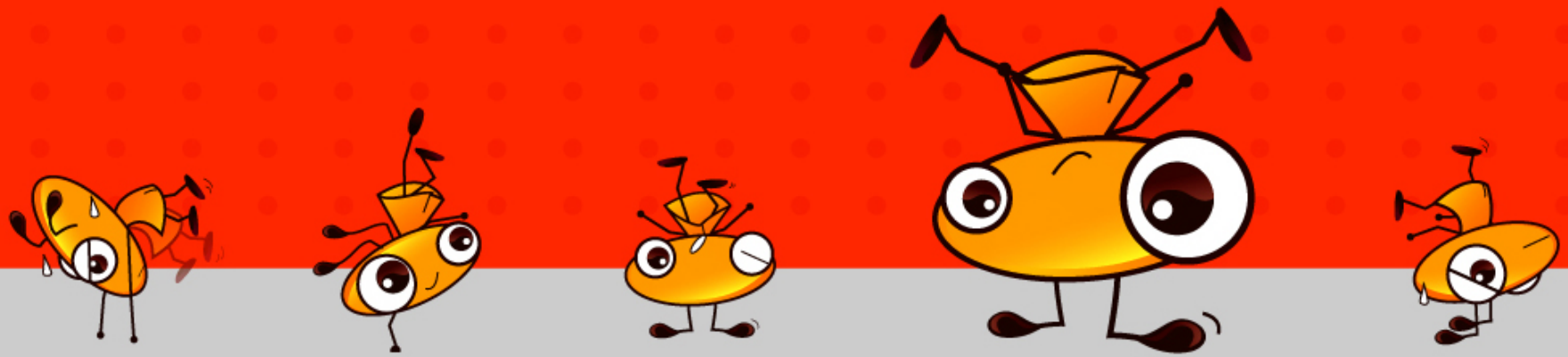
极限存储设计原理及实践

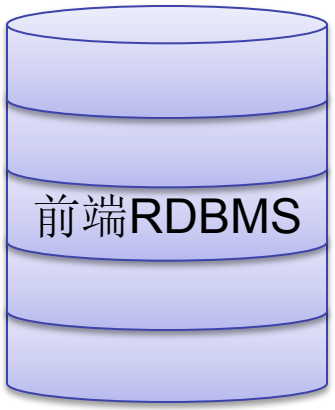
淘宝-数据平台与产品部　图海

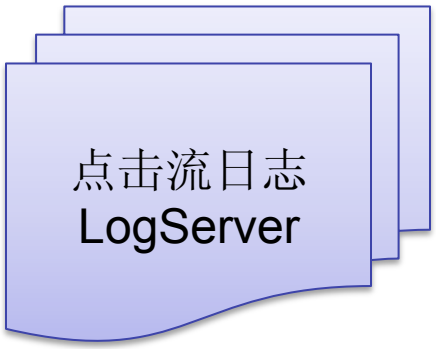
Monday, December 12, 11

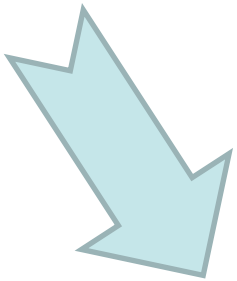


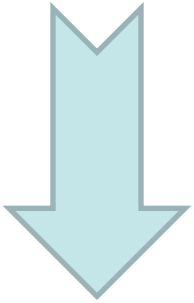

场景
方案
效果
优化

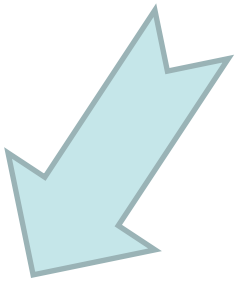
云梯的存储职责











DataX

TimeTunnel

DBSync

everyday…
everytime…

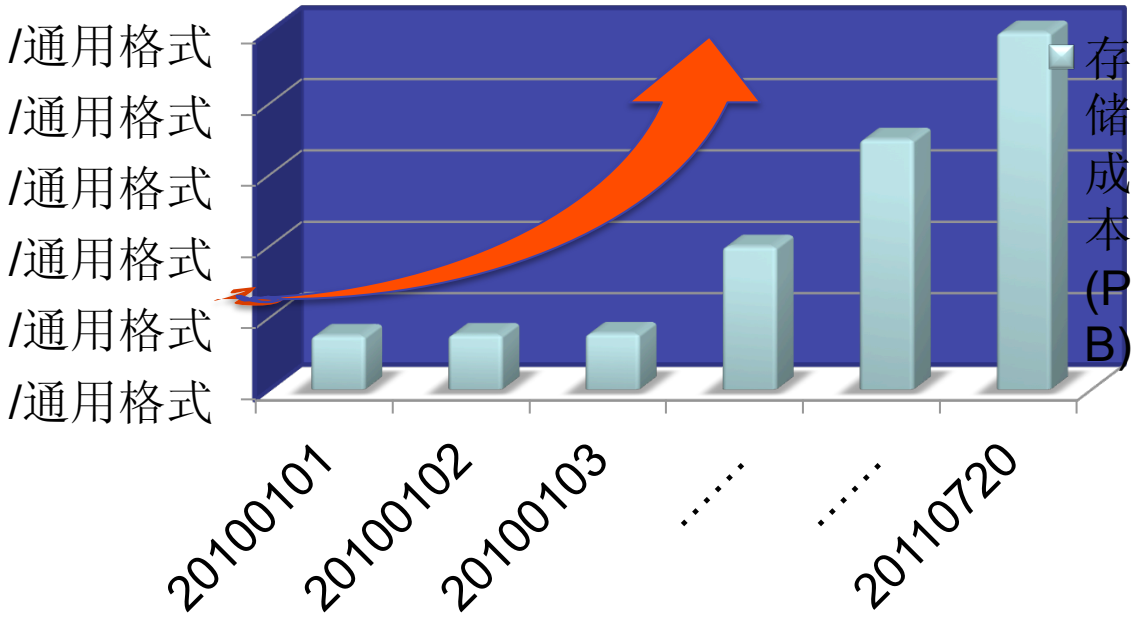
云梯1

商品表	
20080101	500G
20080102	502G
20080103	505G
……	…G
20110720	1000G

点击流日志	
……	200G
20080102	202G
20080103	205G
……	…G
20110720	300G

用户表	
……	200G
20080102	202G
20080103	205G
……	…G
20110720	300G

存储成本持续上升



怎么办?

怎么办?怎么办?怎么办?

删除历史数据。立竿见影。
省力又省事

商品表		
	20080101	500G
	20080102	502G
	20080103	505G
	……	…G
	20110720	1000G

“21世纪核心的竞争
是数据的竞争”
“谁拥有更多数据，
谁就拥有未来”

前端交易系统、商品中心、用户中心等出于效率的考虑，不会长期保存大量历史数据，而数据平台作为企业数据分析及挖掘的基础设施，天生具有保存历史数据的职责，非但如此，如何快速、高效的获取历史上任意一天的快照数据也成为设计历史数据存放方式时的重要考量。

数据分类

商品表:
- 商品ID
- 商品名
- 商品状态
- 创建时间
- 所属类目
- ……

交易表:
- 订单ID
- 支付ID
- 物流ID
- 支付时间
- 订单状态
- ……

典型操作:
- 新增商品/订单(new)
- 商品/订单状态变更(update)
- 商品下线/订单撤销(delete)

典型的数据库增删改操作

数据特点:
- 有业务主键，确保记录唯一性
- 全量快照数据量巨大(>1TB)，数据分析需要全量快照数据
- 每日变更量占比很少(远低于5%)
- 数据记录冗余度非常高

注:变更指发生增删改的记录

※当时存量数据中**70%**属于此类特征的业务数据，且记录冗余度高

数据分类

评价增量表:

- 评价ID
- 用户星级
- 用户昵称
- 评价记录
- 商品名称
- ……

数据特点:

- 有业务主键，确保记录唯一
- 数据只有新增操作，不会变更或删除
- 每天只需保留当天新增评价
- 数据记录冗余度基本为0

点击流日志:

- 记录时间
- IP地址
- 引用链接
- 机器ID
- 用户ID
- ……

数据特点:

- 没有业务主键
- 属于日志流水，每日新增数据
- 数据记录重复程度非常低，每条都基本唯一
- 数据记录冗余度基本为0

※存储总体占比不高，且数据冗余度较低，优化空间有限

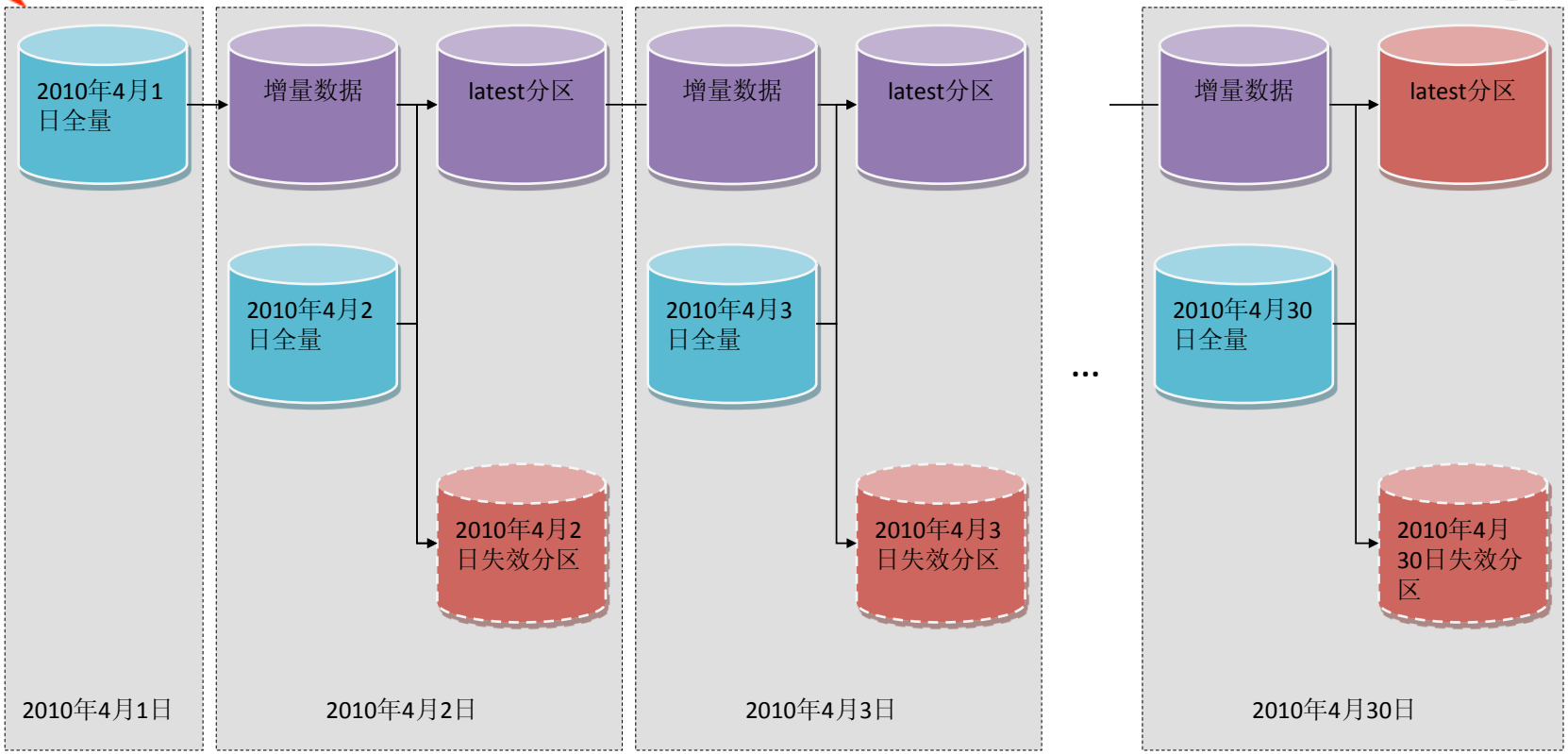
思考&讨论

问题:

如何设计方案达到以下效果?

- 减少/去除冗余数据，降低存储成本
- 保证快照数据的快速访问
- 对业务应用透明或降低应用改造成本

参考方案



注:类似于数据库系统中常见的增量备份或周期备份策略

优点:

- 易于理解，在数据库备份中广泛应用
- 实现较为简单

缺点:

- 访问快照数据成本太高
- 无法直接反应删除/被变更数据，需要额外设计
- 应用改造成本较高

记录生命周期

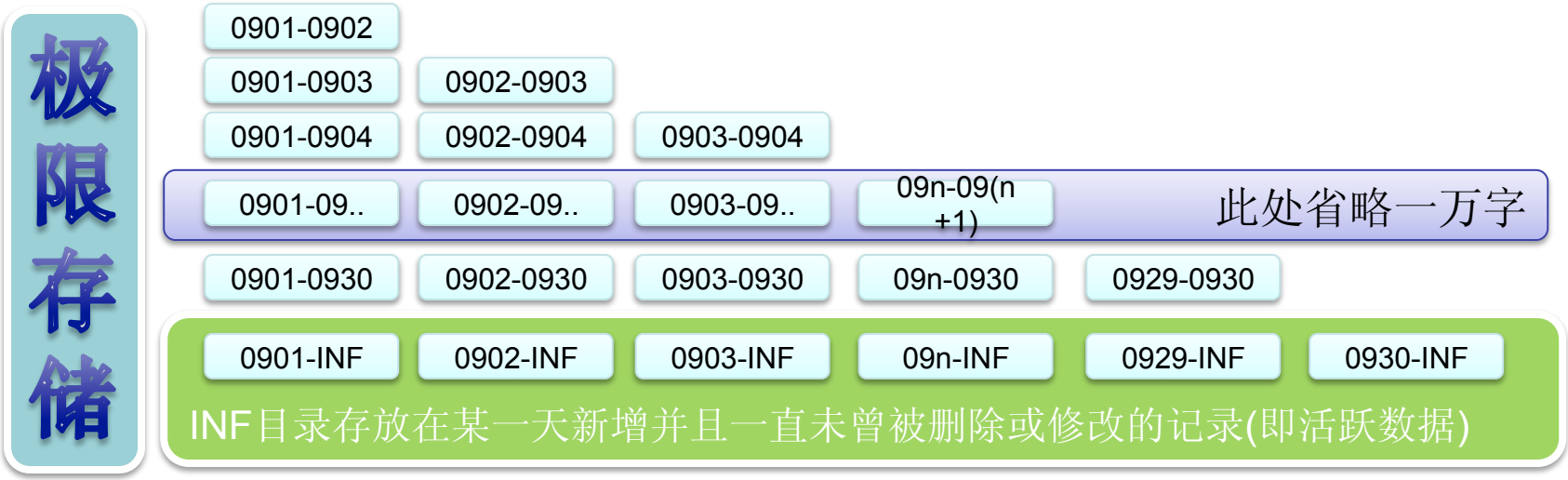
- ✓ 数据天生以行进行分割，行数据在数据库中称为一条数据记录(Record).
- ✓ 一条记录对应可能有Insert/Update/Delete操作
  - Insert通常对应一条全新的记录，意味着记录的新生
  - Delete通常是原有的记录被删除，意味着记录的死亡
  - Update是在原有的记录上修改某些字段，一条Update操作可以拆分为Delete/Insert原子对操作，即从记录的维度来看，相当于前一条记录死亡，后一条记录新生

因此，我们可以认为，任何一条记录(行数据)必定在历史上某天新生(start)，并在其后的某一天死亡(end)，而这个start-end对就定义为该记录的生命周期。

活跃数据和死亡数据

- 活跃数据
  - ✓一条记录，在其产生之后直至当天仍旧存活(未被Delete/Update)，那么我们认为它是一条活跃数据
  - ✓对于活跃数据，其产生(start)日期已经明确，但死亡(end)日期并不确定
  - ✓数据标签:start-INFINITY(无穷大),如20110401-INF
- 死亡数据
  - ✓一条记录，在当天以前就被更改（被Delete/Update），那么我们认为它是一条死亡了的数据
  - ✓对于死亡数据，其产生(start)和死亡(end)日期都已经明确
  - ✓数据标签:start-end,如20110401-200110423

