SOFAStack

资金安全监控
技术白皮书

产品版本：AntStack Plus 1.13.1

文档版本：20230707

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.资金安全概述 05
2.应用场景 06
3.产品架构 07
4.风险点分类及举例 09
4.1. 风险点分类 09
4.2. 风险点举例 09
5.资金安全核对脚本示例 18
5.1. 一致性核对规则 18
5.2. 业务正确性核对规则 21
5.3. 时效性核对规则 23
5.4. 幂等检查规则 24
5.5. 额度检查规则 25
6.离线表的使用说明 27
7.分区的使用说明 28
8.规则执行性能问题说明 30
9.日常预警说明 31
10.资源优化 32
11.基础术语 33

1.资金安全概述

什么是资金安全

资金安全即一切涉及到企业或企业客户的资金的行为均属于资金安全范畴，需要从多方面去保障个体或企业客户的资金的安全性。

什么是资损

任何由于产品设计缺陷、系统故障、系统缺陷、人为操作、安全漏洞等导致公司或公司客户直接或间接的蒙受资金损失的事件，或未获得相应的优惠使用等，且最终均需由平台赔偿或平台多出流量、权益、钱等，都属于资损事件。

通俗的讲，资损就是非预期的资金表现。

产品设计缺陷	客户操作充值 5 元，收到我司验证支付的短信提示也是 5 元，但是交易实际支付了 500 元。原因是充值服务商和我司约定的接口中金额字段的单位不一致，商户单位是元，我司单位是分，导致交易金额变成实际订单金额的 100 倍。
系统故障	商户群红包业务一笔转账发生并发请求，重复转账 2 次未被拦截， 并发控制策略是 ADDP + 数据库表唯一性约束，2 层防御手段都被击穿。 原因：业务系统发布过程中 & DB 切 failover & addp 推 DRM，推错值。
人为操作	某营销活动运营提交了错误的活动配置，把实物奖品配置成了红包，导致用户本该在 X 权益部分看到一个实物奖品即只有一个链接，不需要发红包的，但实际结果是用户领了红包。
安全漏洞	客户反映 ** 账户被盗，账户内有申请数字证书和绑定手机开通动态口令，但款项还是被异常支出，期间证书与手机的短信服务均没有起到任何作用，也没有短信下发给客户。

理论资损的定义

- 结合资损故障现象，预估可能带来的资损金额。
- 被动赔付的场景，可按照业务约定的赔付比例或结合客满实际收到的反馈量预估赔付金额。

解决了哪些问题

- 发现潜在资损风险，避免资金损失。
- 及时发现资损故障，降低资金损失以及社会舆论。

2.应用场景

业务护航

您可以阶段性地或定期梳理涉及资金链路的核心业务。可通过配置核对规则，对各类数据作相互核对或对数据内容作逻辑检查，系统将按规则监控业务的资损风险。

变更风险检查

业务变更发布前，您可以添加变更业务表与关联表的核对规则，或者变更业务表数据的检查规则，确保变更上线后不存在资损风险监控盲点。同理，技改类变更发布前，您也可以针对变更点新增核对规则，或修改原有核对规则。

