水平分表使用手册

一、水平分表

当系统使用到一定阶段，有些表单会累积大量的数据，对应的数据库表行数超多，影响查询使用。

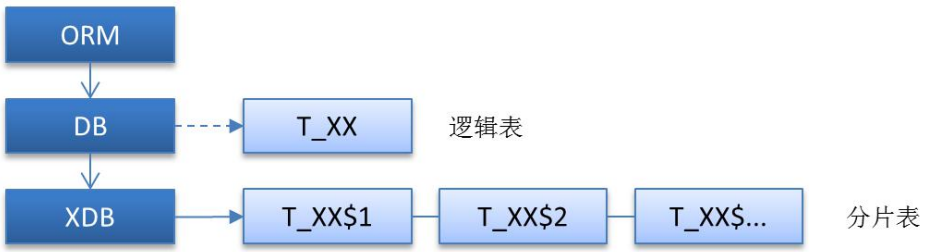
水平分表是自主研发的针对苍穹产品的库内水平分表，是解决在线单表数据行过大的方案，把表数据拆分到当前数据库的多个物理分表中存储，数据行按指定列的规则分布到各分片表中。

分表所依据的列，叫“**分片属性**”，列的规则即对列数据的值按一定的算法计算出分片表索引，这种计算规则叫“**分片策略**”。

水平分表是在 SQL 层支持的，对业务代码无入侵性，表单分表与不分表，不影响功能的使用，仅对性能有影响，在系统运行期动态分表。分表主要是优化查询性能，根据分片条件定位到少量的分片表进行查询。

在系统中，可以动态对表单进行分表，设置好**分片配置**，启用分片将执行在线数据迁移，迁移完毕即完成了分表，在迁移过程中对应的表单数据不可访问。

在编写应用程序时，依旧使用原始表名(逻辑表)进行访问，数据访问的组件如下：



二、功能简介

菜单路径：**系统服务云-->配置工具-->水平分表**，含三类功能：分片配置、分片操作日志、分片辅助(数据统计)。

1.分片配置

配置表单的分片属性、分片策略及策略参数。启用分表后，系统会自动做数据迁移，即把原始的表数据拆分到分片表中。数据迁移过程中，其表单数据是不可访问的，系统会有提示信息。

2.分片操作日志

是每次数据迁移操作(分片、恢复)的日志，用于记录过程中出现的异常信息，如服务节点无响应(宕机)。

3.分片辅助

提供辅助功能，如数据统计、运行期的分片指标采集。

三、分片配置操作

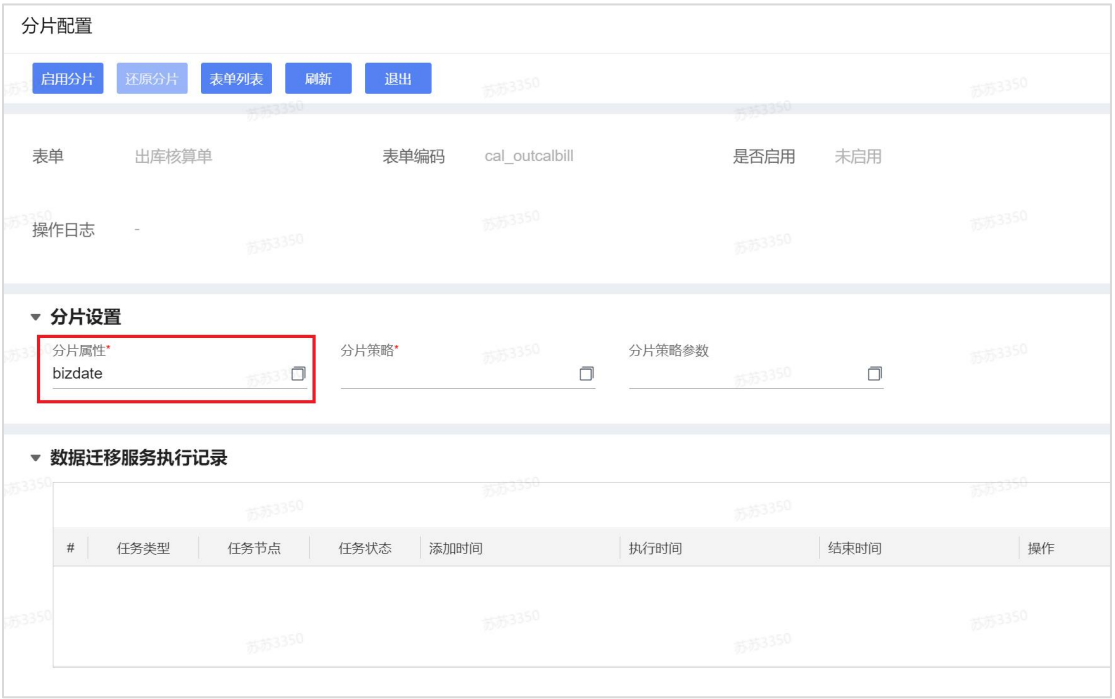
1.分片操作步骤：

1) 打卡分片配置(列表)，点击**新增**。
2) 弹出对话框，**选择需要分片的表单**，确认。
3) 进入分片配置主界面，依次设置：**分片属性、分片策略、分片策略参数**。
4) 点击**启用分片**，将弹出分片任务，系统执行数据迁移。
5) 迁移完毕，则分表完成，表单可正常使用。

