全局唯一索引(使用关键字 global)。

4.6.4. 全局索引

GoldenDB 的全局索引是内部维护了索引列与分片键之间的关系。

4.6.4.1. 全局索引自身的功能和性能限制

GoldenDB 的全局索引存在如下限制（全局索引功能持续更新，请参考当前版本）：

1. 不支持 ALTER TABLE ADD/DELETE/MODIFY 全局索引（主键）操作，使用（drop/create）代替;
2. WHERE 条件中字段必须配联合全局索引的全部列，才能使用该全局索引;
3. WHERE 存在多个索引等值链时: 索引 1 AND 索引 2，第一个索引生效；索引 1 OR 索引 2，则索引不生效;
4. 包含有全局索引的表，不支持修改分片键值;

5. 如果 where 子句对全局索引列含有 or 或者 in 谓语，无法将 SQL 指定到分片；

6. 全局索引只能通过 create table 语法来创建，无法通过 alter table 语法来新增；

7. 多级分片表不支持全局索引；

8. 不能在复制表创建全局索引；

9. 全局主键不支持的类型：float，double，blob，text；

10. 不支持删除联合全局索引中的列；

11. 全局索引字段不支持 INSERT ON DUPLICATE UPDATE;

12. 全局索引字段不支持 MERGE INTO 操作;

13. 不支持 SHOW TABLE 查看全局索引表;

14. 对有全局索引的表修改索引字段或查询条件为索引字段的情况，prepare 不生效；

15. 有全局索引的表，LDS 数据导入时不自动处理索引数据；索引数据需单独处理;

16. 有全局索引的表，重分布源表时不自动重分布索引表;索引表的重分布由系统管理员单独发起。

4.6.4.2. 使用全局索引的注意事项

1. 全局索引主要适用选择性高的列(账号，卡号等等)，严禁对选择性较低的列(比如日期类)单独作为全局索引。

2. 限制单表的全局索引的个数：不超过 1 个。

4.6.5. 组合索引

4.6.5.1. 索引列的先后顺序规则

1. 将具有选择性较高、使用最频繁且经常是等值查询或者 N 的列放在组合索引的最左侧。

2. 将具有一定选择性且是范围查询(>、<、>=、<=、BETWEEN...AND)放在组合索引的右侧。

3. 排序列必须紧跟在等值条件列的后面，能使用该索引避免排序。如下例：对于 KEY (a，b)

```
-- 该查询能使用索引(a，b)来排序
SELECT * FROM tbl WHERE a = 1 ORDER BY b LIMIT 5
  -- 该查询不能使用索引(a，b)来排序--
SELECT * FROM tbl WHERE a IN (1，2) ORDER BY b LIMIT 5
  --该查询不能使用索引(a.b)来排序
SELECT * FROM tbI WHERE a between 1 and 10 ORDER BY b LIMTT 5
```

例如:

对于如下 SQL:

```
select
  id，
  part_no，
  mib_ac_no，
  tr_datetime，
  vch_no，
  vch_no_seq，
  br_no，
  ccy，
  itm_no，
  itm_seq，
  eff_date，...
from
  acc_online_mid_detail
Where
  mib_ac_no = ’ 10000001 ’
  and vch no = '008039'
  and vch_no seg = 1201
  and br_no = '1731'
  and tr_datetime >= '2020-12-20 00:00:00'
  and tr_datetime < ’2020 -12-21 00:00:00’
  and tr_timestamp >= '2020-12-20 00:00:00'
  and tr_timestamp < '2020-12-20 00:00:00'
limit
  1 \ G
```

使用如下 2 个索引:

- 索引 1 (mib_ac_no，vch_no，vch_no_seq，br_no，tr_datetime，tr_timestamp)
- 索引 2 (mib_ac_no，tr_datetime，vch_no，vch_no_seq，br_no，tr_timestamp)

  由于 SQL 中 vch_no，vch_no_seq，br_no 的谓词是等号且这些列有一定的选择性，tr_datetime，

tr_timestamp 是范围。采用索引 1 的性能高于索引 2。

4.6.5.2. 尽量能让更多的查询使用

如下两个语句:

```
SELECT * FROM tbl WHERE a=5 AND b=6;
SELECT * FROM tbl WHERE a>5 AND b=6;
```

- 索引(b，a)能够满足上面的 2 个查询。
- 索引(a，b)对第 2 个查询的性能不好。

4.6.5.3. 匹配规则

索引匹配起始于等号，终止于范围谓词。查询条件的列必须是索引的第一列才能匹配索引。例如：

```
SELECT FROM tbl WHERE a=1 and b BETWEEN 6 AND 9 AND c=6
```

对于 KEY(a，b，c)，该查询只能匹配索引（a，b，c)的 a 列和 b 列，c 列不属索引匹配列。因为索引匹配列终止于范围谓词。

```
-- 对于 KEY (a，b)，该查询可以使用索引(a，b)，a 列是索引匹配列。
SELECT * FROM tbI WHERE a=6
-- 该查询不能匹配索引(a，b)。
SELECT * FROM tbI WHERE b=6
```

4.6.5.4. 列数限制

一般地，每个组合索引的列总量不超过 5 个。

4.6.6. 索引注意事项

4.6.6.1. 索引数量限制

1. 一般地，每个表索引数量不超过 5 个。特殊情况下，如果部分表存在大量多维度查询的交易，索

引数可以超过 5 个，但索引数不宜太多。

2. 索引并不是越多越好！索引可以增加查询效率，但同样也会降低插入和更新的效率，甚至有些情况下会降低查询效率。

GoldenDB 优化器在选择如何优化查询时，会根据统一信息，对每一个可以用到的索引来进行评估，以生成出一个最好的执行计划，如果同时有很多个索引都可以用于查询，就会增加 GoldenDB 优化器生成执行计划的时间，同样会降低查询性能。

4.6.6.2. 对 TEXT、BLOB 和长 VARCHAR，必须使用前缀索引

1. 对于 BLOB、TEXT，或者很长的 VARCHAR 类型的列，为它们的前几个字符（具体几个字符是在建立索引时指定的）建立索引，这样的索引就叫前缀索引。
2. 这样建立起来的索引更小，所以查询更快。
3. 前缀索引也有坏处，它不能在 ORDER BY 或者 GROUP BY 中使用前缀索引，也不能用做覆盖索引。
4. 创建前缀索引的语法：

```
create index idx_column on table_name(column_name(prefix_length))
```

或者

```
alter table table_name add key(column_name(prefix_length))
```

prefix_length 的选择原则，满足 count(distinct (left(column_name, prefix_length)))/count(*) 的值大于 0.8。

4.6.6.3. 不在选择性低的列上建立单独索引

选择性低的列，比如状态字段，性别字段等等这些字段中存放的数据可能总共就不到 10 个不同的值，每个值可能存放在成千上万或者更多的记录。对于这类字段，我们没有必要创建单独索引。因

为即使创建了索引，GoldenDB 大多数时候不会使用，即使 GoldenDB 选择了这种索引，可能会带来很大的性能问题。由于索引字段中的每个值都含有大量的记录，那么存储引擎在根据索引访问数据的时候会带来大量的随机 IO。