历史数据扫雷

批量检查存量业务的历史数据是否已存在差异，可以及时分析差异原因、修复漏洞、追回资损。

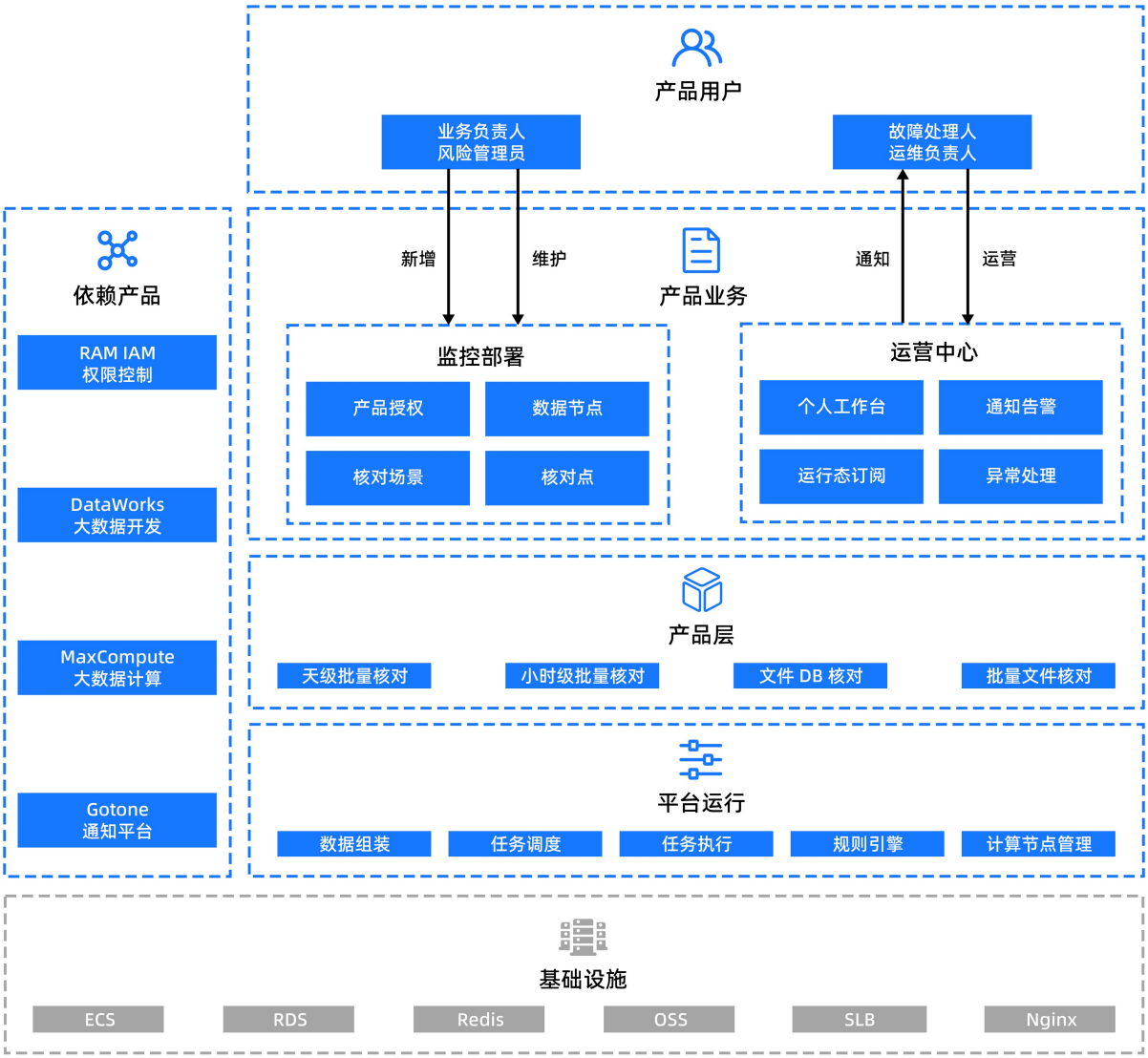
数据质量监控

数据缺漏时，也可能间接引发资损。您可以配置核对规则以检查数据完整性，对数据质量作监控，及时发现故障。

3.产品架构

系统架构

下图展示了资金安全监控的整体系统架构。

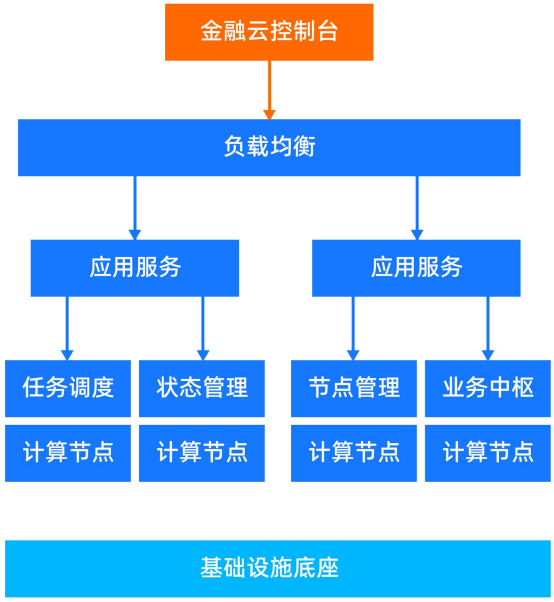


各模块说明如下：

- **产品用户**：该产品可能具备的操作角色。
- **依赖产品**：在使用该产品可能需要使用和依赖的产品。
- **产品业务**：可根据实际情况对使用方式进行调整。
- **产品层**：提供核对能力。
- **平台运行**：平台运行所依赖的一些内部组件。
- **基础设施**：依赖的基础部署设施，这通常包含在云基础底座中。

部署架构

下图展示了资金安全监控的整体部署架构。

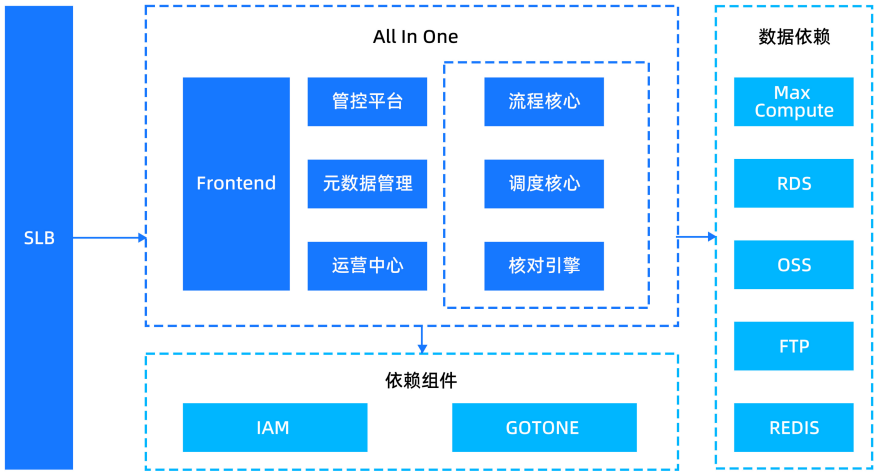


各模块说明如下：

- **金融云控制台**：产品的操作界面主要通过金融云控制台展现。
- **负载均衡**：使用通用的负载均衡组件进行操作界面流量调拨和管控，以满足容灾及运维场景。
- **应用服务**：为上层操作提供相应服务。
- **各关键业务模块**：主要的部署计算角色，这都集成在同一应用内。
- **基础设施底座**：依赖的 ECS、RDS 等基础设施，这通常包含在云基础底座中。

应用架构

下图展示了资金安全监控的整体应用架构。主要包括前端通过 SLB 作负载，服务端作为一个 All-in-one 的应用作部署，应用的主要组件以及应用的软硬件依赖。



4.风险点分类及举例

4.1. 风险点分类

序号	风险大类
1	一致性
2	业务型
3	时效性
4	数据型
5	配置操作
6	算法、计算决策

4.2. 风险点举例

一致性

上下游一致

分类描述：

在上下游传递时，部分资金相关字段（如金额、状态、币种、汇率），需要保持上下游一致，否则会导致资损。

分类发生场景：

上游系统或者域之间与下游系统交换数据期间，约定不一致所致。

分类模型：

上游 A0，下游 B0，比率 C，上游关联 A1，下游关联 B1。

常见分类核对公式：

- A0==B0
- A0==B0 * C

分类 tag：

一致性、上下游、信息一致。

内外部信息一致

分类描述：

当行方与外部交换数据、文件时，需要保证数据明细的正确性、完整性。如果出现不一致、缺数据、多数据有可能导致资损。

分类发生场景：

当业务处于未知状态时，数据回查很容易出问题，另外与外部机构进行清算对账时，也可能出现不一致。

分类模型：

核对字段（内部数据 A，外部数据 B，size(A),A[i].a0，size(B)，B[i].b0，比率 c），关联字段（A[i].关联 a1，B[i].关联 b1）。