2.操作示例图如下：

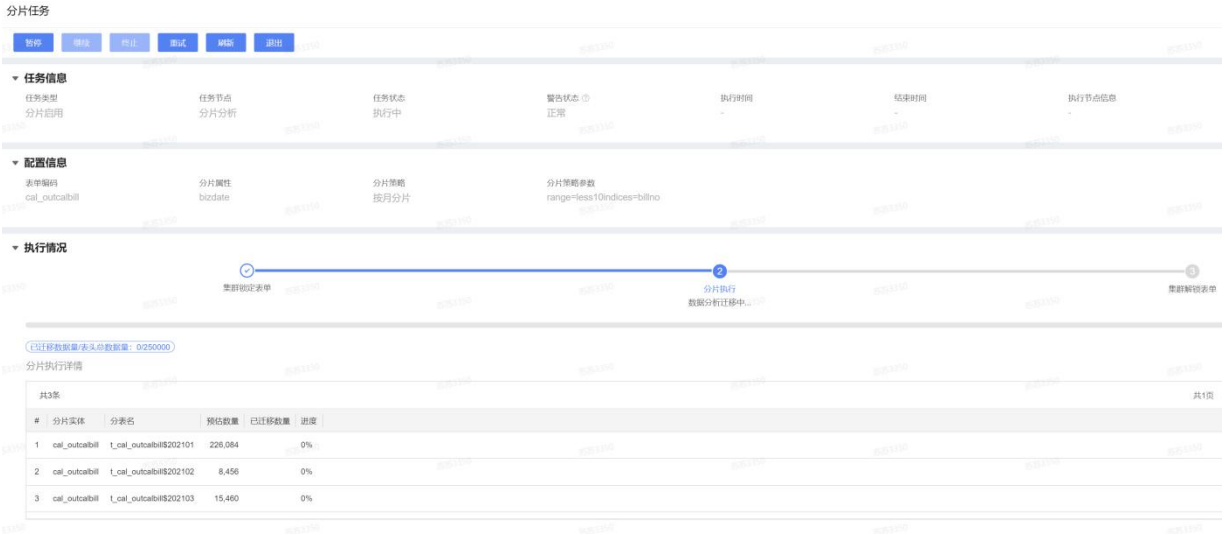

金蝶云 苍穹
首页
应用
配置工具
分片配置列表
分片配置
新增
删除
刷新
表结构校验
退出
共4条
表单名称 表单编码 分片策略 分片属性
1 核算成本记录 cal_costrecord_subentity 映射策略 costaccount,bizdate
2 直接调拨单 im_transdirbill 按月分片 biztime
3 采购入库单 im_purinbill 按月分片 biztime


新增分片配置
表单名称*
出库核算单
表单编码
cal_outcalbill
取消
确认


分片配置
启用分片
还原分片
表单列表
刷新
退出
表单
出库核算单
表单编码
cal_outcalbill
是否启用
未启用
操作日志
-
分片设置
分片属性*
bizdate
分片策略*
分片策略参数
数据迁移服务执行记录
任务类型 任务节点 任务状态 添加时间 执行时间 结束时间 操作


选择分片策略
按日分片
分片规则：分片属性值按yyMMdd格式化
示例源表：t_gl_voucher
分表后：t_gl_voucher$200317、t_gl_voucher$200318、t_gl_voucher$200319...
按月分片
分片规则：分片属性值按yyyyMM格式化
示例源表：t_gl_voucher
分表名：t_gl_voucher$202003、t_gl_voucher$202004、t_gl_voucher$202005...
按年分片
分片规则：分片属性值按yyyy格式化
示例源表：t_gl_voucher
分表名：t_gl_voucher$2020、t_gl_voucher$2021、t_gl_voucher$2022...
日期Hash分片
分片规则：分片属性值格式化后哈希值取模
示例源表：t_gl_voucher
分表名：t_gl_voucher$0、t_gl_voucher$1、t_gl_voucher$2...
映射策略
分片规则：值映射，详细参见文档。
示例源表：t_gl_voucher
分表名：t_gl_voucher$0、t_gl_voucher$1、t_gl_voucher$2...
取消
确定


