> 关注公众号：大数据技术派，回复：资料，领取1024G资料。

拉链表是针对数据仓库设计中表存储数据的方式而定义的，顾名思义，所谓拉链，就是记录历史。**记录一个事物从开始，一直到当前状态的所有变化的信息。**

下面就是一张拉链表，存储的是用户的最基本信息以及每条记录的生命周期。我们可以使用这张表拿到最新的当天的最新数据以及之前的历史数据。

注册日期	用户编号	手机号码	t_start_date	t_end_date
2017-01-01	001	111111	2017-01-01	9999-12-31
2017-01-01	002	222222	2017-01-01	2017-01-01
2017-01-01	002	233333	2017-01-02	9999-12-31
2017-01-01	003	333333	2017-01-01	9999-12-31
2017-01-01	004	444444	2017-01-01	2017-01-01
2017-01-01	004	432432	2017-01-02	2017-01-02
2017-01-01	004	654321	2017-01-03	9999-12-31
2017-01-02	005	555555	2017-01-02	2017-01-02
2017-01-02	005	115115	2017-01-03	9999-12-31
2017-01-03	006	666666	2017-01-03	9999-12-31

说明：

- `t_start_date` 表示该条记录的生命周期开始时间，`t_end_date` 表示该条记录的生命周期结束时间；
- `t_end_date` = '9999-12-31' 表示该条记录目前处于有效状态；
- 如果查询当前所有有效的记录，则 `select * from user where t_end_date = '9999-12-31'`
- 如果查询 2017-01-01 的历史快照，则 `select * from user where t_start_date <= '2017-01-01' and end_date >= '2017-01-01'`，这条语句会查询到以下记录：

拉链表的使用场景

在数据仓库的数据模型设计过程中，经常会遇到下面这种表的设计：

1. 有一些表的数据量很大，比如一张用户表，大约10亿条记录，50个字段，这种表，即使使用ORC压缩，单张表的存储也会超过100G，在HDFS使用双备份或者三备份的话就更大一些。
2. 表中的部分字段会被update更新操作，如用户联系方式，产品的描述信息，订单的状态等等。
3. 需要查看某一个时间点或者时间段的历史快照信息，比如，查看某一个订单在历史某一个时间点的状态。
4. 表中的记录变化的比例和频率不是很大，比如，总共有10亿的用户，每天新增和发生变化的有200万左右，变化的比例占的很小。

对于这种表的设计？下面有几种方案可选：

- 方案一：每天只留最新的一份，比如我们每天用datax抽取最新的一份全量数据到Hive中。
- 方案二：每天保留一份全量的切片数据。
- 方案三：使用拉链表。

为什么使用拉链表

方案一：每天只留最新的一份

这种方案就不用多说了，实现起来很简单，每天drop掉前一天的数据，重新抽一份最新的。
优点很明显，节省空间，一些普通的使用也很方便，不用在选择表的时候加一个时间分区什么的。
缺点同样明显，没有历史数据，先翻翻旧账只能通过其它方式，比如从流水表里面抽。

方案二：每天保留一份全量的切片数据

每天一份全量的切片是一种比较稳妥的方案，而且历史数据也在。
缺点就是存储空间占用量太大太大了，如果对这边表每天都保留一份全量，那么每次全量中会保存很多不变的信息，对存储是极大的浪费。
当然我们也可以做一些取舍，比如只保留近一个月的数据？但是，需求是无耻的，数据的生命周期不是我们能完全左右的。

方案三：拉链表

拉链表在使用上基本兼顾了我们的需求。
首先它在空间上做了一个取舍，虽说不像方案一那样占用量那么小，但是它每日的增量可能只有方案二的千分之一甚至是万分之一。
其实它能满足方案二所能满足的需求，既能获取最新的数据，也能添加筛选条件也获取历史的数据。
所以我们还是很有必要来使用拉链表的。

拉链表的设计

在Mysql关系型数据库里的user表中信息变化

在2017-01-01表中的数据是：

注册日期	用户编号	手机号码
2017-01-01	001	111111
2017-01-01	002	222222
2017-01-01	003	333333
2017-01-01	004	444444

在`2017-01-02`表中的数据是，用户`002`和`004`资料进行了修改，`005`是新增用户:

注册日期	用户编号	手机号码	备注
2017-01-01	001	111111	无
2017-01-01	002	233333	（由222222变成233333）
2017-01-01	003	333333	无
2017-01-01	004	432432	（由444444变成432432）
2017-01-02	005	555555	（2017-01-02新增）

在`2017-01-03`表中的数据是，用户`004`和`005`资料进行了修改，`006`是新增用户:

注册日期	用户编号	手机号码	备注
2017-01-01	001	111111	
2017-01-01	002	233333	
2017-01-01	003	333333	
2017-01-01	004	654321	(由432432 变成 654321)
2017-01-02	005	115115	(由555555 变成 115115)
2017-01-03	006	115115	(2017-01-03 新增)