4.6.6.4. 禁止给表中的每一列都建立单独的索引

给每一列都建一个单独索引，会导致 INSERT、UPDATE 和 DELETE 性能大幅下降，索引占的空间变大。优化器通常只会选择其中一个索引，放弃其它的索引。即使选择了同时利用两个或更多的索引通过 INDEX_MERGE 来优化查询，可能收到的效果不会比选择其中一个单列索引更高效。

4.6.6.5. 禁止重复索引和冗余索引

1. 重复索引是以相同的顺序在相同的列上创建索引。例如：

```
CREATE TABLE t1(
id INT NOT NULL PRIMARY KEY,
UNIQUE(id),
key(id)
);
```

该表在 ID 列上创建了 3 个相同的索引。规范的方法：

```
CREATE TABLE t1(
id INT NOT NULL PRIMARY KEY
);
```

2. 冗余索引是指一个索引的列为另一个索引的前 N 列。例如 KEY(a，b)和 KEY(a)就是冗余索引，规范的方法是只保留 key(a，b，c)。

```
CREATE TABLE t2(
id INT NOT NULL PRIMARY KEY,
a INT NOT NULL DEFAULT 0,
b INT NOT NULL DEFAULT 0,
c INT NOT NULL DEFAULT 0,
KEY(a, b, c),
KEY(a, b),
KEY(a),
);
```

4.6.6.6. 不允许使用全文索引

由于全文检索返回结果集时会考虑权重、最小字符长度为 4 等，可能导致得到出乎意料的结果集，因此不允许使用全文索引。

4.7. 禁止事项

4.7.1. 禁止使用外键约束

外键约束使得表与表之间相互耦合，影响 UPDATE/DELETE 等 SQL 性能，有可能造成死锁，高并发情况下容易成为数据库瓶颈。应在应用层实现外键约束。

4.7.2. 禁止使用视图

虽然视图一定程度上让 SQL 看起来更简单，但背后的逻辑并没有减少，视图嵌套使用还容易导致性能问题。无特殊理由，禁止使用。

4.7.3. 禁止使用 event、存储过程、自定义函数

event、存储过程、自定义函难以调试且扩展性差，无移植性，应在应用层实现。无特殊理由，禁止使用。

4.7.4. 禁止使用触发器

使用触发器可能导致性能问题，应放在应用层实现。

4.7.5. 禁止使用临时表

第五章 数据库开发规范

5.1. 通用开发规范

5.1.1. 必须使用成熟的数据库连接池技术

具体要求如下:

1. 保持稳定的长连接数，不频繁的连接数据库。
2. 保持稳定的性能，高压力下性能不下降。
3. 能适应复杂的网络环境，避免断网问题。
4. 有自动重新连接数据库的机制。

数据库连接 url 配置

```
rewriteBatchedStatements=true&
useUnicode=true&
characterEncoding=utf8&
zeroDateTimeBehavior=convertToNull&
connectTimeout=3000&
socketTimeout=600000&
dumpListSize=0&
useCursorFetch=true&
prepStmtCacheSqlLimit=20480&
prepStmtCacheSize=500&
reclaimBufferLength=51200&
convertClientPstmts=false&
isConnectionLevel=true
```

联机：prepStmtCacheSqlLimit=1024&prepStmtCacheSize=500

批量：prepStmtCacheSqlLimit=20480&prepStmtCacheSize=500

示例（批量应用）

```
jdbc:mysql://127.0.0.1:3306/ens_rb?
rewriteBatchedStatements=true&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&connectTimeout=3000&socketTimeout=600000&dumpListSize=0&useCursorFetch=true&prepStmtCacheSqlLimit=20480&prepStmtCacheSize=500&reclaimBufferLength=51200&convertClientPstmts=false&isConne ctionLevel=true
```

连接池的配置参考

配置项	基线值	配置说明	备注
maxWait	60000	业务线程获取连接最大等待时间(毫秒)，超过则抛出异常	建议配置最大等待时间，避免业务线程长时间等待获取连接
validationQuery	SELECT 1 FROM DUAL	验证连接有效性使用的 SQL	MySQL、Oracle 可不配置，使用 GoldenDB 数据库需要配置
testOnBorrow	TRUE	当应用向连接池申请连接时，连接池会判断这条连接是否是可用的	建议设置为 ture，会多一次网络交互的时间，影响很小部分的性能
testOnReturn	TRUE	当应用使用完连接，连接池回收连接的时候会判断该连接是否还可用	建议设置为 ture，会多一次网络交互的时间，影响很小部分的性能
testWhileIdle	TRUE	连接空闲时验证	连接池将会判断连接是否处于空闲状态，如果是，则验证这条连接是否可用
keepAlive	true	连接保持，包括连接数小于最小空闲连接数时补充连接和定时发送验证语句用于保证连接有效性	建议开启，在连接数小于最小连接数时可以自动补充连接
keepAliveBetweenTimeMillis	120000	配置连接保持时间间隔（毫秒），当开启连接保持后，空闲连接会在满足连接保持时间间隔后进行连接验证	配置应满足 minEvictableIdleTimeMillis >= keepAliveBetweenTimeMillis > timeBetweenEvictionRunsMillis
maxEvictableIdleTimeMillis	900000	连接空闲时间大于该值，不管 minidle 是多少都关闭这个连接	根据实际需求进行配置，应小于数据库连接最大空闲时长、F5 空闲超时时长、防火墙允许的长连接时长等，默认 7 小时
minEvictableIdleTimeMillis	300000	配置一个连接在池中最小生存的时间，单位是毫秒	配置应满足 minEvictableIdleTimeMillis >= keepAliveBetweenTimeMillis > timeBetweenEvictionRunsMi

			llis
timeBetweenEvictionRunsMillis	60000	连接验证时间间隔（毫秒）	配置应满足 minEvictableIdleTimeMillis >= keepAliveBetweenTimeMillis > timeBetweenEvictionRunsMillis
poolPreparedStatements	false	是否缓存 preparedStatement，也就是 PSCache	会导致 JDBC 缓存失效或者运行 statement 报错
maxPoolPreparedStatementPerConnectionSize	0	指定每个连接上 PSCache 的大小	关闭 PSCache
removeAbandoned	false	发现连接泄露后是否回收连接	会导致正在运行的链路被强制关闭
minIdle	10	连接池中的最小空闲连接数，如果空闲的连接数大于该值，则关闭多余的连接	>0，与 maxEvictableIdleTimeMillis 配合可以解决 F5 CN 负载不均匀问题。一般配置为最大值的 1/2，建议最小连接数至少满足日常业务处理需要的连接数。
exceptionSorter	exceptionSorter=com.alibaba.druid.pool.vendor.MySqlExceptionSorter	剔除"不可用连接"，Druid 使用了 ExceptionSorter 机制	使用 GoldenDB 数据库需要配置