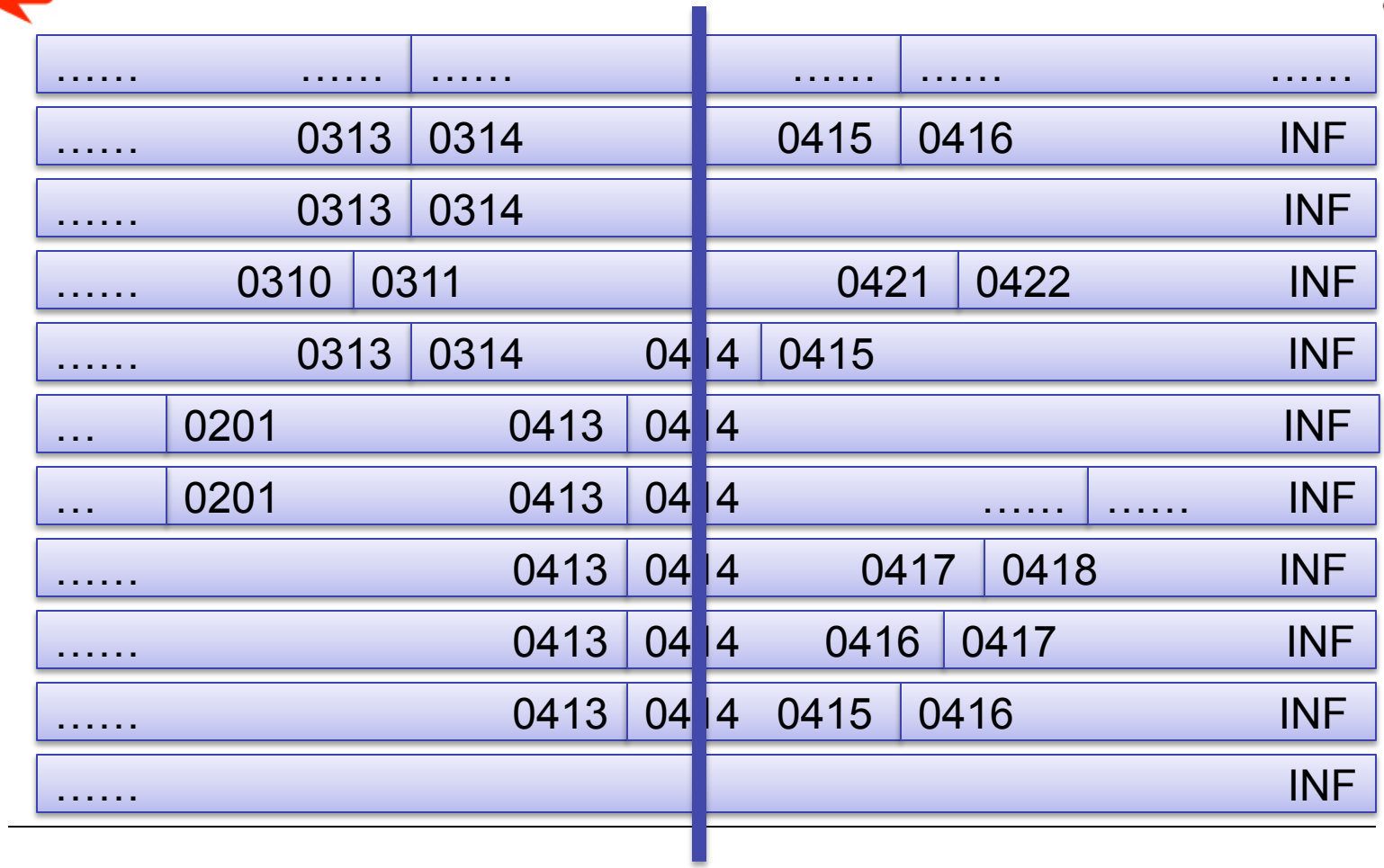
极限存储原理



三个结论:

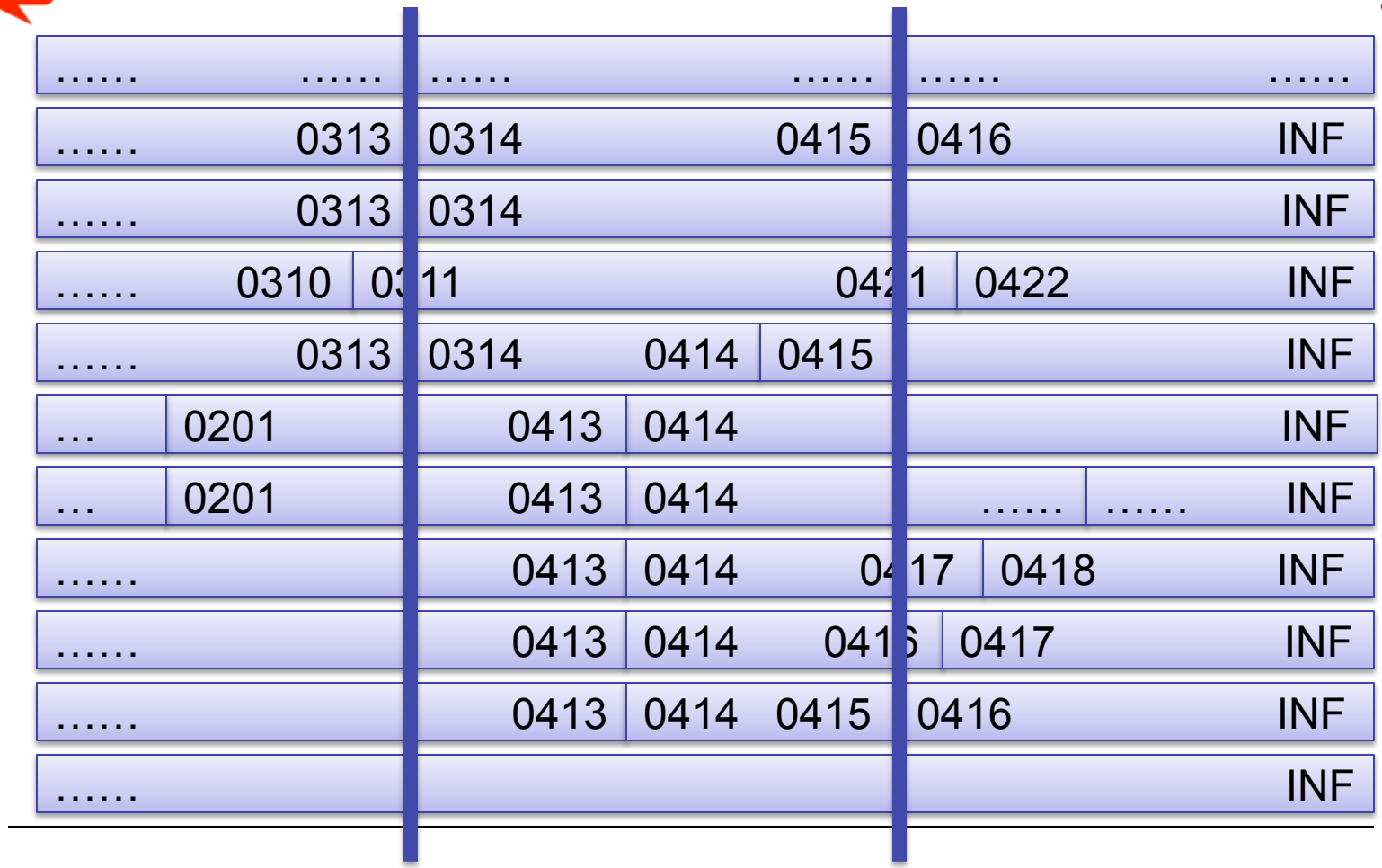
- ✓ 任意一条记录，由于其生命周期确定，必定对应唯一的一个数据标签
- ✓ 一个数据标签对应符合该生命周期的记录集合(该记录集合有为空的可能性)
- ✓ 历史上出现的所有记录，必然可以成功的划分到不同的生命周期数据标签里去