设置分片参数
数据量估算*
0~10亿行
快速索引
billno



3. 数据迁移在后头服务执行，界面可以关闭，下次进入界面：**分片配置列表-->分片配置-->数据迁移服务执行记录-->查看详情**。

4. 迁移过程，可以操作：

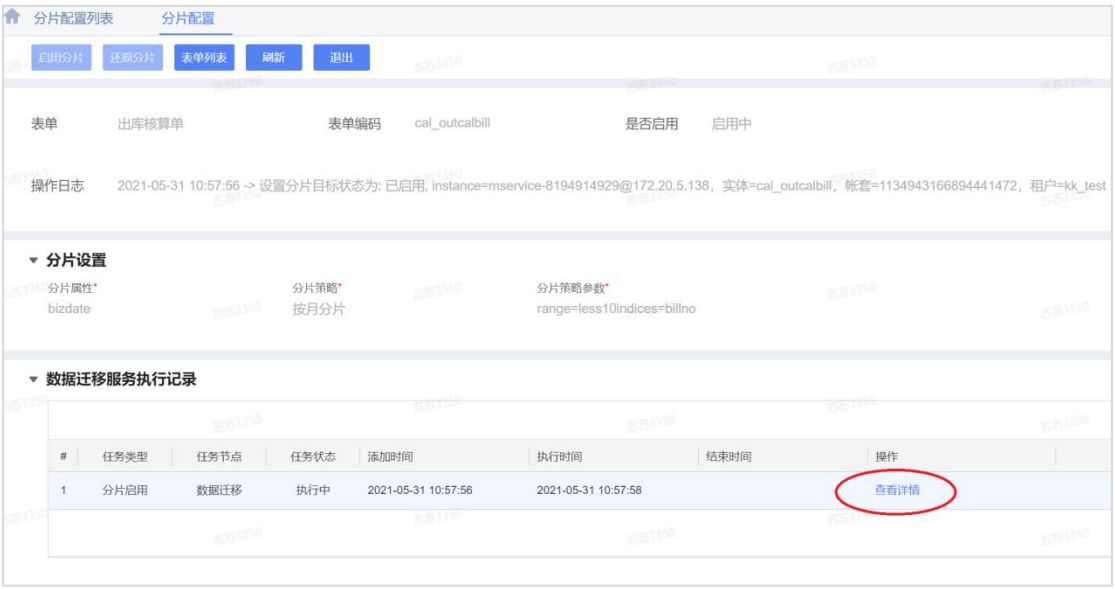
   1) **终止**：暂停-->终止，即取消分片。

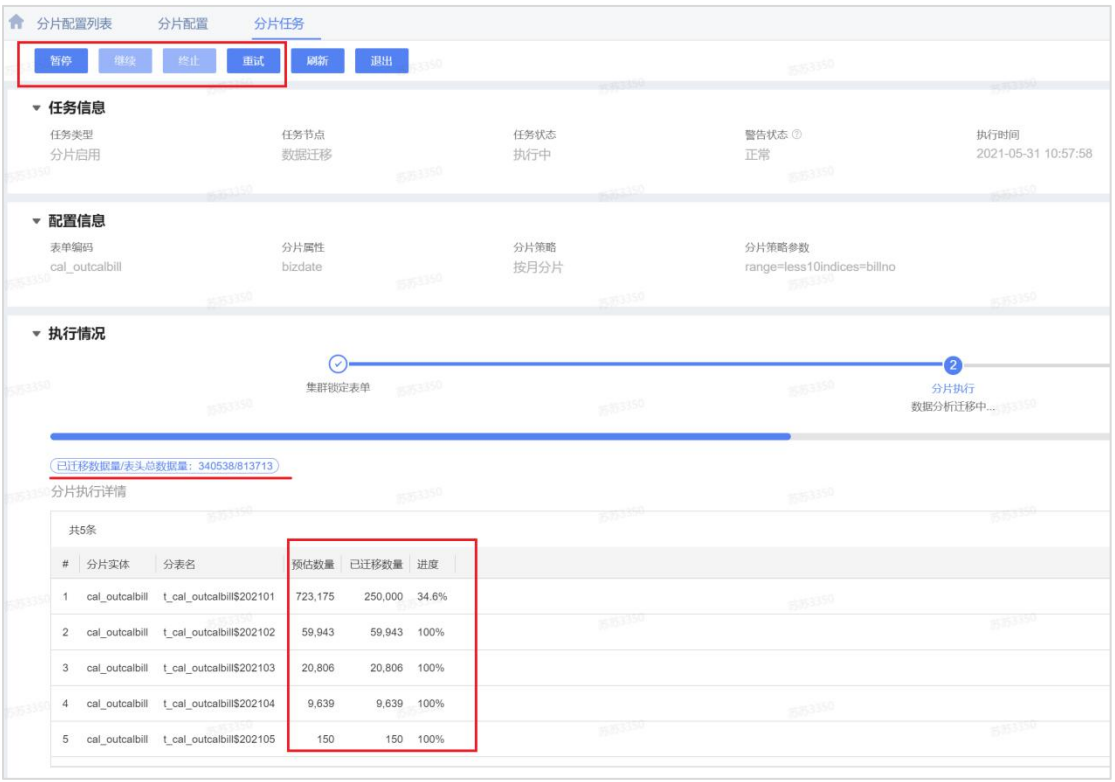
   2) **暂停**：暂停...-->继续。

   3) **重试**：若中途出现错误，则会提示并中断，在**分片操作日志**中可查询详细信息，待解决后，点击重试继续执行迁移。

5.数据迁移**可用性保障**：数据迁移在其中一个 SYS 服务节点执行，若这个节点重启或宕机，则会重新选择一个 SYS 服务节点继续执行。

6.对已分表的表单，可取消水平分表：分片配置点击“还原分片”，系统将执行数据迁移任务，把多个分片表合并成原始表。





四、分片属性与策略的选择

1. 分片属性选择

选择的分片属性一般需要符合下特征之一：

① 具有**隔离性**，大部分场景作为必选的过滤条件，如组织、期间。

② 具有**时序性**，数据按时序累积分布，如业务日期、期间。

③ 具有**固定性**，不常更新、必录项。

因数据会随着使用时间累积增长，应尽可能选择含与时序相关的属性。

分片属性通常选择为为 1 个或 2 个(组合)，属性仅限为表头主表的属性，不可选扩展表属性。

2. 分片策略选择

系统提供了五类分片策略，如下表。

选择建议：映射策略和日期策略可覆盖大部分的使用场景，单属性分片且为日期类型时，选用日期策略，其余选择映射策略。

分片策略	使用场景	备注
映射策略	属性类型：任意 单、多属性分片	万能的策略
日期策略	属性类型：日期 单属性分片	按日分片 按月分片 按年分片 日期哈希分片--先格式化日期
ID 时序策略	Long 类型的主键、引用属性	从 ID 值中获取创建时间，做日期分片用。
哈希取模策略	属性类型：任意 单、多属性分片	数据分布在预设的 N 张分表中
自定义策略	以上策略不能满足的情况	用户自定义，配置类路径关联。

3. 映射策略

1) **映射策略**，把某类属性值(多属性则为组合值)的数据，分到一个表中，系统记录分片值与分片索引的映射关系。

当进行查询时，解析分片属性的过滤条件，在映射表中查询到分片表，然后从分片表中查询数据。

多属性分片，若过滤条件只含其中部分属性的条件，映射策略依旧定位到它对应的分片表(通过查询映射表)。

若条件不含分片属性，则将对所有分片表进行查询，这种情况应避免，也就是在程序的代码中访问数据库，需要加分片属性条件。这个优化过程叫**优化适配**，一个表单虽然可以任

意设置分片属性，但是考虑到性能因素，要根据大部分业务场景使用的情况进行选择，参考上面的“分片属性的选择”。

分片策略参数如下：

2) **数据量估算**：表单表头行数的估算，水平分表的内置索引表存储了全量索引列数据，它会根据数据行数进行数据库物理表的分区存储。估算的数据为“在线”的热数据，以后已完结的数据将被归档到单独的数据库中。

3) **快速索引**：选择需要做索引的属性列，水平分表内置了索引表(物理表)，用于辅助快速定位索引值对应的分片表，其中表单主键是内置的，快速索引列建议不超过 3 个。过滤条件并非总是包含分片属性，比如通过单据编码 billno 进行查询，通过单据 id 加载表单，用快速索引定位避免了对全分片表查询。

4) **属性分隔符**：多属性分片，在映射表中属性值之间的分割符号，默认为#，一般不需要修改。如果分片属性可能包含#，则需要修改为其它不不可能包含的，以提高分片表定位的准确性。