连接数的配置根据业务情况来判断，以实际性能压测做参考。

深农商的数据源配置（druid）

```
spring:
    datasource:
    #数据源启动初始化类
    driver-class-name: com.mysql.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource
```

```yaml
    druid:
      name: ${spring.application.name}
      # 打开 PSCache，并且指定每个连接上 PSCache 的大小
      pool-prepared-statements: true
      #要启用 PSCache，必须配置大于 0，当大于 0 时，poolPreparedStatements 自动触发修改为 true
      max-pool-prepared-statement-per-connection-size: 20
      #最小空闲连接数保活
      keep-alive: true
      # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
      time-between-eviction-runs-millis: 60000
      # 配置一个连接在池中最小生存的时间，单位是毫秒
      min-evictable-idle-time-millis: 300000
      use-unfair-lock: true
      #用来检测连接是否有效的 sql，要求是一个查询语句,如果 validationQuery 为 null，testOnBorrow、testOnReturn、testWhileIdle 都不会其作用
      validation-query: SELECT 1 FROM DUAL
      #申请连接的时候检测，如果空闲时间大于 imeBetweenEvictionRunsMillis，执行 validationQuery 检测连接是否有效
      test-while-idle: true
      #申请连接时执行 validationQuery 检测连接是否有效，做了这个配置会降低性能
      test-on-borrow: false
      #归还连接时执行 validationQuery 检测连接是否有效，做了这个配置会降低性能
      test-on-return: false
      use-global-data-source-stat: false
      #以下为基于 spring boot web 的内嵌 druid 监控，若需开启请将三个值均置为 true
      filter: stat,slf4j
#      filter:
#        stat:
#          merge-sql: true
#          enabled: true
#        wall:
#          enabled: false
#        slf4j:
#          enabled: true
      stat-view-servlet:
        enabled: true
        login-username: admin
        login-password: admin
        allow:
      web-stat-filter:
        enabled: true
      #初始化物理连接个数
      initial-size: 8
      #最小连接池数量
```

```
    min-idle: 8
    #最大连接池数量
    max-active: 64
    #  配置获取连接等待超时的时间
    max-wait: 30000
    #通过 connection-properties 属性打开 mergesql 功能;慢 SQL 记录
    connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
```

下面是广发银行的连接数的配置：

spring.datasource.druid.keepAlive=true

spring.datasource.druid.maxActive=250

spring.datasource.druid.minIdle=50

5.1.2. 必须使用与数据库服务器端版本一致的 JDBC 驱动

GoldenDB 官方的版本是 ZXCLOUD-GoldenDB-Java-ConnectorV[version]，请使用此版本进行开发。

5.1.3. 减少和数据库的交互次数

有如下方法:

1. 使用 Redis 等缓存技术来减少数据库的交互操作；
2. 使用 addBatch 方式可以大大减少客户端与服务器端的交互次数；
3. join 可以降低数据库的交互次数，但是 join 的表不能超过 2 张；
4. 采用 In List 的方式可以大大降低 SQL 请求的数量；
5. 查询使用 setFetechsize 减少数据准备时间和交互次数。

5.1.4. 使用流式获取执行大结果避免 OOM

5.1.4.1. 多分片

GoldenDB连接说明-JDBC-ODBC.p

5.1.5. SQL 异常处理

- 应用程序必须捕获 SQL 异常，并记录在在日志中。
- 出现异常时，必须回滚事务，并断开连接。

5.2. 读写分离

读写分离，基本的原理是让主数据库处理事务性增、改、删操作(INSERT、UPDATE、 DELETE) ，而备数据库处理 SELECT 查询操作，一方面提高数据库处理性能，另一方面减少读操作对主库的影响。GoldenDB 支持租户级、CN 级语句级别的读写分离，读写分离按照应用需要开启。一般来说对于实时性要求不高并且有大量查询的场景建议开启读写分离，比如银行的向下游供数过程。

关于读写分离的几点说明：

1. 目前只有非事务中，UR 隔离级别的 select 语句才会进行读写分离（开启 CN 级别事务级读写分离除外，下文另外说明）。
2. 读写分离策略由连接实例级别策略+语句级别策略共同决定。在连接实例上配置，可选择的配置类型有：不开启读写分离，本地同城策略，异地策略。其中本地同城策略需要配置权重。开启本地同城策略的情况，不管权重如何配置，都属于 READBALANCE 的场景。
3. 语句级别的读写分离策略，通过在 sql 语句中携带 hint 实现。hint 包括 READMASTER、

READBALANCE、READSLAVE。读写分离策略与语句级别 hint 共同作用下的读写分离参考如下：

	无 hint	READMASTER	READBALANCE	READSLAVE
不开启读写分离	读主 DB	读主 DB	读主 DB	报错
本地同城	根据权重选择主备 DB(备注 1)	读主 DB	根据权重选择主备 DB(备注 1)	根据权重选择备 DB（备注 2)
异地	读异地备 DB(备注 3)	读主 DB	读异地备 DB(备注 3)	读异地备 DB(备注 3)
备注 1：此场景选择 DB 读写分离根据权重权责 DB，受到 DB 状态和 DB 延迟的影响				
备注 2：此场景选择 DB 读写分离根据权重权责 DB，受到 DB 状态和 DB 延迟的影响				
备注 3：此场景选择异地备 DB，受到 DB 状态和 DB 延迟的影响				

4. 由于实际运行中，可能会出现 DB 宕机或者主备 DB 时延超过配置阈值的异常情况，所以实际读语句选择下发的 DB，除了受到读写分离策略的影响，还受到当前系统中 DB 的运行状态影响。
5. 关于本地同城策略下权重的含义。本地同城策略下，可以配置主权重，本地备机权重，同城备机权重的比例，最后生效的是三者的比例而不是填写的具体数值。例如 1：2：1 与 2：4：2 效果完全相同。另外比例指的是读语句在两个地点之间分发的比例，与该地点实际有几个备 DB 无关。举例：配置权重比例为 1：2：3，语句下发到本地备的概率为 1/3，语句下发到同城备的概率为 1/2。在此基础上，继续假设本地备机有 5 个，同城备机有 4 个，每个本地备机的概率为 1/（3*5）=1/15。同理，每个同城备机的概率为 1/（2*4）=1/8。
6. 关于地点的含义。我们说的本地，同城，异地是以主 DB 所在位置为基准确定的，与主 DB 相同机房的备机为本地备机，与主 DB 不同机房相同城市的备机为同城备机，与主 DB 不同城市的备机为异地备机。根据这个定义，发生主备切换后读写分离实际分发情况也会发生改变。举例，假设 room1 是本地机房，room2 是同城机房。那么 room1 的备机就是本地备机，room2 的备机就是同城备机。再假设读写分离配置本地同城策略，权重为 1：2：3，则 room1 的备：room2 的备收到的语句比例为 2：3。此时主机如果从 room1 切换到 room2, room2 的备机变为本地备机，room1 的备机变为同城备机，room1 的备：room2 的备收到的语句比例变为 3：2。

5.3. 事务设计规范

5.3.1. 隔离级别

隔离级别详细介绍:

语句类型	隔离级别	隔离级别说明	适用场景
读语句	UR(uncommitsted read)	未提交读	适用于允许脏读或者不存在读写冲突的业务场景
	CR(consistency read)	强一致性读，先查询活跃 GTILD，后查询数据，严格保证返回结果处于分布布式事务已提交状态，不存在脏读可能	适用于分布式事务中不允许脏读场景
写语句	CW(consistency write)	强一致性写，需要判断分布式写冲突，不允许多个事务同时写相同数据	适用于存在分布式写冲突，但不允许多个事务同时写相同数据场景
	SW(single-node write)	单节点写	适用于单节点集群