历史快照原理



- ✓ 所有被蓝色线条经过的数据标签，其数据内容组合起来即为0414这天的数据全量快照
- ✓ 同理，历史上任意一天的数据快照均可以该方式获得

历史区间快照原理

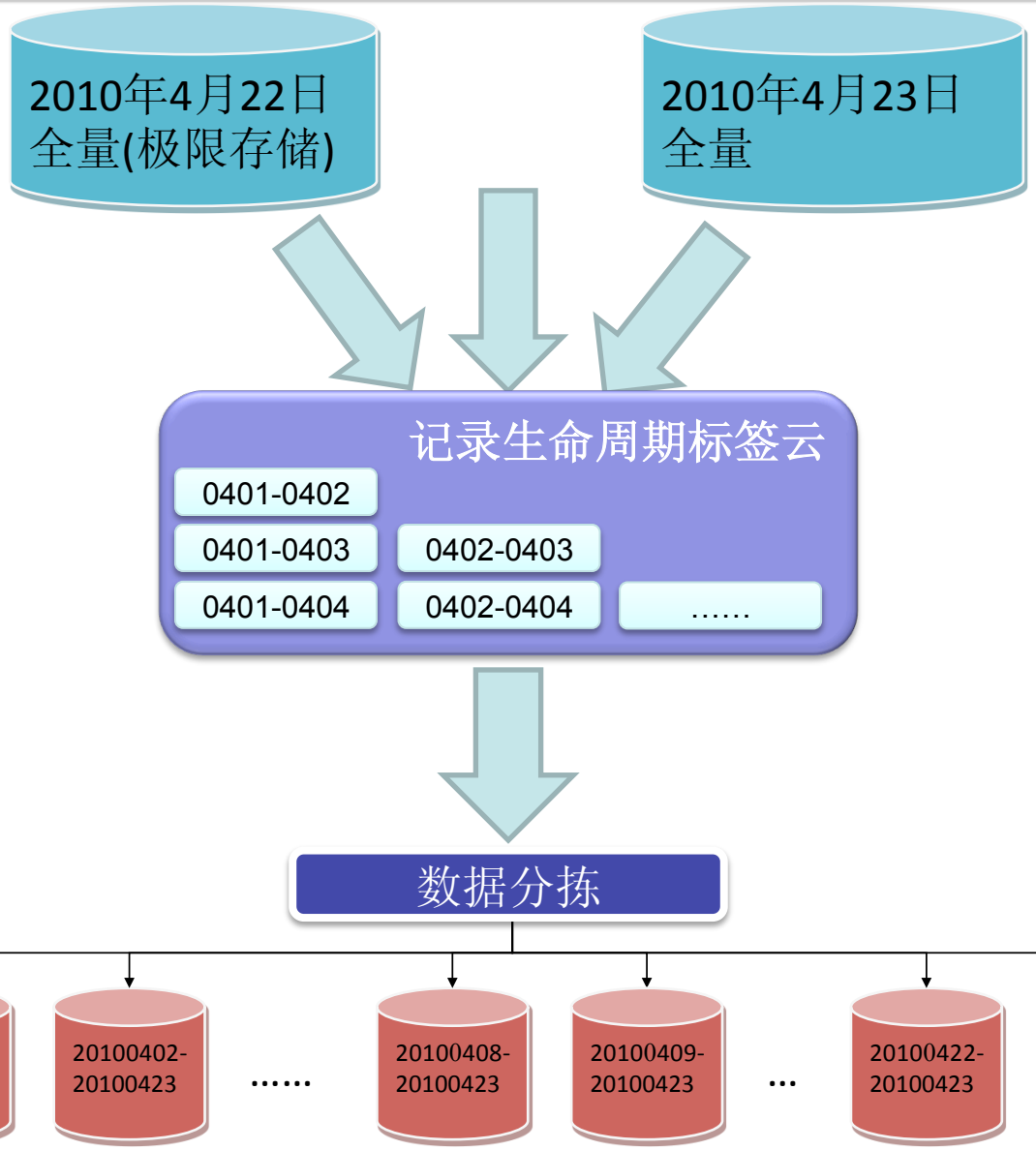


- ✓ 所有在两条蓝色线条以内以及穿过任意一条蓝色线条的数据标签，其数据内容组合起来即为0314-0415的数据全量快照

方案主体逻辑

- 包含以下主要步骤：
- 1.通过主键关联对比昨天全量和今天全量的数据差异，并将这些数据区分为活跃(Lived)或过期(Expired)数据。
- 2.对于对比的结果数据进行统计，获得每个生命周期下实际的数据条数，统计结果用来产生不同生命周期的记录到文件目录的映射。
- 3.使用mapreduce数据对第1步结果进行分发，相同生命周期的数据会被写入到对应的唯一的生命周期目录下(依赖2的统计结果)。
- 4.使用hive的双重分区映射生命周期目录，这样用户可以通过灵活的hive分区过滤来获得期望的数据。
- 5.数据验证，为了保证应用极限存储后结果的正确性，因此增加了数据条数对比的验证规则。