常见分类核对公式：

- A[i].a0==B[i].b0
- A[i].a0==B[i].b0*c

分类 tag：

一致性、外部机构、信息一致。

上下游幂等

分类描述：

某个服务处理过程中，对一笔业务请求只处理一次，并在重复请求的情况下，返回同样的处理结果或者幂等错误码。如果服务端幂等没有控住，就会产生资损。

分类发生场景：

内部系统之间或机构账户之间与外部系统交换数据期间，幂等号约定不一致所致。

分类模型：

核对字段（上游单据流水 A0，下游单据流水 B0，A0 下游单据号 BID，B0 上游单据号 AID），关联字段。

常见分类核对公式：

- size (A0,BID)==1
- size (B0,AID)==1

分类 tag：

幂等、上下游、重复处理。

内外部幂等

分类描述：

在蚂蚁系统与外部合作机构的系统进行数据传递时，由于重发与幂等机制，导致一笔业务的内外单据数量、状态不一致。

分类发生场景：

内外部系统由于重发导致一笔业务被当作多笔业务处理，比如水电缴费、提现业务银行多笔打款。

分类模型：

内部单据 A，外部单据 B，幂等单号（orderNo），关联字段（orderNo）。

常见分类核对公式：

- A.status==B.status

- size (A.orderNo)==size (B.orderNo)

分类 tag：

一致性、幂等。

在离线一致

分类描述：

离线数据涉及到后续离线清洗使用的字段需要保证和在线的字段数据一致，经常在数据离线时，因为数据离线通道所致。

分类发生场景：

数据离线到 ODPS 等，特别是离线后的数据用于结算、账单、报表，或者会清洗回流给如营销的数据。

分类模型：

在线.主键，在线.A0，在线字段，离线.主键，离线.B0，离线字段列表。

常见分类核对公式：

- 在线.A0==离线.B0
- size (在线.主键) ==size (离线.主键)
- in_list (在线字段，离线字段列表)

分类 tag：

一致性、离线、信息一致。

借贷平衡

分类描述：

资金流业务会产生借贷双方，如果出现借贷双方账户出现金额不平、状态不匹配，很可能会发生资损。

分类发生场景：

借方或贷方异常时，如果没有发起充退等补偿操作，可能会造成借贷不平。

分类模型：

核对字段（借方.A0，贷方.B0，比率 C），关联字段（借方.关联 A1，贷方.关联 B1）。

常见分类核对公式：

- A0==B0
- A0==B0 * C

分类 tag：

资金流、平衡。

业务型

合约正确性

分类描述：

在非法的合约下产生了业务，或者在合约的限定范围之外发生了业务，比如超过了合约的预期时间等。

分类发生场景：

合约的时间是一个比较容易出现的地方，特别是在合约修改频繁时。

分类模型：

contract.属性，biz.属性。

常见分类核对公式：

biz.属性 in contract.属性范围内。

分类 tag：

合约正确性、静态数据查询。

该分类常见发现工具：

免疫、TM。

参与实体准入

分类描述：

业务场景对参与实体（用户、商户、机构、产品、环境等）有一定的准入要求，当未达到要求的参与方发起业务时，可能会导致资损。

分类发生场景：

营销活动对用户、商户准入有要求的业务，比如用户年龄、等级等。

分类模型：

核对字段（用户标识 A，黑名单 BLACKLIST，白名单 WHITELIST，取值范围[min,max]）。

常见分类核对公式：

- A in WHITELIST
- A not in BLACKLIST
- A in [min, max]

分类 tag：

准入、数据质量。

该分类常见发现工具：

免疫、TM、T1TH。

资金流正确性

分类描述：

在特定的业务下，某个交易的资金应该从某个特定的账户流出和特定的账户流入。

分类发生场景：

所有的资金流动的场景都会发生此问题。

分类模型：

流入单据 A，流出单据 B。

常见分类核对公式：

A.accountNo 业务配置的帐号 || B.accountNo 业务配置的帐号。

分类 tag：

资金流、对账。

该分类常见发现工具：

免疫。

发生额正确性

分类描述：

业务凭证和帐务流水的核对，常见于业务发生时需要和实际的帐务流水做一致性的核对。

分类发生场景：

业务在发生的时候，所有资金信息都必须表现在帐务上。

分类模型：

凭证主键，凭证金额，凭证状态，凭证币种，凭证汇率，帐务表，凭证主键，帐务金额，帐务状态，帐务币种，帐务汇率。

常见分类核对公式：

- 凭证金额==帐务金额
- 凭证状态==帐务状态
- 凭证币种==帐务币种
- 凭证金额 * 凭证汇率==帐务金额
- count（凭证主键）==count（帐务表.凭证主键）

分类 tag：

发生额、帐务流水、业务流水。

该分类常见发现工具：

免疫+(TM)、T1TH、AMG。

余额正确性

分类描述：

用户（包括买家、商家、支付宝、机构等）在执行某个业务后的账户余额必须符合资金流流动后的结果且余额不能小于 0，另外也经常做周期性的核对，比如期初和期末的差额需要等于所有交易流水的总和。

分类发生场景：

业务执行过程中账户、金额等出错，或账务异步处理等导致。

分类模型：

user.balance，biz.amount。

常见分类核对公式：

- targetBalance==originBalance-biz.amount
- targetBalance>=0

分类 tag：

资金流动、余额正确性。

该分类常见发现工具：

业务核对资金流模块、日切。

金额拆分控制

分类描述：

在各种交易支付中，往往涉及一对多或者多对多的不同账户的金额计算后，按支付明细支付，其明细与支付总单据不一致。

分类发生场景：

使用优惠券、红包、银行卡一起支付。分润时也需要考虑这个情况，经常需要考虑到业务的拆分逻辑。

分类模型：

汇总单据 A（账号、金额、币种、状态），List<明细单据 B>（账号、账号类型、金额、币种、状态、orderNo）。

常见分类核对公式：

- A.amount==sum(B.amount)
- 支付工具金额拆分逻辑

分类 tag：

业务规则、计算型。

该分类常见发现工具：

T0、TM、th、业务对账系统。

逆向流程控制

分类描述：

在做逆向业务时，需要考虑正向业务的状态和金额控制。

分类发生场景：

退款、拒付、退票、赎回、退税、退费等场景，需要考虑逆向的流程控制，特别是有多种逆向时，比如部分退款后有部分是拒付的。

分类模型：

正向.状态，正向.金额，正向.工具 1 金额，逆向 1.金额，逆向 2.金额，逆向 3.金额，可能更多。

常见分类核对公式：

- in_list（正向.状态,状态列表）
- 正向.金额 >=sum（逆向 1.金额，逆向 2.金额，逆向 3.金额，可能更多）
- 正向.工具 1 金额 >=sum（逆向 1.工具 1 金额，逆向 2.工具 1 金额，逆向 3.工具 1 金额，可能更多）