1. GoldenDB 写操作使用 CW 隔离级别，读操作默认使用 CR 隔离级别，分布式事务内 SQL 严禁使用 SW 隔离级别。
2. 如果业务场景允许脏读，则推荐使用 UR，可以提升处理性能。比如原语句为 select id from t1，可以改成 select id from t1 UR。
3. 在允许分布式不一致读的大数据量查询场景中，使用 UR 隔离级别。否则在开启参数 max_sql_rows_ret=10000 后，返回数据量超过 1 万条时会报错:EROR 10832（HY000): ERR: result row num，9822092 exceed max:10000!
4. 对于使用游标场景(resultSetType=”FORWARD_ONLY” fetchSize=“1000”），使用UR 隔离级别。

5.3.2. 提交和回滚

- 多个 SQL 组成的一个事务，必须由应用程序控制事务提交，避免隐式自动提交。
- 显式事务中，异常退出前，必须对该事务进行回滚操作并断开连接。

5.3.3. 联机交易必须尽快地提交事务

联机交易必须尽快提交事务，释放锁资源。

5.3.4. 批处理必须避免大事务，必须分批提交

为了避免大事务，批处理程序中必须有分批提交的逻辑。

1. 批处理严禁一条记录提交一次。
2. 高并发的批处理程序，为了减少对联机交易的影响，单次提交记录数限制在 50-200 之间。
3. 并发低或者批量执行期间无联机交易的批处理程序，单次提交记录数不可以超过 1000。

5.4. DDL 规范

5.4.1.【建议】一般地，必须在业务低峰期或者停业期间才允许执行 DDL。

一般情况下，必须在业务低峰期或者停业期间才允许执行 ddl；特殊情况下，比如生产发生性能故障时，可临时紧急增加索引。所有的 ddl 语句需要经过 DBA 审核，执行 ddl 之前需要评估 ddl 执行时间。

5.4.2.【建议】建表分发策略根据业务特征进行选择。

5.4.3.【规则】避免 DDL 语句中存在大量中文注释，避免在执行过程中字符集问题导致错误。

5.4.4.【规则】避免删库删表操作，防止误删数据。

禁止在应用程序中出现 drop table 的业务逻辑；禁止在生产环境下执行删除列的操作。

5.4.5.【规则】不能对库表进行并发 DDL 操作。

5.4.6.【建议】表重组

表经过长时间的删除、插入、更新操作，数据页和索引页变得越来越零散，会产生碎片。碎片较多时，读取有效的数据块效率就会受到影响。相同的查询内容，碎片越多，逻辑读就越高。为提高性能。建议定期进行碎片整理。避免因为“空洞”导致性能问题。

需要定期对碎片率超过 30%的表进行重组，insight 页面->监控统计->表空间监控可以查询碎片率。表重组方式：OPTIMIZE TABLE【table_name】，OPTIMIZE 包含了表重建和索引统计信息更新，是在线操作但不是 DDL，对 DML 操作影响不大。在生产中操作时也需要在业务空闲期时执行。

5.4.7.【规则】DDL 执行失败处理

如果 DDL 失败，需要检查失败原因，并重新执行。以下操作失败后可以重复执行：增加列、增加索引、删除列或者索引、增加分区、删除分区。DDL 变更失败可能会紧表，查明原因且确保 MDS 与 DB 表结构一致后才可以解禁，非以上可以重复执行的操作导致的不一致需要人为做一个反向 DDL。

5.4.8.【建议】DDL 无须指定 CHARSET 和 COLLATE。

参考例子：

```
 <select id="getAll" resultMap="postResultMap" fetchSize="-2147483648">
select * from post,
</select>

CREATE TABLE pdct_param(
param_code vachar(50)  NOT NULL DEFAULT ‘’ COMMENT” ‘参数代码’,
param_class_code vachar(50) NOT NULL DEFAULT  ‘’COMMENT ‘参数分类代码，功能分类代码’,
param_desc  vachar(200) NOT NULL DEFAULT  ‘’ COMMENT ‘参数描述’,
...
ceate_ts datetime(6)  NOT  NULL  DEFAULT CURRENT_TIMESTAMP(6)   COMMENT ‘创建时间’,
update_ts datetime(6)  NOT  NULL  DEFAULT CURRENT_TIMESTAMP(6)  ON UPDATE  CURRENT_TIMESTAMP(6)  COMMENT
‘最后更新时间’,
PRIMARY KEY(param_code)
)COMMENT=’参数表’;
```

注意点：

1. DDL 中只指定 CHARSET=utf8mb4·，默认的 COLLATE 是 utf8mb4_0900_ai_ci，它与通常采用的 utf8mb4_bin 不一致。如果同时不指定 CHARSET 和 COLLATE，建表时就会采用数据库默认字符集和默认排序规则。因此在 DDL 无须指定 CHARSET 和 COLLATE。
2. 若指定了 CHARSET=utf8mb4，必须指定 COLLATE=utf8mb4_bin。

5.4.9.【建议】分区表的创建和添加分区要控制一次性添加的数量，不能一次性添加过多的分区，防止出现打开文件数过多从而影响数据库的正常使用。参考统一监管和 crm 数据库，因一次性添加分区太大导致 too many open files 错误。

5.5. SELECT 规范

5.5.1.【规则】避免使用 select *方式全字段查询数据，按业务需要捞取特定字段

说明：使用通配符字段查询表时，如果因业务或数据库升级导致表结构发生变化，可能出现与业务语句不兼容的情况。因此业务应指明所需查询的表字段名称，避免使用通配符。其次查询不必要的字段，容易产生性能问题及造成不必要的网络传输数据。

5.5.2.【规则】避免对大字段（如 VARCHAR(2000)）执行 ORDER BY、DISTINCT、GROUP BY、UNION、on 关联等会引起排序的操作

说明：此类操作性能比较低下，其次内部对 order by/group by 等排序长度有配置限制。

5.5.3.【建议】避免在 SELECT 目标列中使用子查询，可能导致计划无法下推到 DN 执行，影响执行性能

举例 1(t2.id 为唯一键)：

```
select (select t2.id1 from t2 where t2.id = t1.id) as col1 from t1 where id1 > 100;
```

建议写法：

select t2.id1 as col1 from t1 left join t2 on t1.id = t2.id where t1.id1 > 100;

举例 2：

select (select max(t2.id1) from t2 where t2.id = t1.id) as col1 from t1 where id1 > 100;

建议写法:

select T.max_id1 as col1 from t1 left join (select max(t2.id1) max_id1，id from t2 group by id)T on t1.id = T.id where t1.id1 > 100;

5.5.4.【规则】避免在 SELECT 语句全表大数据量进行聚合函数操作，消耗资源大、性能差

举例 1：select count(1) from t1;

举例 2：select count(1) from t1 group by id1;

5.5.5.【规则】避免 select 目标列的别名与表名相同

举例：select id+1 as id from t1 where id > 1;