如果在数据仓库中设计成历史拉链表保存该表，则会有下面这样一张表，这是最新一天（即2017-01-03）的数据：

注册日期	用户编号	手机号码	t_start_date	t_end_date
2017-01-01	001	111111	2017-01-01	9999-12-31
2017-01-01	002	222222	2017-01-01	2017-01-01
2017-01-01	002	233333	2017-01-02	9999-12-31
2017-01-01	003	333333	2017-01-01	9999-12-31
2017-01-01	004	444444	2017-01-01	2017-01-01
2017-01-01	004	432432	2017-01-02	2017-01-02
2017-01-01	004	654321	2017-01-03	9999-12-31
2017-01-02	005	555555	2017-01-02	2017-01-02
2017-01-02	005	115115	2017-01-03	9999-12-31
2017-01-03	006	666666	2017-01-03	9999-12-31

说明：

- `t_start_date` 表示该条记录的生命周期开始时间，`t_end_date` 表示该条记录的生命周期结束时间；
- `t_end_date = '9999-12-31'` 表示该条记录目前处于有效状态；
- 如果查询当前所有有效的记录，则 `select * from user where t_end_date = '9999-12-31'`
- 如果查询 2017-01-01 的历史快照，则 `select * from user where t_start_date <= ‘2017-01-01’ and end_date >= '2017-01-01'`。

拉链表的实现与更新

Hive中实现拉链表

1. 我们需要一张 `ODS` 层的用户全量表。至少需要用它来初始化。
2. 每日的用户更新表。

而且我们要确定拉链表的时间粒度，比如说拉链表每天只取一个状态，也就是说如果一天有3个状态变更，我们只取最后一个状态，这种天粒度的表其实已经能解决大部分的问题了。

获取每日的用户增量

监听`Mysql`数据的变化，比如说用`Canal`，最后合并每日的变化，获取到最后的一个状态。
假设我们每天都会获得一份切片数据，我们可以通过取两天切片数据的不同来作为每日更新表，这种情况下我们可以对所有的字段先进行`concat`，再取`md5`，这样就ok了。
流水表，有每日的变更流水表

表结构

`ods`层的`user`表

```
CREATE EXTERNAL TABLE ods.user (
  user_num STRING COMMENT '用户编号',
  mobile STRING COMMENT '手机号码',
  reg_date STRING COMMENT '注册日期'
COMMENT '用户资料表'
PARTITIONED BY (dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION '/ods/user';
)
```

`ods`层的`user_update`表

```
CREATE EXTERNAL TABLE ods.user_update (
  user_num STRING COMMENT '用户编号',
  mobile STRING COMMENT '手机号码',
  reg_date STRING COMMENT '注册日期'
COMMENT '每日用户资料更新表'
PARTITIONED BY (dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION '/ods/user_update';
)
```

拉链表

```
CREATE EXTERNAL TABLE dws.user_his (
  user_num STRING COMMENT '用户编号',
  mobile STRING COMMENT '手机号码',
  reg_date STRING COMMENT '用户编号',
  t_start_date ,
  t_end_date
COMMENT '用户资料拉链表'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION '/dws/user_his';
)
```

更新

假设已经初始化了`2017-01-01`的日期，然后需要更新`2017-01-02`那一天的数据

```
INSERT OVERWRITE TABLE dws.user_his
SELECT * FROM
(
    SELECT A.user_num,
           A.mobile,
           A.reg_date,
           A.t_start_time,
           CASE
                WHEN A.t_end_time = '9999-12-31' AND B.user_num IS NOT NULL THEN
'2017-01-01'
                ELSE A.t_end_time
           END AS t_end_time
    FROM dws.user_his AS A
    LEFT JOIN ods.user_update AS B
    ON A.user_num = B.user_num
UNION
    SELECT C.user_num,
           C.mobile,
           C.reg_date,
           '2017-01-02' AS t_start_time,
           '9999-12-31' AS t_end_time
    FROM ods.user_update AS C
) AS T
```

补充

拉链表和流水表

流水表存放的是一个用户的变更记录，比如在一张流水表中，一天的数据中，会存放一个用户的每条修改记录，但是在拉链表中只有一条记录。
这是拉链表设计时需要注意的一个粒度问题。我们当然也可以设置的粒度更小一些，一般按天就足够。

查询性能

链表当然也会遇到查询性能的问题，比如说我们存放了5年的拉链数据，那么这张表势必会比较大，当查询的时候性能就比较低了，个人认为两个思路来解决：

1. 在一些查询引擎中，我们对start_date和end_date做索引，这样能提高不少性能。
2. 保留部分历史数据，比如说我们一张表里面存放全量的拉链表数据，然后再对外暴露一张只提供近3个月数据的拉链表。

猜你喜欢

Hadoop3数据容错技术（纠删码）

Hadoop 数据迁移用法详解

Flink实时计算topN热榜

数仓建模分层理论

数仓建模方法论

大数据组件重点学习这几个