5) **日期格式**：当分片属性含日期时，把属性值用这个格式格式化后，作为映射表中的 key。

6) 自定义参数：每个策略都有各自的自定义参数规则，映射策略的自定义参数，用于自定义映射的规则，如

```
p1.valueMapper=kd.bos.xdb.sharding.strategy.map.mapper.HashModMapper
p1.valueMapper.mod=3
```

表示，第一个分片属性的值，映射规则采用 HashMod，模数为 3。意在对这个属性进行“均摊”，到 3 个表中。

结合其余属性，表个数 x3，如：组织+业务日期，p1 表示组织，p2 表示业务日期。不配则不指定，业务日期若按 yyyy-MM 进行格式化，这个分片属性组合表示：所有组织每个月分 3 张表存储。

系统只内置了一个映射 HashModMapper，可自行实现：

接口： kd.bos.xdb.sharding.strategy.map.mapper.ValueMapper

超类：kd.bos.xdb.sharding.strategy.map.mapper.AbstractValueMapper

```
package kd.bos.xdb.sharding.strategy.map.mapper;

/**
 * 自定义值映射规则
 *
 * @author zzf
 *
 */
@FunctionalInterface
public interface ValueMapper {

    /**
     * @param fieldIndex
     *            分片属性的位置索引
     * @param field
     *            分片属性字段名
     * @param value
     *            原始值
     * @return 映射的值
     */
    Object mapValue(int fieldIndex, String field, Object value);
}
```

示例：HashModMapper

```
public class HashModMapper extends AbstractValueMapper {
    private final int mod;

    public HashModMapper(Map<String, String> paramMap) {
        super(paramMap);
        mod = Integer.parseInt(paramMap.getOrDefault("mod", "5"));
        if (mod <= 1) {
            throw new IllegalArgumentException("HashModMapper: mod should b
        }
    }

    @Override
    public Object mapValue(int fieldIndex, String field, Object value) {
        return HashCodeUtil.getHashCodeSingle(value) % mod;
    }
}
```

4. ID 时序策略

表单 ID 值为创建时所产生，故 ID 值里包含了创建时间，可作为日期类型进行分片，用作没有日期属性的表单但又期望分表具有时序性。

用 id 过滤日期范围：

```
IDRange range = ID.getIDRangeOfDay(new Date());

QFilter todayFilter = QFilter.of("id>=? and id<=?", range.getMinId(), range.getMaxId());
```