说明：id 别名与实际表中的 id 名冲突，容易产生理解上歧义

5.5.6.【规则】避免 select 中 order/group by 字段不指定表名

举例 1：select id1+1 as id from t1 where id1 > 10 group by id;

说明：group by 中的 id 容易产生歧义

举例 2：select id1+1 as id from t1 where id1 > 10 order by id;

说明：order by 中的 id 容易产生歧义

5.5.7.【规则】select 多表 join 时，输出列存在与多个表中时，需要指定表名

举例：select id from t1 join t2 on t1.id = t2.id where t1.id1 < 10;

说明：输出列中的 id 字段，t1 与 t2 表中均有 id 字段，此时需要指定表名，否则默认取的是第一个表 t1 的

5.5.8.【高危】避免 select 中存在大结果集的 exists/not exists/in/not in 子查询到 CN 节点计算，容易消耗 CN 节点大量内存资源

举例 1：select 1 from t1 where exists(select 1 from t2 where t1.id = t2.id);

举例 2：select 1 from t1 where not exists(select 1 from t2 where t1.id = t2.id);

建议写法：select 1 from t1 left join t2 on t1.id = t2.id where t2.id is null;

举例 3：select 1 from t1 where t1.id in(select id from t2);

举例 4：select 1 from t1 where t1.id1 not in(select id from t2);

说明：exists 子查询结果集很大，即 select 1 from t2 查询结果集很大，容易消耗大量内存进行关联计算。

5.5.9.【建议】select 查询中，在不改变语义情况下，建议将 rownum 改成 limit，rownum 是 oracle 语法，不推荐 oracle 语法和 mysql 语法混用，改用 limit

举例 1：select 1 from t1 where id1 < 100 and rownum < 100;

建议写法：select 1 from t1 where id1 < 100 limit 100;

举例 2：select * from(select id， id1，rownum rn from t1 where id1 < 100 order by id)T where T.rn < 10;

建议写法：select id， id1， rownum rn from t1 where id1 < 100 order by id limit 10;

举例 3：select (select t2.id from t2 where t2.id = t1.id and rownum < 2) as col1 from t1 where id1 < 10;

建议写法：select (select t2.id from t2 where t2.id = t1.id limit 1) as col1 from t1 where id1 < 10;

5.5.10. 【建议】select 查询中，避免套多层无效 from 子查询，影响执行效率

举例 1：select * from(select id， id1，rownum rn from t1 where id1 < 100 order by id)T where T.rn < 10;

建议写法：select id， id1， rownum rn from t1 where id1 < 100 order by id limit 10;

举例 2：select * from(select id， id1 from t1 where id1 < 10)T order by T.id;

建议写法：select id， id1 from t1 where id1 < 10 order by id;

5.5.11. 【建议】select 查询中，尽量将过滤条件提前

举例：select * from(select id， max(id1) from t1 where id2 > 100 group by id)T where T.id < 10;

建议写法：select * from(select id，max(id1) from t1 where id2 > 100 and id < 10 group by id)T;

5.5.12. 【建议】建议业务不要查询大结果集

5.5.13. 【禁止】禁止大表不带查询条件直接查询

举例:

```
SELECT
CASE
        batch_ln_acct_detail.tx_bal_type
        WHEN '1' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '2' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '3' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '4' THEN
        batch_ln_acct_detail.tx_amt ELSE 0
    END AS '本金变动金额',
CASE
        batch_ln_acct_detail.tx_bal_type
        WHEN '5' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '6' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '7' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '8' THEN
        batch_ln_acct_detail.tx_amt
        WHEN '9' THEN
        batch_ln_acct_detail.tx_amt ELSE 0
    END AS '欠息变动金额',
    batch_ln_acct_detail.tx_trace_no AS '业务流水号',
    batch_ln_acct_detail.tx_date AS '发生日期',
    batch_ln_acct_detail.tx_bal_type AS '变动类型',
    batch_ln_acct_detail.dc_flag AS '交易标识',
    batch_ln_acct_mast.receipt_no AS iouNumb,
    batch_ln_acct_detail.id
FROM
    batch_ln_acct_mast_all batch_ln_acct_mast
    INNER JOIN batch_ln_acct_detail_b batch_ln_acct_detail ON batch_ln_acct_mast.legal_entity =
batch_ln_acct_detail.legal_entity
    AND batch_ln_acct_mast.ln_ac_id = batch_ln_acct_detail.ln_ac_id
    AND batch_ln_acct_mast.del_ind = '0'
    AND batch_ln_acct_detail.del_ind = '0'
WHERE
    1 = 1
ORDER BY
    batch_ln_acct_detail.tx_date DESC,
```

batch_ln_acct_detail.id DESC LIMIT 8200000, 50000
存在问题： 1 大表没有查询条件
建议： 1 因限制时间范围查询，禁止大表没有查询条件，禁止大量数据返回。

5.5.14. 【高危】不要捞取大量数据到 CN 层进行二次计算，容易阻塞同链路的后续语句

5.5.15. 【禁止】不允许 exists /not exists 关联条件为非等值字段关联

举例 1：select 1 from t1 where exists/not exists(select 1 from t2 where t1.id = t2.id +1);

举例 2：select 1 from t1 where exists/not exists(select 1 from t2 where t1.id > t2.id);

举例 3：select 1 from t1 where exists/not exists(select 1 from t2 where t1.id = t2.id and t1.id > 2);

5.5.16. 【禁止】不允许 exists /not exists 跨层关联条件

举例：select 1 from t1 where exists/not exists(select 1 from(select 1 from t2 where t2.id = t1.id));

5.5.17. 【禁止】不允许 exists /not exists 关联子查询为 union/union all

举例：select 1 from t1 where exists/not exists(select 1 from t2 where t2.id = t1.id union

select 1 from t3 where t3.id = t1.id);

5.5.18. 【禁止】不允许 exists /not exists 关联子查询中 where 条件中含有 or 条件

举例 1：select 1 from t1 where exists/not exists(select 1 from t2 where t2.id = t1.id or t2.id = t1.id1);

举例 2：select 1 from t1 where exists/not exists(select 1 from t2 where t2.id = t1.id and (t2.id1 = 3 or t2.id1 = 4));

5.5.19. 【禁止】不允许 exists /not exists 关联子查询多表 join

举例：select 1 from t1 where exists/not exists(select 1 from t2 join t3 on t2.id = t3.id where t2.id = t1.id);

5.5.20. 【禁止】不允许 exists /not exists 关联子查询中使用 start with

举例：select 1 from t1 where exists/not exists(select 1 from t2 where t2.id = t1.id start with t2.id1 = 1 connect by t2.id2=t2.id1);

5.5.21. 【禁止】不允许在 select 输出列中使用 exists /not exists 关联子查询

举例：select (case when exists/not exists(select 1 from t2 where t1.id = t2.id) then 1 else 2 end as col1 from t1 where id1 < 10;

5.5.22. 【禁止】标量子查询关联条件不允许非字段非等值关联

举例 1：select (select t2.id from t2 where t2.id = t1.id + 1) as col1 from t1 where id1 < 100;

举例 2：select (select t2.id from t2 where t2.id < t1.id) as col1 from t1 where id1 < 100;