分类 tag：

逆向流程、金额控制、状态控制。

该分类常见发现工具：

免疫。

额度控制

分类描述：

因受合规监管需要或业务特性，导致业务产品在用户维度单笔、单日、每月会有金额的上、下限，如出现超过上限或小于下限则会出现资金风险。

分类发生场景：

根据额度计算收费，如余额宝提现超 1 万时收取费用、业务单笔额度控制等。

分类模型：

额度（业务），产品帐（业务，用户）。

常见分类核对公式：

- sum（产品帐（业务，用户））≤ 额度（业务）
- biz.amount≤ 单笔额度（业务）

分类 tag：

额度超限。

该分类常见发现工具：

T1TH、免疫+（TM）。

收费计算正确性

分类描述：

当产品涉及费用、税、利息时，需要根据费率、税率、利息等配置来计算收费金额，并根据业务规则将费用分摊到多方（比如蚂蚁、机构、商户、用户等）。如果收费配置匹配失败，都可能造成资损。

分类发生场景：

业务使用（配置）了错误的收费模板、费用分摊时未考虑小数尾数；利息计算使用日期区间有误。

分类模型：

核对字段（交易金额 A，总费用配置 B，总费用金额 C），关联字段（借方.关联 A1，贷方.关联 B1）。

常见分类核对公式：

费用计算：A*B==C。

分类 tag：

费、税、息、佣金。

该分类常见发现工具：

免疫、TM、TH、T1。

汇总型计算正确性

分类描述：

在业务系统中做资金汇总或资金池控制时，总体资金/量跟明细汇总出来的金额/量不一致的情况下，会产生资损。一般是业务逻辑并发控制有问题。

分类发生场景：

营销预算、结算打批、清算文件解析等总量控制，拆分汇总场景，账单。

分类模型：

核对字段（明细流水 A，汇总流水 B，A.处理状态 SA，B.处理状态 SB，A.资金字段 MoneyA，B.资金字段 MoneyB，B.数量汇总字段 SizeB），关联字段（A.汇总流水号 BID，B.汇总流水号 BID）

常见分类核对公式：

- sum(A.MoneyA) = MoneyB

- size(A) = B.SizeB
- A.SA==B.SB

分类 tag：

汇总、打批、总量。

该分类常见发现工具：

T1、总督。

时效性

时效性符合预期

分类描述：

因为破坏了 SLA 承诺，资金没有正确走到该有的实体，导致的资损节间问题。或者因为类似国际汇率、一些行情的实时波动的数据，在预期的时间内没有完成交易导致的资金安全问题。

分类发生场景：

汇率和行情波动、国际的拒付业务。需要在特定时间内和渠道做沟通，否则就算平台的问题，导致平台资损。

分类模型：

biz.时效，业务约定.可接受时效。

常见分类核对公式：

biz.时效 ≤ 业务约定.可接受时效。

分类 tag：

时效性。

该分类常见发现工具：

天眼。

数据型

外部输入的正确性

分类描述：

在有用户输入时或外部 API 请求时，需要校验外部输入的正确性。

分类发生场景：

用户页面输入的参数，包含用户输入和管理员输入，外部系统调用内部系统的调用参数。

分类模型：

输入参数 1，输入参数 2。

常见分类核对公式：

- 特定值 <= 输入参数 1<= 特定值
- in_list（输入参数 2，值列表）
- mapping [输入参数 1]== 输入参数 2

分类 tag：

正确性、单源。

该分类常见发现工具：

免疫，更多的信息因为单源无法检查。

清洗数据的正确性

分类描述：

当对生产数据进行清洗、加工时，如果发生数据延迟，或者取数和计算逻辑错误，都可能会导致资损。

分类发生场景：

离线数据同步出现延时或异常时，可能造成结果有误。

分类模型：

/

常见分类核对公式：

无

分类 tag：

数据质量、离线。

该分类常见发现工具：

TH、T1、总督。

配置操作

配置输入正确性

分类描述：

在佣金比率、价格、发奖有效期规则、机构账号等配置中，由于输入配置值跟实际业务预期不一致所致。

分类发生场景：

营销发奖把优惠券面额 5% 折现应该配置 0.05，结果配置成 5。

分类模型：

核对字段（配置要素：账号、金额、时间、状态、比率等）。

常见分类核对公式：

- A.account IN(?)
- B.priceAmount bettwen(0,10000)
- B.rate bettwen(5,10)
- B.endtime <today+1Yeah，endtime >beginTime，begingTime>now()

分类 tag：

配置差错、操作型。

该分类常见发现工具：

免疫、T1TH。

5.资金安全核对脚本示例

5.1. 一致性核对规则

一致性核对规则用于验证上下游系统间、内外部系统间或在线与离线系统间的数据一致性，尤其是资金相关的字段（金额、币种、单位等）在左表和右表中一致。

规则模板

标准一致性规则 SQL 模板

```
SELECT t_left.*
     ,t_right.*
FROM (***
      )t_left
FULL OUTER JOIN (***
               )t_right
    ON ***
WHERE t_left.pk IS NULL
    OR t_right.pk IS NULL;
```

> 说明
>
> 一般用全连接 FULL OUTER JOIN，根据业务场景也会用左连接 LEFT JOIN 或者右连接 RIGHT JOIN。

驱动表核对规则 SQL 模板

```
SELECT t_left.*
     ,t_right.*
 FROM (***
      )t_left
LEFT OUTER JOIN (***
               )t_right
   ON ***
WHERE t_left.pk IS NULL
    OR t_right.pk IS NULL
```

> 说明
>
> 模板仅供参考。

标准两表一致性核对符合预期举例

业务场景：在支付业务中，商户先在收单系统（acquire 表）创建支付单，用户随后支付订单（pay 表）。正常情况下用户支付成功后，acquire 表和 pay 表记录都是成功状态且一一对应，并且支付金额、币种和时间一致。

数据表：

- acquire 增量表 `ods_acquire_delta_mm`：

order_id	payer_id	currency	amount	order_status	fund_type	business	complete_time	created_time	jstime	dt	hh	mm
20191019111119997	A158653	CNY	680	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
20191019111119998	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