5. **自定义策略**

扩展基类：kd.bos.xdb.sharding.strategy.BaseCustomStrategy

实现对应方法。

难度较大不建议自己扩展，建议用“映射策略+自定义 ValueMapper”来实现。

五、数据存储

分表的格式：**原表名$后缀**

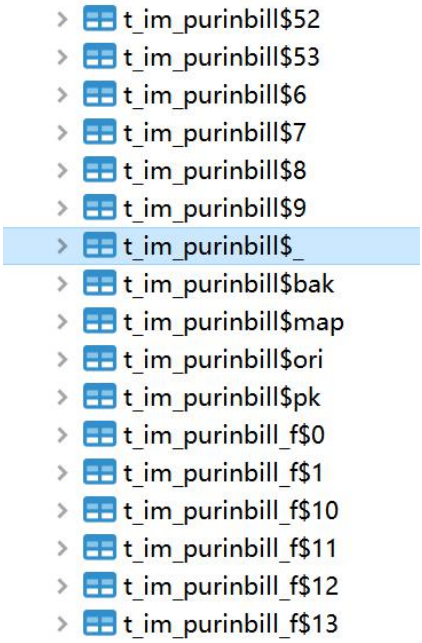
水平分表的表类型如下：

表类型	表名	说明
原型表	T_XX$_	空表，用于记录表结构，在未定位到分片表时，使用此表代替。
索引表	T_XX$pk	快速索引表
数据表	T_XX$0、T_XX$1、T_XX$2021 等	数值后缀
映射表	T_XX$map	映射策略的映射表
备份表	T_XX$bak	数据迁移过程中，对原始表进行备份。
中间表	T_XX$m 等	数据迁移过程中用到的中间表，迁移完毕将自动删除。
原始表	T_XX$ori	对原始表结构的备份。

注意：请勿直接对分表进行 DDL、增删改操作，所有的 SQL(KSQL 类型)，都应使用原表。

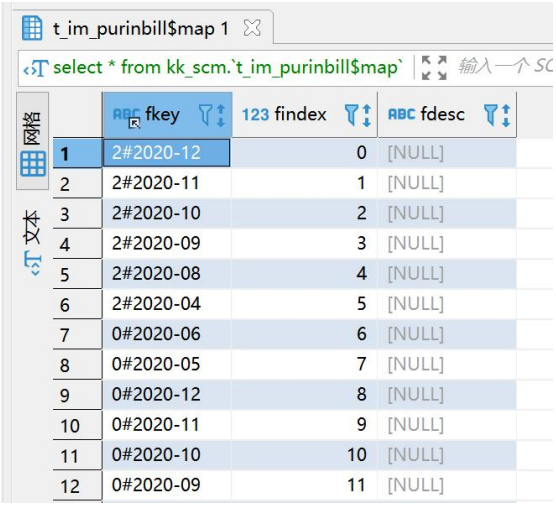
若数据表表名过长(超出数据库表名的长度限制)，系统会映射为库内唯一的短表名，名称映射保存在 SYS 库的 t_cbs_shard_name_map 中。

表名示例：



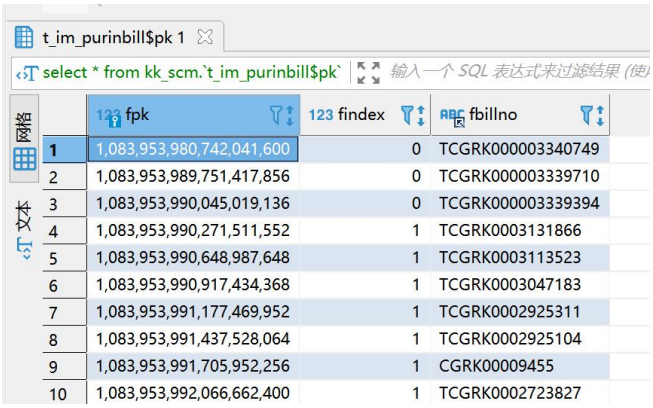
一个表单，含多个表，如表头、分录、扩展表，一张单据的数据都存在同一个分片索引表中。如表头存在 52 的分片表，则其分录、子分录、扩展表等数据也存在对应的 52 的分片表中。

映射策略表数据形如:



	fkey	findex	fdesc
1	2#2020-12	0	[NULL]
2	2#2020-11	1	[NULL]
3	2#2020-10	2	[NULL]
4	2#2020-09	3	[NULL]
5	2#2020-08	4	[NULL]
6	2#2020-04	5	[NULL]
7	0#2020-06	6	[NULL]
8	0#2020-05	7	[NULL]
9	0#2020-12	8	[NULL]
10	0#2020-11	9	[NULL]
11	0#2020-10	10	[NULL]
12	0#2020-09	11	[NULL]

索引表数据形如(含快速索引 billno)：



	fpk	findex	fbillno
1	1,083,953,980,742,041,600	0	TCGRK000003340749
2	1,083,953,989,751,417,856	0	TCGRK000003339710
3	1,083,953,990,045,019,136	0	TCGRK000003339394
4	1,083,953,990,271,511,552	1	TCGRK0003131866
5	1,083,953,990,648,987,648	1	TCGRK0003113523
6	1,083,953,990,917,434,368	1	TCGRK0003047183
7	1,083,953,991,177,469,952	1	TCGRK0002925311
8	1,083,953,991,437,528,064	1	TCGRK0002925104
9	1,083,953,991,705,952,256	1	CGRK00009455
10	1,083,953,992,066,662,400	1	TCGRK0002723827