5.5.23. 【禁止】标量子查询输出列不允许表达式

举例 1：select (select t2.id+1 from t2 where t2.id = t1.id) as col1 from t1 where id1 < 100;

举例 2：select (select abs(t2.id) from t2 where t2.id = t1.id) as col1 from t1 where id1 < 100;

5.5.24. 【禁止】标量子查询不允许跨层关联

举例：select (select id from(select id from t2 where t2.id = t1.id)) as col1 from t1 where id1 < 100;

5.5.25. 【禁止】标量子查询中 where 条件不允许 OR 条件

举例 1：select (select id from t2 where t2.id = t1.id or t2.id is null) as col1 from t1 where id1 < 100;

举例 2：select (select id from t2 where t2.id = t1.id and (t2.id is null or t2.id1 > 1) as col1 from t1 where id1 < 100;

5.5.26. 【禁止】标量子查询中不允许 union/union all

举例：select (select id from t2 where t2.id = t1.id union/union all select id from t3 where t3.id = t1.id) as col1 from t1 where id1 < 10;

5.5.27. 【禁止】标量子查询中不允许多表 join

举例：select (select t2.id from t2 join t3 on t2.id = t3.id where t2.id1 = t1.id) as col1 from t1 where id1 < 10;

5.5.28. 【禁止】标量子查询中不允许 start with

举例：select (select t2.id from t2 where t2.id = t1.id start with t2.id1 = 1 connect by t2.id2=t2.id1) as col1 from t1 where id1 < 100;

5.5.29. 【规则】order by 排序，默认认为 NULL 值为小值，若需要特定 NULL 值顺序，显式写上 NULLS first/last

举例：

insert into t1(id，id1) values(1，1)，(2，2)，(3，NULL);

select id from t1 where id1 > 100 order by id1;

结果：

3

1

2

5.5.30. 【禁止】union/union all 全局 group by/order by 不允许非 union/union all 输出列 group by/order by

举例 1：select id， id1 from t1 union/union all select id， id1 from t2 group by id+1;

举例 2：select id， id1 from t1 union/union all select id， id1 from t2 order by id+1;

举例 3：select id， id1 from t1 union/union all select id， id1 from t2 order by id2;

举例 4：select id， id1 from t1 union/union all select id， id1 from t2 group by id2;

5.5.31. 【禁止】不允许 some/any/all 子查询

5.5.32. 【禁止】不允许常量 in/not in 子查询

举例 1：select 1 from t1 where 22 in/not in(select id from t2 where id1 > 1);

举例 2：select 1 from t1 where 22 in/not in(select id1 from t2 where t2.id = t1.id);

5.5.33. 【禁止】不允许 in/not in 关联子查询

举例：select 1 from t1 where t1.id1 in/not in(select id1 from t2 where t2.id = t1.id);

5.5.34. 【禁止】不允许 with as 递归查询

举例：

```
with hgo as
(
    select id，id1，name from t1 where id1 = 1
    union all
    select h.id，h.id1，h.name from t1 h join hgo h1 on h.id=h1.id1
)
select * from hgo;
```

5.5.35. 【禁止】不允许 sequence 与聚合函数、子查询同在输出列

举例 1：

select seq.nextval， sum(id) from t1 where id1 < 10 group by id2;

举例 2：

select seq.nextval， (select 1 from dual) as col1 from t1 where id1 < 10;

举例 3：

select * form (select seq.nextval， sum(id) from t1 where id1 < 10 group by id2)T;

举例 4：

select * from(select seq.nextval， (select 1 from dual) as col1 from t1 where id1 < 10)T;

5.5.36. 【规则】分页查询使用规则

当处理大量数据时，一次性查询所有数据会导致性能下降、服务器响应时间变长，内存溢出等问题。而使用分页查询，可以将查询结果分割成多个部分，每次只查询部分数据，从而提高性能、减少内存消耗、提高用户体验和数据安全性。通过分页查询，不仅可以提高前端展示的效率，减少页面加载时间和网络传输的数据量，也可以优化后端查询的效率，提高系统的并发性能和稳定性。因此，在开发中，我们应该充分利用分页查询，尽可能减少一次查询数据的量，以达到优化系统性能的目的。

分页语句写法(start：起始记录数，page_offset：每页记录数)：SELECT class_num，ID，username FROM student WHERE username like 'Mike%' ORDER BY class_num LIMIT start，page_offset;

当表中数据非常大时，要求使用查询主键的方式进行分页；将主键先进行排序并全部取出，每次取主键区间的边缘值，例如：select p_id，..... from t1 where p_id（或代理主键）>=709995 and p_id <=7100999 order by p_id;

5.5.37. 【禁止】不允许@变量赋值

举例 1：select :@ = 1;

禁止原因

分布式不支持。

5.6. INSERT 规范

5.6.1.【规则】insert values 时，分片键字段必须为常量值(不能为表达式)

举例：insert into t1(id， id1 ，id2) values(1+1， 1， 1);

说明：其中的 id 列为分片键字段，存在表达式影响分片键计算分片

5.6.2.【规则】禁止使用 INSERT ON DUPLICATE KEY UPDATE

说明：ON DUPLICATE KEY UPDATE 操作需要检测唯一约束冲突检测，影响插入性能;

其次存在多个唯一约束时，更新逻辑较为复杂，容易产生与业务预期不符值更新。若确实需要更新，可以考虑使用 merge into 语法。

5.6.3.【建议】insert select 时不要使用 select *，避免数据库做*展开操作

5.6.4.【建议】使用 executeBatch 方式或者 insert 多 values 方式提升插入性能

举例：insert into t1 values(1，1，1)，(2，2，2)，(3，3，3) ...，(m，m，m);

说明：使用上例中多 values 的方式替代单 values 多次插入的方式，提升插入性能。也可以使用 executeBatch 方式，即批量插入的方式提升插入性能。

5.6.5.【禁止】insert values 时，values 值不支持子查询

举例：insert into t1 values(1，1，(select abs(id) from t2 where id1=1));

5.6.6.【建议】尽量避开使用 replace 语句

说明：replace 存在唯一键冲突替换的问题，分布式需要做唯一键冲突检测，影响性能。

5.7. UPDATE 规范

5.7.1.【建议】联机交易 UPDATE 必须根据主键或唯一索引进行更新

5.7.2.【禁止】禁止更新主键列或者唯一索引列

5.7.3.【建议】禁止使用大事务

更新的规范需遵守 WHERE 子句的规范，对大表进行更新，要避免大事务，避免主从同步延迟加大。建议单事务不超过 200M。

5.7.4.【禁止】禁止在 UPDATE 操作的 SET 子句出现 AND 连接符号

如果把 SET 子句的连接符号“，”，错误的写成“AND”，将会导致语意完全错误，所以禁止在 SET 子句中出现 AND 连接符号字样。

5.7.5.【禁止】禁止对分片字段进行修改，特殊情况需要向 DBA 申请

修改分片字段会导致数据在数据分片间迁移，可以理解为在新数据分片进行 INSERT，在原数据分片进行 DELETE。可以简单使用 delete+insert 语句替代。