- pay 增量表 `ods_pay_delta_mm`：

pay_order_id	request_id	role_id	pay_currency	pay_amount	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
20191019111119997	20191019134000012	A158653	CNY	680	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
20191019111119998	20191019134000013	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

SQL 样例：

```
SELECT t_left.*
     ,t_right.*
 FROM (SELECT t1.order_id
             ,t1.payer_id
             ,t1.currency
             ,t1.amount
            ,t1.jstime as left_jstime
         FROM schema.ods_acquire_delta_mm t1
        WHERE t1.order_status = 'SUCCESS'
          AND t1.fund_type = 'PATMENT'
          AND t1.business ='ACQUIRE'
          AND t1.dt = '{yyyyMMdd-5m}'
      AND t1.hh='{hh-5m}'
      AND t1.mm='{mm-5}'
      )t_left
FULL OUTER JOIN (SELECT t2.pay_order_id
                       ,t2.role_id
                       ,t2.pay_currency
                       ,t2.pay_amount
                      ,t2.jstime as right_jstime
                  FROM schema.ods_pay_delta_mm t2
                 WHERE t2.biz_order_type = 'PATMENT'
                   AND t2.order_status = 'SUCCESS'
               AND t2.business ='ACQUIRE'
                    AND t2.dt = '{yyyyMMdd-5m}'
           AND t2.hh='{hh-5m}'
           AND t2.mm='{mm-5}'
                )t_right
   ON t_left.order_id = t_right.pay_order_id
  AND t_left.payer_id = t_right.role_id
  AND t_left.currency = t_right.pay_currency
  AND t_left.amount = t_right.pay_amount
WHERE t_left.order_id IS NULL
    OR t_right.pay_order_id IS NULL;
```

核对结果：SQL 运行正常，没有查询出异常数据，无预警。

特别说明：在该一致性核对中，使用了 `FULL OUTER JOIN` ，为避免监控误报噪音（数据由于业务时间问题，两张表的数据没有落在相同的核对周期内导致预警），提供了自动降噪功能，但需要在 SQL 中，左右表查询出的数据包含该降噪字段 jstime（有且只有该字段可以作为降噪字段），同时在创建规则界面上配置降噪参数。

标准两表一致性核对不符合预期举例

数据表：

- acquire 增量表 `ods_acquire_delta_mm` ：

order_id	payer_id	currency	amount	order_status	fund_type	business	complete_time	created_time	jstime	dt	hh	mm
20191019111119997	A158653	CNY	680	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
20191019111119998	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

- pay 增量表 `ods_pay_delta_mm` ：

pay_order_id	request_id	role_id	pay_currency	pay_amount	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
20191019111119997	20191019134000012	A158653	CNY	180	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
20191019111119998	20191019134000013	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

SQL 样例：同 标准两表一致性核对符合预期 。

核对结果： SQL 运行正常，查询出两张表中金额不一致的异常数据，产生预警。

单向一致性核对举例

业务场景：在支付业务中，商户先在收单系统（acquire 表）创建支付订单，用户随后支付订单（pay 表）。正常情况下用户支付成功后，acquire 表和 pay 表记录都是成功状态且一一对应，并且支付金额、币种和时间一致。 但是用户支付的周期比较长，在增量表中仅包含最近时段的数据，两表数据无法关联起来。因此我们以最后发生数据变更的 pay 增量表作为驱动表，LEFT JOIN 关联 acquire 全量表做一致性核对。

数据表：

- 收单全量表 `ods_acquire_full_mm` ：

order_id	payer_id	currency	amount	order_status	fund_type	business	complete_time	created_time	jstime	dt	hh	mm
20191019111119997	A158653	CNY	680	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
20191019111119998	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

- 支付增量表 `ods_pay_delta_mm` （驱动表）：

pay_order_id	request_id	role_id	pay_currency	pay_amount	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
20191019111199997	20191019134000012	A158653	CNY	680	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
20191019111199998	20191019134000013	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

SQL 样例：

```
SELECT t_left.*
     ,t_right.*
 FROM (SELECT t2.pay_order_id
       ,t2.role_id
       ,t2.pay_currency
       ,t2.pay_amount
       ,t2.jstime as right_jstime
       FROM schema.ods_pay_delta_mm t2
       WHERE t2.biz_order_type = 'PATMENT'
       AND t2.order_status = 'SUCCESS'
       AND t2.business ='ACQUIRE'
       AND t2.dt = '{yyyyMMdd-5m}'
       AND t2.hh='{hh-5m}'
       AND t2.mm='{mm-5}'

      )t_left
LEFT OUTER JOIN (
  SELECT t1.order_id
             ,t1.payer_id
             ,t1.currency
             ,t1.amount
           ,t1.jstime as left_jstime
           FROM schema.ods_acquire_full_mm t1
          WHERE t1.order_status = 'SUCCESS'
            AND t1.fund_type = 'PATMENT'
            AND t1.business ='ACQUIRE'
            AND t1.complete_time >= '{yyyyMMdd-1} 00:00:00'
            AND t1.dt = '{yyyyMMdd-5m}'
            AND t1.hh='{hh-5m}'
            AND t1.mm='{mm-5}'
                )t_right
   ON t_right.order_id = t_left.pay_order_id
  AND t_right.payer_id = t_left.role_id
  AND t_right.currency = t_left.pay_currency
  AND t_right.amount = t_left.pay_amount
WHERE t_right.order_id IS NULL
    OR t_left.pay_order_id IS NULL;
```

5.2. 业务正确性核对规则

一般业务规则检查

根据业务需求给定的条件，设计核对规则。

业务场景：某营销活动（FUNDRISK201910191445001）仅仅针对新用户，老用户不可以参加。

数据表：营销奖品发放流水表 `ods_promo_delta_mm` 。

request_id	role_id	account_ty	activity_id	rule_id	status	award_amo	business	jstime	created_time	dt	hh	mm
2019102013400004	H001236	NEW	FUNDRISK201910191445001	promo001	SUCCESS	10.00	PROMO	2019-10-20 15:27:06	2019-10-20 15:27:06	20191020	15	35
2019102013400005	N190666	OLD	FUNDRISK201910191445002	promo001	SUCCESS	50.00	PROMO	2019-10-20 15:28:06	2019-10-20 15:28:06	20191020	15	35