六、特性说明

1. 水平分表功能特性：

① 适配苍穹领域模型：多级关联(4 级：表头-分录-子分录-扩展表)，理论上不限级别。

② 分片策略：支持多属性分片、多种分片策略，简捷地自定义分片策略。

③ 内置 PK 索引，支持全局快速索引。

④ 可更新分片属性。

⑤ 支持分片 SQL Hint：充分优化适配。

⑥ 支持最大限度分片条件定位：in or between like 等。

⑦ 支持大部分 SQL 特征：union、order by、distinct、top/limit。

⑧ 支持多种数据库：目前支持 MySQL、Oracle，以后将支持所有苍穹所支持的数据库。

⑨ 自适应并发查询。

⑩ 分布式运行：缓存、锁、表缓存版本等。

⑪ 数据迁移与管理。

⑫ 指标采集与监控。

2. 使用限制

① 不支持方言 SQL (/*dialect*/前缀)。

② 不支持 SQL：avg、having、exists。

③ 不支持此类 update 操作： update A set ... select from B...，其中 B 为分片表。

七、实施建议

1. 在测试/沙箱环境预先做分表操作，分表后进行全功能测试。

2. 根据**分片指标采集**的数据，进行适配优化。

3. 检查是否存在分表所限制使用的 SQL 进行数据访问。

八、适配优化

1. 发现问题

   1) 不带分片属性条件过滤的 sql。

   2) 一次查询过多分片表的 sql。

   3) 执行慢的 sql。

2. 定位方法

XDB 内置了性能监控功能，配置好告警阀值，开启性能监控后，将自动输出警告信息。警告信息含调用堆栈，可快速定位具体的代码调用方法。

1) 开启参数

非开发环境则在 mc 中配置，通常只需关注三个参数：

① xdb.xpm.enable=true

② xdb.xpm.export.type=bill

③ xdb.xpm.alarm.updateShardingField=true

参数明细如下：

```
【相关开关配置】
//开启 sql 日志
System.setProperty("db.sql.out", "true");
System.setProperty("db.sql.out.withParameter", "true");

//开启 xdb 性能监控
System.setProperty("xdb.xpm.enable", "true");

//监控信息输出类型：
//std 为输出到控制台，log 为输出到日志，bill 为输出到“分片指标采集”表单。
System.setProperty("xdb.xpm.export.type", "bill");

【告警阀值配置】

//告警 1：当 sql 缺少分片条件时，输出告警信息。
//默认开启，无需配置。
```

```
//告警 2：当 sql 语句执行超过 5000 毫秒时，输出告警信息。
System.setProperty("xdb.xpm.alarm.totalSpent", "5000");

//告警 3：当 sql 语句定位分片表数，超过 8 个时，输出告警信息。
System.setProperty("xdb.xpm.alarm.shardingTableCount", "8");

//告警 4：sql 语句含对分片属性的更新，默认开：true。
System.setProperty("xdb.xpm.alarm.updateShardingField", "true");

//告警 5：sql 语句执行错误，默认关：false。
System.setProperty("xdb.xpm.alarm. executeError", "false");
```

2)告警信息

开发环境建议在控制台输出(xdb.xpm.export.type=std)，非开发环境，在 monitor 中查询关键字：alarm。告警信息形如：

告警：缺少分片条件

```
XPM metrics alarm: QueryMetrics{name=5a3cb00a14daa651, sharding=true, hitParseCache=0, missParseCache=1, tc
XPM WithoutFullShardingCondition bos_pea_xdb_show_2@One2OneMapStrategy: t_pea_xdb_show_2[fbringdate, flevel
SQL:
SELECT 1
FROM t_pea_xdb_show_2
at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsBase.logCallStack(QueryMetricsBase.java:154)
        at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsImpl.logCallStack(QueryMetricsImpl.java:9)
        at kd.bos.xdb.sharding.sql.condition.ConditionShardingSQL.sharding(ConditionShardingSQL.java:164)
        at kd.bos.xdb.engine.ShardingEngineImpl.shardingTable(ShardingEngineImpl.java:516)
        at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:383)
        at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:257)
        at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:91)
        at kd.bos.xdb.XDBExecutor.query(XDBExecutor.java:109)
        at kd.bos.xdb.ext.ExtXDB.query(ExtXDB.java:36)
        at kd.bos.db.XDBImpl.query(XDBImpl.java:117)
        at kd.bos.db.AbstractDBImpl.query(AbstractDBImpl.java:161)
        at kd.bos.db.DB.query(DB.java:78)
        at kd.bos.db.DB.query(DB.java:69)
        at kd.bos.form.plugin.tools.SQLResultGrid.getData(SQLResultGrid.java:131)
        at kd.bos.form.plugin.tools.SQLResultGrid.handleBusinessLogic(SQLResultGrid.java:115)
        at kd.bos.form.plugin.tools.SQLResultGrid.afterBindData(SQLResultGrid.java:57)
        at kd.bos.form.plugin.FormViewPluginProxy.fireAfterBindData(FormViewPluginProxy.java:158)
```