5.7.6.【禁止】禁止在 UPDATE 语句中同时更新两张或以上表

单语句需要拆分为多语句，SQL 语义更清晰。

5.8. DELETE 规范

5.8.1.【建议】避免在 delete 语句中使用 limit，delete 记录需要明确删除的记录

举例：delete from t1 where id1 > 10 limit 10;

5.8.2.【建议】避免在 delete 语句中使用无意义的 order by

举例：delete from t1 where id1 > 10 order by id;

5.8.3.【建议】delete 语句中 where 条件，尽量使用索引/主键过滤率高的过滤条件

5.8.4.【规则】delete 语句中必须带 where 过滤条件，避免全表扫描删除

举例：delete from t1;

5.8.5.【规则】若需清空表记录，使用 truncate 性能更好

5.8.6.【规则】使用删除分区的方式清理过期数据

5.8.7.【禁止】不支持多表删除，即在 delete 语句中同时删除多张表

5.9. WHERE 子句

5.9.1. 访问分片表

高并发 SQL 必须带分片键。

错误案例:

```
DELETE FROM x.repay_plan  WHERE  loan_acct_no='100001' AND repay_type=’I’ AND term_count=2;
```

正确例子:

```
DELETE FROM x.repay_plan  WHERE part_no=xxx and  loan_acct_no=’10000’°AND repay_type=’I’°AND term_count=2;
```

5.9.2. 访问分区表

联机交易会发生并发访问的 SQL 中访问分区表，必须在查询中增加分区限制条件来限制访问的分区。

5.9.3. 联机交易的 SQL 禁止使用全表扫描

5.9.4. 联机交易的 SQL 禁止使用全索引扫描

禁止原因

由于联机交易，一般处理的记录数很少，索引很适合优化联机交易。当执行计划是全索引扫描，当表数据量很大时，全索引扫描导致高开销的数据库磁盘 IO。除非该表是系统配置表，数据量很小，记录行数小于 1000 行。

优化方案

建立高效索引。

5.9.5. 禁止多个通过 and 连接的条件只能匹配组合索引中的部分选择性低的列

禁止原因

索引效率低，需要读取更多的索引页和数据页，IO 和 CPU time 都会增加，导致服务器资源压力大，响应时间变长。

示例

```
select
id，part_no，mib_ac_no，tr_datetime，vch_no，vch_no_seq，br_no，ccy，...
from acct_online_mib_detail
where mib_ac_no= ‘1000000001’
and vch_no=’00039’
and vch_no_seq=101
and br_no=’1731’
and tr_datetime >=’2020-12-20 00:00:00’
and tr_datetime<’2020-12-21 00:00:00’
and tr_timestamp >=’2020-12-20 00:00:00’
and tr_timestamp <’2020-12-21 00:00:00’
Limit 1 \G
```

原索引(`mib_ac_no`，`tr_datetime`，`vch_no`，`vch_no_seq`，`br_no`，`tr_timestamp`)违背了索引列先后顺序原则--即在索引中，谓词是“=”的列放在索引最左侧，范围(>=，>，<=，<) 谓词列放在索引右侧，导致 vch_no 无法匹配索引。

优化

索引修改为(`mib_ac_no`，`vch_no`，`vch_no_seq`，`br_no`，`tr_datetime`，`tr_timestamp`)。

5.9.6. 禁止 or 谓词应用在带有分片键列的谓词

禁止原因

会导致多次下发到所有的分片。

示例

```
select * fom t1 where
(loan_acct no ='1234' and record_prop ='1234' and part_no =1 )
or
(loan acct no ='2345' and record_prop ='1234' and part_no =2 );
```

优化方案 1

SQL 改写为:

```
select * fom t1 where
(loan_acct_no ='1234' and record_prop ='1234' and part_no =1 )
Union all
  select * from t1 where
(loan_acct_no ='2345' and record_prop ='1234' and part_no =2 );
```

代化方案 2

SQL 改写为:

```
select * from t1 where
(loan_acct_no, record_prop, part_no) in (('1234', '1234', 1)('2345', '1234', 2));
```

5.9.7. 不能匹配索引的场景

5.9.7.1. 禁止对索引字段进行数学计算

原 SQL 示例

表 card_detail 有索引(ac，update_time)，有如下查询:

```
select ac, amt
from card_detail
where ac=? and ( now() - update_time <3600)
```

优化 SQL 示例

```
select ac, amt
from card_detail
where ac=? and ( update_time < now()-3600)
```

5.9.7.2. 禁止对索引列使用函数

原 SQL 示例

表 card_detail 有索引(ac，update_time)，update_time 为 datetime 数据类型，有如下查询:

```
sselect ac, amt
from card_detail
where ac=? and date(update_time)= '2010-01-01'
```

优化 SQL 示例

SQL 修改为

```
select ac, amt
from card_detail
Where ac=? and
  update_time>= '2010-01 -01 00:00:00'  and update_time< '2010-01-02 00:00:00';
```

5.9.7.3. 禁止隐式数据类型转换

原 SQL 示例

假设字段 card_detai.card_no 的类型为 varchar(20)，card_no 是索引列。

```
select ac，amt
from card_detail
where card_no=12345678
```

优化 SQL 示例

该 SQL 应该修改为

```
select ac，amt
from card_detail
where card_no='12345678'
```

5.9.7.4. 禁止使用前缀进行模糊前缀查询

在使用 LIKE 关键字进行查询时，如果匹配字符串的第一个字符为”%”，不能匹配索引。只有”%”不在第一个位置，才能匹配索引。

原 SQL 示例

避免使用:

```
select id，val
from table
where val like '%name';
```

优化 SOL 示例

可以使用%模糊后缀查询，如:

```
select id，val
from table
where val like 'name%';
```

5.9.7.5. 禁止 OR 子句中的列没有全部索引

对于 OR 子句，如果要利用索引，则 OR 之间的每个条件列都必须用到索引；如果没有索引，则应该考虑增加索引。

原 SQL 示例

例如综合收单的历史查询 sql:

```
SELECT
```

```
    OUT_TRADE_ID,
    SETTLE_DATE,
    TRANS_DATETIME,
    END_DATETIME,
    ORIG_TRANS_DATETIME,
    SETTLE_DATETIME,
    MCHNT_CD,
    ACTUAL_AMOUNT,
    API_CHANNEL,
    AUTH_CD,
    BAK1,
    BAK2,
    BANK_ACCOUNT_SUPPLY_STATUS,
    BANK_ACTIVITY_ID,
    BANK_DISCOUNT,
    BANK_FULL_REDUCTION
FROM
    TBL_TRANS_HST
WHERE
    1 = 1
    AND MCHNT_CD = '463428182490018'
    AND ( TRACE_NO = '23092204071A00026831' OR GLB_SEQ_NO = '23092204071A00026831' OR TRADE_ID =
'23092204071A00026831' )
ORDER BY
    TRANS_DATETIME DESC
    LIMIT 1;
```

优化 SQL 示例

TRACE_NO = '23092204071A00026831' OR GLB_SEQ_NO = '23092204071A00026831' OR TRADE_ID = '23092204071A00026831' 这部分 sql 因为 TRADE_ID 字段没有索引所以无法走到这 TRACE_NO 、GLB_SEQ_NO、TRADE_ID 三个字段的索引，导致走了 order by 索引，由于 where 条件的筛选条件在表里没有匹配记录导致查询时间特别长。应增加索引 IDX_T1_TRADE_ID(TRADE_ID)。