SQL 样例：

```
SELECT *
 FROM schema.ods_promo_delta_mm promo
WHERE promo.dt = '{yyyyMMdd-5m}'
 AND promo.hh='{hh-5m}'
 AND promo.mm='{mm-5}'
  AND promo.account_type <>'NEW'
  AND promo.activity_id='FUNDRISK201910191445001'
;
```

核对结果：查询到有一条异常数据，用户类型为 OLD 的用户也参与了该营销活动，产生预警。

总分核对

明细表多条记录汇总的总金额等于汇总表对应的金额。

说明：总分核对建议用全量表。

业务场景：同一订单不同支付方式交易流水的总金额等于订单金额。

数据表：

- `ods_pay_full_mm`：

pay_order_id	request_id	role_id	pay_currency	pay_amount	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
2019101911119997	2019101913400012	A158653	CNY	180	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
2019101911119998	2019101913400013	B189125	EUR	7200	SUCCESS	PAYMENT	ACQUIRE	2019-10-19 15:32:12	2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

- `ods_pay_detail_full_mm`（明细表中同一个支付单有多种不同的支付方式）：

pay_order_id	request_id	role_id	pay_currency	pay_amount	pay_methord	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
2019101911119997	2019101913400010	A158653	CNY	150	Card	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
2019101911119997	2019101913400011	A158653	CNY	30	Coupon	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
2019101911119998	2019101913400012	B189125	EUR	2000	Balance	SUCCESS	PAYMENT	ACQUIRE	2019-10-19 15:32:12	2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35
2019101911119998	2019101913400013	B189125	EUR	5000	Card	SUCCESS	PAYMENT	ACQUIRE	2019-10-19 15:32:12	2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35
2019101911119998	2019101913400014	B189125	EUR	200	Coupon	SUCCESS	PAYMENT	ACQUIRE	2019-10-19 15:32:12	2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

SQL 样例：

```
SELECT t_left.*
      ,t_right.*
  FROM (SELECT t1.pay_order_id
              ,t1.role_id
           ,t1.pay_currency
              ,t1.pay_amount
          FROM schema.ods_pay_full_mm t1
          WHERE t1.biz_order_type = 'PATMENT'
            AND t1.order_status = 'SUCCESS'
            AND t1.business ='ACQUIRE'
            AND t1.dt = '{yyyyMMdd-5m}'
            AND t1.hh='{hh-5m}'
            AND t1.mm='{mm-5}'
            AND TO_DATE(t1.complete_time ,'yyyyMMdd') >= '{yyyyMMdd-1}'
       )t_left
FULL OUTER JOIN (SELECT t2.pay_order_id
                      ,t2.role_id
                     ,t2.pay_currency
                      ,SUM(t2.pay_amount) as amount
                  FROM schema.ods_pay_detail_full_mm t2
                 WHERE t2.biz_order_type = 'PATMENT'
                AND t2.order_status = 'SUCCESS'
                AND t2.business ='ACQUIRE'
                   AND t2.dt = '{yyyyMMdd-5m}'
               AND t2.hh='{hh-5m}'
               AND t2.mm='{mm-5}'
                   AND TO_DATE(t2.complete_time ,'yyyyMMdd') >= '{yyyyMMdd-1}'
                GROUP BY t2.pay_order_id
                        ,t2.role_id
                        ,t2.pay_currency
               )t_right
    ON t_left.pay_order_id = t_right.pay_order_id
   AND t_left.role_id = t_right.role_id
   AND t_left.pay_currency = t_right.pay_currency
   AND t_left.pay_amount = t_right.amount
 WHERE t_left.pay_order_id IS NULL
     OR t_right.pay_order_id IS NULL
;
```

核对结果：规则运行正常，明细表汇总的总金额等于汇总表的金额，无预警。

5.3. 时效性核对规则

对于有时效性要求的业务，可以用到业务时间戳实时发现状态流转不符合预期的数据。

> **说明**
>
> 由于该脚本用到全量表，比较耗资源，不推荐频繁执行，仅建议在业务特别需要时使用。

业务场景：支付业务要求所有的支付单据都需要当天处理完成，如果有超过 1 天没有处理的，需要预警，并通过人工推进单据状态。

数据表： `ods_pay_order_full_mm` 。

pay_order_id	request_id	role_id	pay_currency	pay_amount	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
2019101911119997	2019101913400012	A158653	CNY	680	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
2019101911119998	2019101913400013	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35

SQL 样例：

```
SELECT pay.pay_order_id
      ,pay.role_id
      ,pay.pay_currency
      ,pay.pay_amount
  FROM schema.ods_pay_full_mm pay
WHERE pay.order_status = 'PENDING'
 AND pay.order_created_time < '{yyyyMMdd-1} 00:00:00'
  AND pay.dt = '{yyyyMMdd-5m}'
 AND pay.hh='{hh-5m}'
 AND pay.mm='{mm-5}'
;
```

核对结果： 规则运行正常，查询到有一条超过 1 天没有处理的异常数据，单据状态是 PROCESSING，产生预警。

其他说明： 可以使用 `date_add()` 、 `date_sub()` 等函数，根据业务特点增加或减少相应的时间，来判断单据状态是否符合预期。

5.4. 幂等检查规则

数据落库必须符合幂等规则，同一幂等 ID 且状态相同的数据有且只能有一条记录。

> **说明**
>
> 由于该脚本用到全量表，比较耗资源，仅建议在业务特别需要时使用。

规则模板

SQL 模板：

```
SELECT ****
 FROM ***
WHERE ***
GROUP BY table.unique_id
HAVING COUNT (table.unique_id) > 1
```