方案主体逻辑

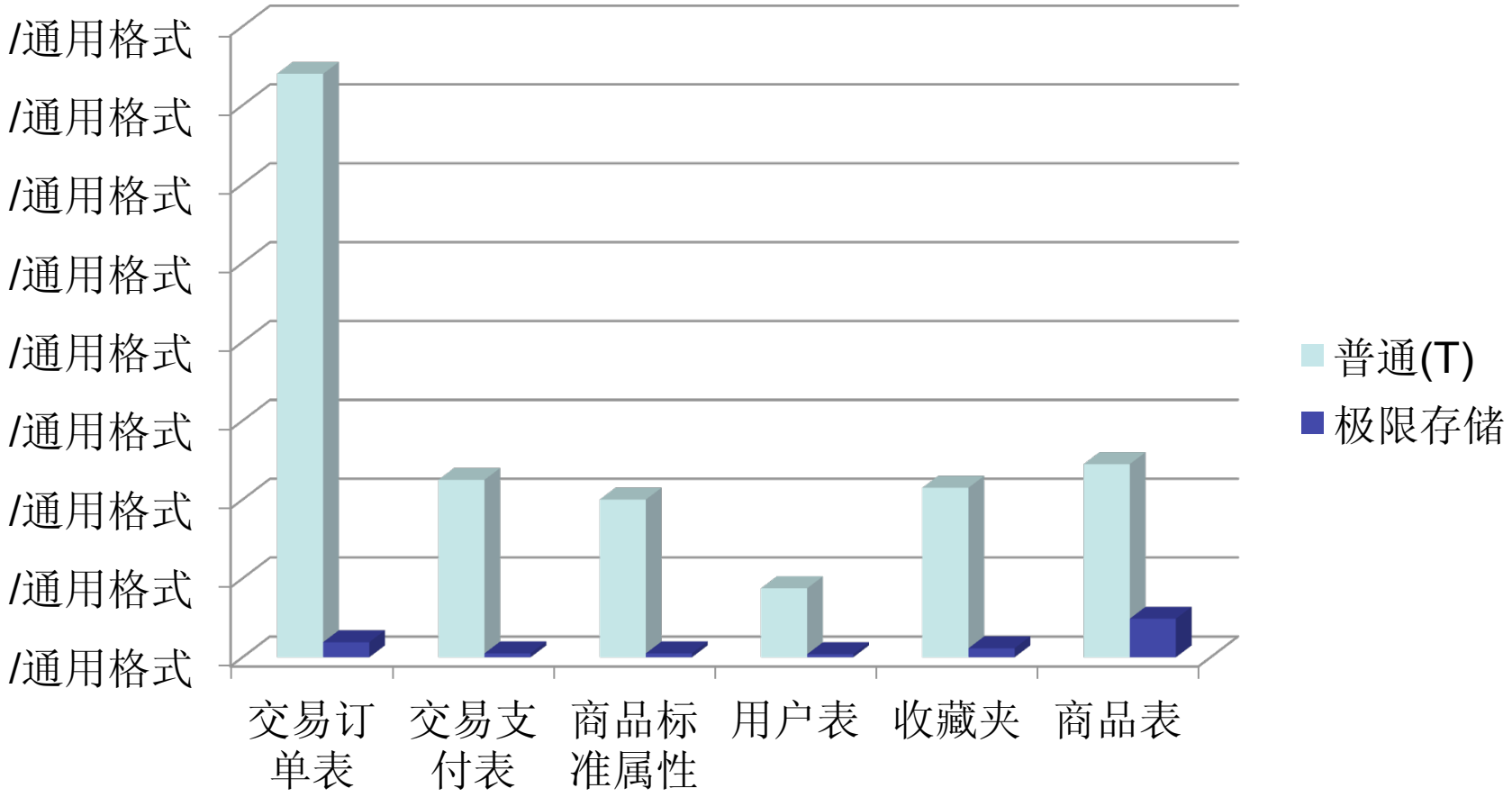


- Hive介绍
- 全文对比
- 数据统计
- 数据分拣
- 分区映射
- 数据验证

遇到的问题

- 产生的目录/文件数非常多
  - 产生目录数及文件数按日呈级数增长
  - 一个月产生465个目录,一年产生66795个目录
    文件数 = 目录数 * reduce数(如1000)
  - 对NameNode压力非常大
  - 对应分区非常多，Hive元数据库压力也很大
- 文件大小不均匀
- 如何快速访问任意一天/一段时期的快照数据
- 分拣中运行出错会导致数据损坏或丢失
- 不同月份数据并行运行丢失数据问题
- 单个数据标签内数据损坏/丢失导致一段时期内快照不准
- 其他的一些保护机制

应用效果



迄今为止已有30余种业务数据完成应用，累积节省存储达15PB。

极限存储使用方法

- Hive:
  - 取某天快照:

    select * from tb_users_exst where pt_start<=‘20100410’ and pt_end>‘20100410'
  - 取某天快照(UDF方式):

    select * from tb_users_exst where exst_pt(pt_start, pt_end, ' 20100410')
  - 取一段时间快照:

    select * from tb_users_exst where pt_start<='20100420' and pt_end>'20100410'
- Hadoop:

在调用setInputDir之前通过提供的方法获得生命周期目录列表，如下:

```
List<String> dateLists = DateListGenerator.generateExStoreListDirs("/group/taobao/taobao/hive/tb_users_exst", "20100410");
```

极限存储应用场景

◆查看一件商品2011年的变更历史:

- 不使用极限存储:

select * from tb_auctions where
pt>=20110101 and pt<=20110731

扫描数据量:450G*7*30 = 92TB

- 使用极限存储:

select * from tb_auctions_exst where
pt_start<=20110101 and pt_end >20110731

扫描数据量:450G*7*2[膨胀率] = 6TB(当前实现)
450G*2[膨胀率] = 900G(理想情况)

◆获取某天增量(delta)数据:

select * from tb_auctions where pt_start = 20110105

注:月头不适用，1号增量需要额外计算

极限存储方案优化

性能优化:

- ✓ 支持从极限存储全量&当天增量产生极限存储数据
  - ➢ 计算时间从2个小时下降至1个小时
  - ➢ 计算成本下降了50%
- ✓ 优化调整运行参数:

```
set io.sort.spill.percent=0.80;
set io.sort.mb=512;
set io.sort.factor=32;
set io.sort.record.percent=0.04;
set mapred.reduce.parallel.copies=8;
set mapred.job.shuffle.input.buffer.percent=0.70;
```

易用性优化:

Hive层增加hook，实现SQL自动替换，对用户及上层业务透明。

如: select * from tb_users where pt='20100410'

经过Hook层语意解析转换后变为:

select * from tb_users_exst where pt_start<='20100410' and pt_end>'20100410'

思考环节

- 源表主键不唯一会出现什么情况?如何快速处理?
- 与参考方案相比，优点在哪里？

联系方式:

微博:http://weibo.com/phrack　　淘图海

邮箱:tuhai@taobao.com

淘宝网
Taobao.com

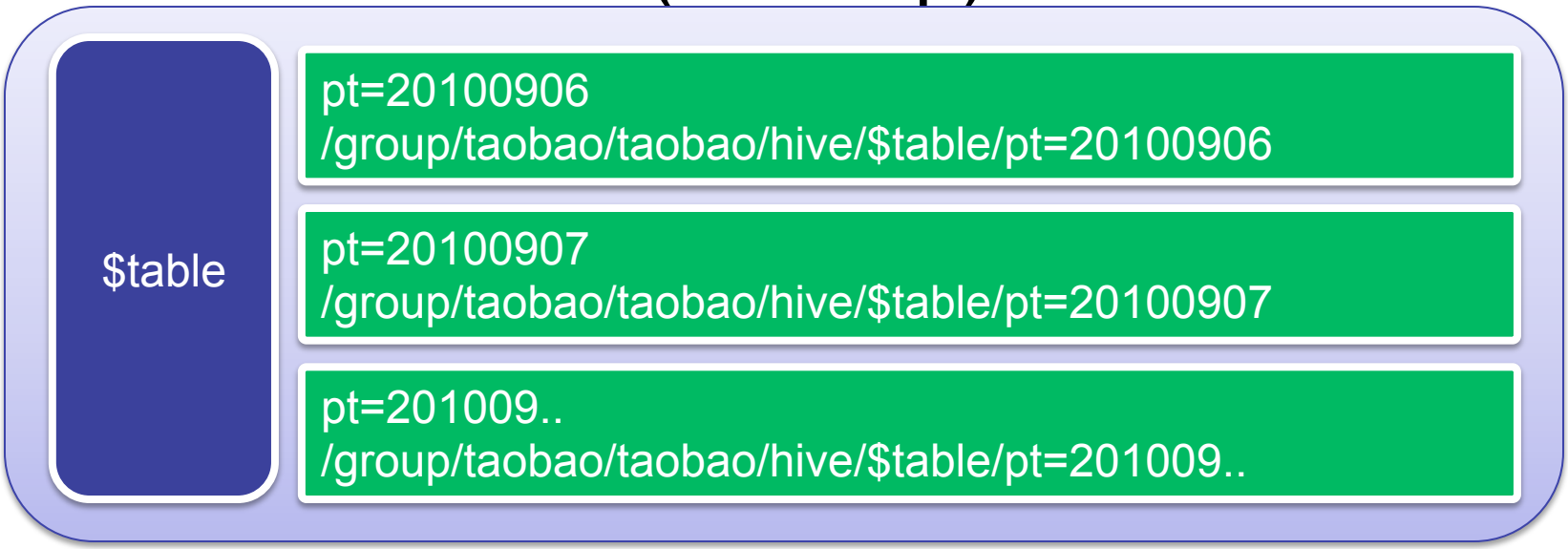
THANK YOU

Hive相关介绍

- 什么是Hive?
  - Hive 是建立在 Hadoop 上的、提供了类SQL界面的数据仓库基础构架。
- Hive分区表
  - 一个Hive表可以拥有一级或多级分区(组合分区)，每个分区对应一个目录，该目录下所有文件的数据合集为该Hive表的分区数据。
  - Hive分区表中的分区字段为伪列，不实际占用任何存储空间，仅通过分区名进行保存
  - 判断分区时可以通过>、<符号进行分区范围限定，如可以通过pt>=20110401 and pt=<20110430获取4月份的所有分区

Hive相关介绍

- Hive数据在云梯(Hadoop)上的存放形式



每个目录下有若干个数据文件，通常文件中每行数据对应Hive表的一条记录，列之前通过给定的分割符进行分割。

数据文件可以是压缩的，也可以是非压缩的，只要和Hive元数据中保存的分区信息一致即可。

返回

全文对比

通过Hive实现对相邻两天的数据全量对比，区分活跃与死亡数据，并同时获得其生命周期信息：

```
FROM
  (SELECT * FROM tb_users_exst   WHERE pt_start <= '20100422' AND pt_end > '20100422' ) o
FULL OUTER JOIN
  (SELECT * FROM tb_users WHERE pt = '20100423000000') n
ON
  o.id = n.id
INSERT OVERWRITE TABLE t_ext_20100423_tb_users PARTITION (pt='EXPIRED')
SELECT …  o.pt_start,  '20100423 '
WHERE
  n.id IS NULL
  OR (n.id <> o.id)
OR (n.nick <> o.nick)
…
INSERT OVERWRITE TABLE t_ext_20100423_tb_users PARTITION (pt='LIVE')
SELECT …  if(  o.id IS NULL
   OR (n.nick <> o.nick) …, '20100423',o.pt_start) as birth_date, NULL as expired_date
WHERE
  n.id IS NOT NULL
```

t_ext_20100423_tb_users比tb_users多了两个字段：

birth_date：记录的产生日期，不可能为NULL

expired_date：记录的消亡日期，如果未死则为NULL

返回

数据统计

通过Hive统计落在各个生命周期区间内的数据条数:

```
INSERT OVERWRITE TABLE t_exs_tb_users
PARTITION (pt='20100423000000')
SELECT
  birth_date,
  expired_date,
  count(1)
FROM
  t_ext_20100423_tb_users
GROUP BY
  birth_date,
  expired_date
```

注:这里的统计信息主要用来为下一步mapreduce作业提供参考，比如计算最终合理的文件数。

返回

数据分拣

经过前面的处理，t_ext_20100423_tb_users两个分区(LIVED,EXPIRED)下的数据其最后两列分别为birth_date和expired_date,即对应数据记录的产生和死亡日期，因此mapreduce可以根据这个信息得知该目录应该存放至那个生命周期目录下。

生命周期目录下云梯1上表现为如下形式:

```
s_bmw_usres_exst
`-- 201004                          # 每月数据存放在月份目录下
    |-- 201004                      # 该目录下存放本月新增且至今未死亡的数据
    |   |-- 20100401                # 0401产生且未死亡
    |   |-- 20100402
    ...
    |   `-- 20100423                # 0423产生且未死亡
    |-- 20100401-20100402           # 0401产生且0402死亡(被删除或更新)
    |-- 20100401-20100403           # 0401产生且0403死亡(被删除或更新)
    |-- 20100401-20100404
    ...
    |-- 20100421-20100423
    `-- 20100422-20100423
```

对于以下两条t_ext_20100423_tb_users表的记录，mapreduce会分别将其分发到/201004/20100403和/201004/20100403-20100405目录下。

```
Record1   20100403    NULL
Record2   20100403    20100405
```

返回

分区映射

极限存储目标表必定存在以下两个分区：
Pt_start：起始日期，一定是一个合法8位日期
Pt_end：死亡日期，对于活跃数据其值为$month_INFINITY

仍然以上页的说明为例，说明分区的映射方式：

```
s_bmw_usres_exst
`-- 201004
    |-- 201004
    |   |-- 20100401            #pt_start= '20100401' , pt_end = '201004_INFINITY'
    |   |-- 20100402            #pt_start= '20100402' , pt_end= '201004_INFINITY'
    ...
    |   `-- 20100423            #pt_start= '20100423' , pt_end= '201004_INFINITY'
    |-- 20100401-20100402          # pt_start= '20100401' , pt_end= '20100402'
    |-- 20100401-20100403          # pt_start= '20100401' , pt_end= '20100402'
    |-- 20100401-20100404
    ...
    |-- 20100421-20100423
    `-- 20100422-20100423          # pt_start= '20100422' , pt_end= '20100423'
```

返回

数据验证

条数验证:

- SELECT count(1) FROM tb_users_exst WHERE pt_start <= '20100423' AND pt_end > '20100423';
- SELECT count(1) FROM tb_users WHERE pt='20100423000000';

内容验证:

类似于方案第一步中的数据全文对比，对每条记录中的每个字段进行一致性比较，从而最终确保数据内容正确无误

返回