5.9.7.6. 禁止 IN 里面列表的个数太多

IN 列表数太多会导致全表扫描。必须控制 IN 列表的个数在 200 个之内。

5.10. TRUNCATE 规范

5.10.1. 使用 truncate 注意事项

Truncate 会删除表的所有数据，使用前要谨慎，以免操作失误。

5.10.2. 使用 truncate 删除表数据后，如果对表重新进行了数据写入或者导入，一定要使用 ANALYZE 对该表进行表统计信息收集！在信贷、crm、流动性、核算等都遇到此类问题，尤其批处理的过程中，truncate 操作后一定要考虑是否需要表统计信息收集。

5.11. 关于批处理设计指导

1、为充分利用分布式数据库的特性，请按分片进行批处理，通过 SHOW DISTRIBUTION FROM【table _name】table 获取表分片信息后，再通过多个线程使用 STORAGEDB 语法针对多个分片并发处理。

2、批处理涉及大数据量查询，请通过游标方式先把数据的主键查询到本地，在本地确定数据分块的上下边界后，通过使用主键范围作为 WHERE 条件分批次查询数据。

第六章 数据导入导出规范

6.1. 使用 loadserver 导入数据

使用 GoldenDB 的 loadsever 可以将大文件自动拆分为多个小文件，避免大事务。详细使用说明请参考以下文档：

LDS使用说明.pdf

6.2. 采用 dbtool 卸数

数据库的导出不能使用 select 查询出来进行字符拼接方式，这种方式容易导致内存溢出。Goldendb 分布式数据库中卸数推荐使用 dbtool 工具进行数据导出。

1、dbtool 数据导入

dbtool -loadserver -V -type="in" -clusterid=42 -skip-ok-check -loadin-support-null=1 -db-conn-info="10.20.170.82:6607" -user=dbproxy -password='passwd' -sql="load data infile '/backup/cust_fl_index_data.data' into table credit_sit_d.cust_fl_index_data CHARACTER SET utf8mb4 fields terminated by ',' optionally enclosed by '\"' lines terminated by '\n' ;"

注意：当源文件中含有生僻字时，需在导数命令里加上 CHARACTER SET utf8mb4，防止出现导入错误。

2、dbtool 数据导出

dbtool -loadserver -V -type="out" -clusterid=46 -clustermode=single -sql="select ITMODSID,ITEMCODE,ITEMNAME from gzyhoric.t21_kri_basic_item_ods into outfile '/ftpORIC/data35/t21_kri_basic_item_ods.data' fields terminated by '|' optionally enclosed by '\"' lines terminated by '\n';" -user=gzyhoric -password='passi23' -host='10.20.181.162:5012'

6.3. 增量卸数

对于大表应根据业务需求尽量采用增量卸数减少卸数时间。

第七章 附录

7.1. 关键字

保留关键字
ACCESSIBLE
ADD
ALL
ALTER
ANALYZE
AND
AS
ASC
ASENSITIVE
BEFORE
BETWEEN
BIGINT
BINARY
BLOB
BOTH
BY

CASE
CHANGE
CHECK
CHGDISKEY
CLOB
COLLATE
COLUMN
CONDITION
CONNECT
CONSTRAINT
CONTINUE
CONVERT
CR
CREATE
CROSS
CURRENT_DATE
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURRVAL
CURSOR
CW
DATABASES
DAY_HOUR
DAY_MICROSECOND
DAY_MINUTE
DAY_SECOND
DEC
DECIMAL
DECLARE
DECODE
DEFAULT
DELAYED
DELETE
DESC
DETERMINISTIC
DISTINCT
DISTINCTROW
DISTRIBUTED
DIV
DOUBLE
DROP
DUAL
EACH
ELSE

ELSEIF
EMPTY
ENCLOSED
ESCAPED
EXCEPT
EXISTS
EXIT
FETCH
FLOAT
FLOAT4
FLOAT8
FOR
FORCE
FOREIGN
FROM
FULLTEXT
GRANT
GROUP
GROUPS
HAVING
HIGH_PRIORITY
HOUR_MICROSECOND
HOUR_MINUTE
HOUR_SECOND
IF
IGNORE
IN
INCREMENT
INDEX
INNER
INSENSITIVE
INSERT
INT
INT1
INT2
INT3
INT4
INT8
INTEGER
INTERVAL
INTO
IS
ITERATE
JOIN
KEEPALIVE
KEY

KEYS
KILL
LEADING
LEAVE
LEFT
LEVEL
LIKE
LIMIT
LINEAR
LOAD
LOCALTIME
LOCALTIMESTAMP
LOCK
LONG
LONGBLOB
LONGTEXT
LOOP
LOW_PRIORITY
MASTER_SSL_VERIFY_SERVER_CERT
MATCH
MAXVALUE
MEDIUMBLOB
MEDIUMINT
MEDIUMTEXT
MIDDLEINT
MINUTE_MICROSECOND
MINUTE_SECOND
MOD
MODIFIES
MULTI_STEP_QUERY
NATURAL
NEXTVAL
NO_WRITE_TO_BINLOG
NOCACHE
NOCYCLE
NOGTID
NOMAXVALUE
NOMINVALUE
NOORDER
NOT
NULL
NULLCOLS
NUMERIC
OF
ON
OPTIMIZE

OPTIONALLY
OR
ORDER
OUT
OUTFILE
OVER
PAGE_CHECKSUM
PARSE_VCOL_EXPR
PARTITION
PARTITION
PLS_INTEGER
PRIMARY
PROCEDURE
PUBLIC
PURGE
RANGE
READ
READ_WRITE
READBALANCE
READMASTER
READS
READSLAVE
REAL
REDISTRIBUTING
REFERENCES
REGEXP
RENAME
REPEAT
REPLACE
REQUIRE
RESIGNAL
RETURNING
REVOKE
RLIKE
ROWNUM
SAMEDB
SCHEMAS
SECOND_MICROSECOND
SELECT
SEMI
SENSITIVE
SEPARATOR
SEQUENCES
SET
SHOW
SIGNAL

SMALLINT
SPATIAL
SPECIFIC
SQL
SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT
SQLEXCEPTION
SQLSTATE
SQLWARNING
SSL
STARTING
STORAGEDB
STRAIGHT_JOIN
SUBDISTRIBUTED
SW
SYNONYM
SYSDATE
TABLE
TERMINATED
THEN
TINYBLOB
TINYINT
TINYTEXT
TO
TRAILING
TRIGGER
UNDO
UNION
UNIQUE
UNLOCK
UNSIGNED
UPDATE
UR
USE
USING
UTC_DATE
UTC_TIME
UTC_TIMESTAMP
VALUES
VARBINARY
VARCHAR
VARCHAR2
VARCHARACTER
VARYING
WHEN

WHERE
WHILE
WINDOW
WITH
WRITE
WRITE_SAMEDB
XOR
YEAR_MONTH
ZEROFILL
FALSE
TRUE