> **说明**
>
> 模板仅供参考。

幂等检查规则举例

业务场景： 在支付业务中，商户先在收单系统（acquire 表）创建支付订单，用户随后支付订单（pay 表）。所有商户订单都只能最多产生一个成功支付单。

数据表： 支付全量表 `ods_pay_full_mm` 。

pay_order_id	request_id	role_id	pay_currency	pay_amount	order_status	biz_order_type	business	complete_time	created_time	jstime	dt	hh	mm
2019101911119997	2019101913400012	A158653	CNY	180	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-19 15:31:10	2019-10-19 16:31:12	20191020	16	35
2019101911119998	2019101913400013	B189125	EUR	7200	PROCESSING	PAYMENT	ACQUIRE		2019-10-19 15:31:12	2019-10-19 15:31:12	20191020	15	35
2019101911119999	2019101913400014	Z201457	USD	30.65	FAILED	REFUND	ACQUIRE	2019-10-20 15:35:09	2019-10-19 15:31:12	2019-10-19 15:35:09	20191020	15	35
2019102011110000	2019102013400001	A158653	CNY	1000.05	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 16:31:18	2019-10-20 15:32:05	2019-10-20 16:31:12	20191020	16	35
2019102011110001	2019102013400002	B189125	EUR	7000000	PROCESSING	PAYMENT	ACQUIRE		2019-10-20 15:31:52	2019-10-20 15:31:12	20191020	15	35
2019102011110002	2019102013400003	Z201457	USD	100.4587	FAILED	REFUND	ACQUIRE	2019-10-20 15:35:09	2019-10-20 15:31:46	2019-10-20 15:35:09	20191020	15	35
2019102011110003	2019102013400006	A158653	CNY	2000.1	SUCCESS	PAYMENT	ACQUIRE	2019-10-20 15:27:12	2019-10-20 15:31:10	2019-10-20 15:27:12	20191020	15	35
2019102011110004	2019102013400007	B189125	EUR	99999	PROCESSING	PAYMENT	ACQUIRE		2019-10-20 15:31:12	2019-10-20 15:26:12	20191020	15	35

SQL 样例：

```
SELECT pay.pay_order_id, COUNT(pay.pay_order_id)
FROM schema.ods_pay_full_mm pay
WHERE pay.dt = '{yyyyMMdd-5m}'
  AND pay.hh='{hh-5m}'
  AND pay.mm='{mm-5}'
  AND pay.order_status = 'SUCCESS'
  AND pay.order_created_time >= '{yyyyMMdd-1} 00:00:00'
GROUP BY pay.pay_order_id
HAVING COUNT (pay.pay_order_id) > 1
;
```

核对结果：

规则运行正常，没有重复支付的数据，无预警。

5.5. 额度检查规则

所有流出类场景都必须有上下限控制，尤其上限。流入一般也需要有额度控制，具体看业务场景。

业务场景：营销活动对于额度的效验需要非常严格，包括单个用户账号的限制、单个设备的限制、奖品上限的限制、奖品个数的限制等，营销发奖金额上限为 20 元人民币。该场景属于高时效场景，所以需要使用增量表核对。

数据表：

- 营销活动奖品发放流水表 `ods_promo_delta_mm` ：

request_id	role_id	account_ty	activity_id	rule_id	status	award_amo	business	jstime	created_time	dt	hh	mm
2019102013400004	H001236	NEW	FUNDRISK201910191445001	promo001	SUCCESS	10.00	PROMO	2019-10-20 15:27:06	2019-10-20 15:27:06	20191020	15	35
2019102013400005	N190666	NEW	FUNDRISK201910191445002	promo001	SUCCESS	50.00	PROMO	2019-10-20 15:28:06	2019-10-20 15:28:06	20191020	15	35

- 额度限制配置表 `ods_lc_limit_rule_full_mm` ：

rule_id	amount_range	jstime	created_time	dt	hh	mm
promo001	12004599	2018-10-22 15:35:06	2018-10-22 15:35:06	20191020	15	35
topup	12012000078	2018-10-22 15:35:06	2018-10-22 15:35:06	20191020	15	35

SQL 样例：

```
SELECT promo.*, lc.*
 FROM schema.ods_promo_delta_mm promo
 WHERE promo.dt = '{yyyyMMdd-5m}'
  AND promo.hh='{hh-5m}'
  AND promo.mm='{mm-5}'
 AND promo.status = 'SUCCESS'
 AND promo.activity_id='FUNDRISK201910191445001'
 AND promo.award_amount<=  (
   SELECT SUBSTR(lmtRule.amount_range ,5 ,2) maxValue
   FROM schema.ods_lc_limit_rule_full_mm lmtRule
   WHERE lmtRule.rule_id = 'promo001'
      AND lmtRule.dt = '{yyyyMMdd-5m}'
      AND lmtRule.hh='{hh-5m}'
      AND lmtRule.mm='{mm-5}'
 ) lc
;
```

核对结果：

规则运行正常，一条数据发奖金额 50，超过额度表中最大额度 45，产生预警。

6.离线表的使用说明

离线表命名规范

核对规则的 SQL 中会用到以下两类表：

- 分钟表
- 全量表

表名格式如下：

- 分钟表表名为：`ods_${table}_delta_mm`
- 全量表表名为：`ods_${table}_full_mm`

其中 `${table}` 是指生产环境的实际表名。

离线表包含的数据

分钟表是规则运行时调度任务自动组装生成的表。分钟表包含任务执行时最近的前一个 5 分钟的数据作为核对主数据，另外前后各扩展了 5 分钟的数据，也就是说分钟表实际包含了 15 分钟的数据。

示例 1：

当前任务执行时间为 09:23，分钟表中核对主数据为 9:16~9:20 之间的数据，前后多扩展 5 分钟，所以实际该分钟表包含 9:10~9:25 的数据。

示例 2：

5 分钟增量表 `ods_employee_delta_mm` 实际上包含前后 5 分钟共 15 分钟时间范围内的数据。

- `ods_employee_delta_mm(dt=20191018,hh=19,mm=00)` 包含当日从 18 时 55 分 00 秒到当日 19 时 09 分 59 秒，共 15 分钟的数据。
- `ods_employee_delta_mm(dt=20191018,hh=19,mm=05)` 包含当日从 19 时 00 分 00 秒到当日 19 时 14 分 59 秒，共 15 分钟的数据。