SQL

调用堆栈

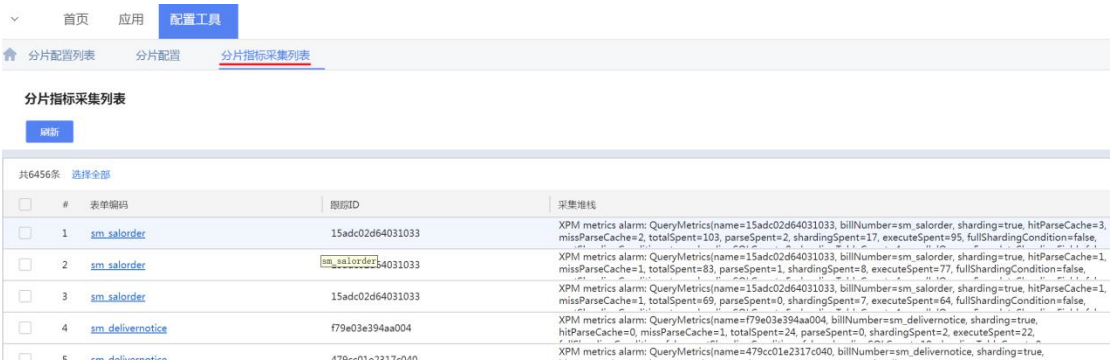
查询分片数过多 (>设定值3)

```
------------------------
tooManyShardingTables(22>3): [T_PEA_XDB_SHOW_2$0, T_PEA_XDB_SHOW_2$1, T_PEA_XDB_SHOW_2$10, T_PEA_XDB
at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsBase.logCallStack(QueryMetricsBase.java:154)
        at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsImpl.logCallStack(QueryMetricsImpl.java:9)
        at kd.bos.xdb.sharding.sql.condition.ConditionShardingSQL.sharding(ConditionShardingSQL.java
        at kd.bos.xdb.engine.ShardingEngineImpl.shardingTable(ShardingEngineImpl.java:516)
        at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:383)
        at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:257)
        at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:91)
        at kd.bos.xdb.XDBExecutor.query(XDBExecutor.java:109)
        at kd.bos.xdb.ext.ExtXDB.query(ExtXDB.java:36)
```

告警：更新分片属性

```
XPM metrics alarm: QueryMetrics{name=5fdfc00651794010, sharding=tru
Update sharding field: flevel
SQL:
UPDATE t_pea_xdb_show_2
SET fmodifytime = ?, flevel = ?
WHERE FId = ?
at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsBase.logCallStack(Qu
        at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsImpl.logCall
        at kd.bos.xdb.sharding.sql.dml.UpdateShardingSQL.limitUpdat
```

若输出到表单(xdb.xpm.export.type=bill)，则在界面“分片指标采集列表”也可以查看到：



采集堆栈

XPM metrics alarm: QueryMetrics{name=223030142092f045, billNumber=scp_saloutstock, sharding=true, hitParseCache
shardingTableCount=0, parallelQuery=4, updateShardingField=false, executeError=false}

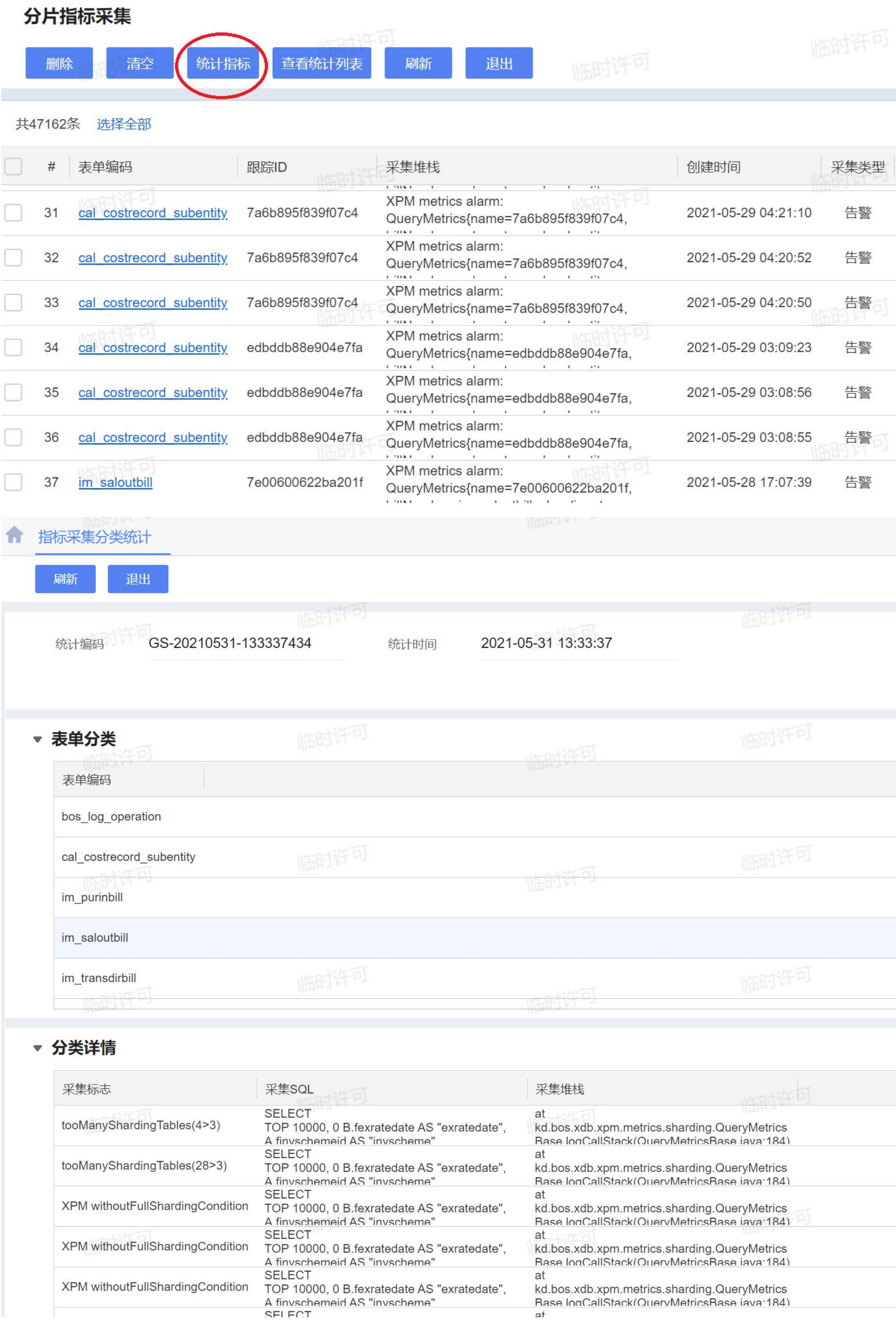
XPM withoutFullShardingCondition scp_saloutstock@DateYearStrategy: t_pur_saloutstock[fbilldate]