全量表会包含历史全部数据，如果业务数据量巨大，建议根据业务需要，仅同步指定时间点到当前时间的数据，以避免大量资源被占用。

全量表每 5 分钟合并一次，生成一个新的全量表。由于全量表较大，所以只保留最近 12 个分区的数据，一般核对都是使用最近一个分区的数据。

7.分区的使用说明

分区使用举例

日期宏	描述
{yyyyMMdd+/-Nm}	基于当前时间加或减 N 分钟，格式是 8 位数字。 示例：`dt='{yyyyMMdd-5m}'`， • 如果当前时间是 20191205 00:01:00，实际 `dt='20191204'` 。 • 如果当前时间是 20191205 00:10:00，实际 `dt='20191205'` 。
{yyyyMMdd+/-N}	基于当前时间加或减 N 分钟，格式是 8 位数字。 示例：`dt='{yyyyMMdd}'`，如果当前时间是 20191205，实际 `dt='20191205'` 。 示例：`dt='{yyyyMMdd-1}'`，如果当前时间是 20191205, 实际 `dt='20191204'` 。 示例：`dt='{yyyyMMdd+1}'`，如果当前时间是 20191205，实际 `dt='20191206'` 。 **说明** 日期宏不仅可以用于分区时间，还可以用于业务时间，数据同步时间等其他时间戳。
{hh+/-Nm}	基于当前时间加或减 N 分钟，格式为 2 位数字。 示例：`hh='{hh-5m}`， • 如果当前时间是 20191205 00:01:00，实际 `hh='23'` 。 • 如果当前时间是 20191205 00:10:00，实际 `hh='00'` 。

日期宏	描述
{mm+/-N}	基于当前时间加或减 N 分钟，格式是 2 位数字，这里 N 取 5 或者 5 的倍数，N 为 0 时，可以不写。 示例：`mm='{mm-5}'`，表示当前时间最近的前一个 5 分钟的数据作为核对主数据。 • 如果当前时间是 20191205 10:01:00，实际 `mm='55'`，主数据为09:55:00-09:59:59。 • 如果当前时间是 20191205 10:08:00，实际 `mm='00'`，主数据为10:00:00-10:04:59。 示例：`mm='{mm}'`， • 如果当前时间是 20191205 10:01:00，实际 `mm='00'`，主数据为10:00:00-10:04:59。 • 如果当前时间是 20191205 10:08:00，实际 `mm='05'`，主数据为 10:05:00-10:09:59。

分区使用方式

当前推荐方式如下：

```
dt='{yyyyMMdd-5m}'
and hh='{hh-5m}'
and mm='{mm-5}'
```

8.规则执行性能问题说明

全量表数据量很大，一般规则推荐使用增量表。如果特殊业务场景必须使用全量表，该规则不建议高频率执行。

9.日常预警说明

业务数据特点

对于异步调用的业务、业务系统处理延迟、业务数据多次变更（数据修改意味着可能存在同样的 KEY 对应着多条数据在数仓，去重配置会导致非预期的结果）等场景，在核对执行时数据不一致而产生误报。

解决方案：

- 使用平台自带的降噪功能。
- 选择单向核对，以数据最后变更的表作为驱动表，单向关联其他的表（LEFT JOIN) 做核对。

业务变更

业务逻辑或数据逻辑变更后而核对规则没有及时变更导致误报。

解决方案：业务系统变更后应及时修改核对规则，此步骤应该纳入软件发布流程的一个标准步骤。

数据仓库、数据同步链路问题

同步数据错误，如丢数据、数据重复等，导致任务成功执行完成，分区正常建立，但是数据是错误的。

解决方案：目前只能在处理异常时发现一例解决一例。核对平台在异常处理中可以提供及时的反馈，把问题通知到对应的表负责人，以求第一时间分析、解决问题。

变更等其他原因

- 数据订正了核对中多张表的部分数据，导致大量的核对不平。
- 第三方抖动、文件缺失等，导致核对不平。
- 变更错误导致核对不平。

10.资源优化

随着离线核对产品上运行的规则增长，大数据平台消耗的资源随之增加。如果对于规则的使用资源量以及执行效率关注较少，平台所依赖的计算资源长期处于满负荷运转的状态，核对的任务经常会出现积压的情况。资源优化的核心点是，减少规则所扫到的数据量。

去除不必要的大表依赖

根据当前的业务场景，尽量减少对大表的依赖，寻找可替代的小表。非必要的情况，不要使用全量表。

减少分钟表关联全量表

如果必须使用全量表的场景，需要增加时间过滤条件，取最近一段时间的数据（例如：最近一天的数据）。

避免规则空跑

可以使用指定时间运行核对任务。通过配置类 cron 表达式的方式在特定的周期运行任务，减少任务的空跑，以及对资源的浪费。

11.基础术语

术语	说明
资金安全监控	SOFAStack 的一款风险防控类型的产品，能帮助您发现资金损失的风险。该产品可以通过执行您自定义的规则，以小时频率等时效策略，离线、文件核对等核对方式，发现资金类数据问题，向您发出告警。您可以在第一时间收到告警，根据异常数据排查问题、分析原因，进而解决问题。
核对点	资金安全布防风险点的抽象集合，用于描述一个风险点的核对方式、动作内容及数据来源等的描述集合。可结合核对场景进行管理。
核对场景	用于对核对点进行分类和规划，以管理不同的资金风险场景。
核对规则	资金安全监控产品中的规则，也叫核对规则，是数据核对的最小单位。规则主要由 ODPS 表和 SQL 脚本构成，另外，您还可设置一些规则相关信息。规则的核对逻辑由 SQL 体现。
任务	资金安全监控产品中的任务（Task）指核对任务，即核对规则周期性执行的任务。每个周期有一组任务实例，每次执行对应一个任务实例。
任务周期	也叫核对周期，指核对任务的调度周期。如核对频率为天核对，就是一天执行一次，例如 2018 年 5 月 29 日执行的核对任务对应的周期是 20180529；如核对频率为小时核对，就是一小时执行一次，例如 2018 年 5 月 29 日 10 点执行的核对任务对应的周期是 2018052910。
异常	核对任务运行中发现数据有差异，系统会创建异常事件（Issue）。 **重要** 任务重跑时可能会重复产生差异。
差异数据	也叫差异，指核对任务执行时（执行 SQL）发现的不匹配核对规则的数据。您可以在异常详情里查看在某个核对周期中发现的抽样差异数据。
差异数量	核对任务执行时发现的差异数据的总数。 **重要** 异常详情页展示的差异数量是在当次任务周期中发现的差异数据总数，差异数据详情（抽样数据）页面仅展现有限数量抽样的差异数据。

术语	说明
核对频率	核对频率有：按天核对、小时核对、5分钟核对和自定义核对。天核对，指一天执行一次核对；小时核对，指一小时执行一次核对；5 分钟核对，指 5 分钟执行一次核对；自定义核对，指根据用户配置的 CRON 表达式触发核对执行。