SQL:

```
SELECT A.FId AS "id", A.fbilldate AS "billdate"
FROM T_PUR_SALOUTSTOCK A
      LEFT JOIN T_PUR_SALOUTSTOCK_A B ON B.FId = A.FId
      LEFT JOIN T_PUR_SALOUTSTOCKENTRY C ON C.FId = A.FId
      LEFT JOIN T_PUR_SALOUTSTOCKENTRY_A D ON D.FEntryId = C.FEntryId
WHERE A.FId != ?
      AND A.FDeliDate <= ?
      AND B.fisinitial = ?
      AND A.FOrgID IN (?)
      AND A.FBillStatus = ?
      AND A.flogstatus != ?
      AND C.FQty > D.FSumReceiptQty + C.frejectqty
      AND C.FQty > D.fsuminstockqty + C.frejectqty
      AND C.FEntryStatus = ?
ORDER BY A.fbilldate DESC
```

```
t kd.bos.xdb.xpm.metrics.sharding.QueryMetricsBase.logCallStack(QueryMetricsBase.java:184)
      at kd.bos.xdb.xpm.metrics.sharding.QueryMetricsImpl.logCallStack(QueryMetricsImpl.java:9)
      at kd.bos.xdb.sharding.sql.condition.ConditionShardingSQL.sharding(ConditionShardingSQL.java:163)
      at kd.bos.xdb.engine.ShardingEngineImpl.shardingTable(ShardingEngineImpl.java:544)
      at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:398)
      at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:267)
      at kd.bos.xdb.engine.ShardingEngineImpl.slice(ShardingEngineImpl.java:95)
      at kd.bos.xdb.XDBExecutor.query(XDBExecutor.java:112)
```

采集的日志，存在许多同类问题，选择“统计指标”可做取**去重统计**：


分片指标采集
删除
清空
统计指标
查看统计列表
刷新
退出
共47162条
选择全部
表单编码 跟踪ID 采集堆栈 创建时间 采集类型
31 cal_costrecord_subentity 7a6b895f839f07c4 XPM metrics alarm: QueryMetrics{name=7a6b895f839f07c4, 2021-05-29 04:21:10 告警
32 cal_costrecord_subentity 7a6b895f839f07c4 XPM metrics alarm: QueryMetrics{name=7a6b895f839f07c4, 2021-05-29 04:20:52 告警
33 cal_costrecord_subentity 7a6b895f839f07c4 XPM metrics alarm: QueryMetrics{name=7a6b895f839f07c4, 2021-05-29 04:20:50 告警
34 cal_costrecord_subentity edbddb88e904e7fa XPM metrics alarm: QueryMetrics{name=edbddb88e904e7fa, 2021-05-29 03:09:23 告警
35 cal_costrecord_subentity edbddb88e904e7fa XPM metrics alarm: QueryMetrics{name=edbddb88e904e7fa, 2021-05-29 03:08:56 告警
36 cal_costrecord_subentity edbddb88e904e7fa XPM metrics alarm: QueryMetrics{name=edbddb88e904e7fa, 2021-05-29 03:08:55 告警
37 im_saloutbill 7e00600622ba201f XPM metrics alarm: QueryMetrics{name=7e00600622ba201f, 2021-05-28 17:07:39 告警
指标采集分类统计
刷新
退出
统计编码 GS-20210531-133337434
统计时间 2021-05-31 13:33:37
表单分类
表单编码
bos_log_operation
cal_costrecord_subentity
im_purinbill
im_saloutbill
im_transdirbill
分类详情
采集标志 采集SQL 采集堆栈
tooManyShardingTables(4>3)
tooManyShardingTables(28>3)
XPM withoutFullShardingCondition
SELECT TOP 10000, 0 B.fexratedate AS "exratedate",
at kd.bos.xdb.xpm.metrics.sharding.QueryMetrics

--End--