# 目录

## 1.增量表加工模板

通过全量表加工出增量表，注意排除时间戳和批次时间，且主键不能重复不能为空，如果有时间戳或者批次时间需要使用临时表手工筛选字段。

```
-- 功能能描述：增量表加工模板
-- 注意：
---       1. concat_ws('', *)并不会将分区字段dt拼接进去
--           但是需要把时间戳和批次字段排除

-- 输入表：credit_cust.dwd_cc_template_a_d（全量表）
-- 输出表：credit_cust.dwd_cc_template_i_d（增量表）
-- 主键  ：pk
select from_unixtime(unix_timestamp()) as insertdt
     , t.pk
     , t.col_1
     , t.col_2
     ...
from credit_cust.dwd_cc_template_a_d t
left join credit_cust.dwd_cc_template_a_d a
on a.dt='${start-1d|yyyyMMdd}'
and concat_ws('',t.*)=concat_ws('',a.*)
where t.dt='${start|yyyyMMdd}'
and a.pk is null
```

## 2.累全量表加工模板

累全量有2个步骤，步骤一：初始化全量表，包含增量表至今的所有分区。步骤二：对全量表的记录进行更新和追加。

```
-- 功能能描述：累全量表加工模板
-- 注意：
-- 输入表：credit_cust.dwd_cc_template_i_d（增量表）
-- 输出表：credit_cust.dwd_cc_template_a_d（全量表）
-- 主键  ：pk

-- init
select from_unixtime(unix_timestamp()) as insertdt
     , from_unixtime(unix_timestamp()) as updatetdt
     , t.pk
     , t.col_1
     , t.col_2
     ...
from credit_cust.dwd_cc_template_i_d t
where t.dt>='20210609'                    -- 增量表的第一个分区
and t.dt<='${start|yyyyMMdd}'
-- init_end

select if(t.pk is not null, t.insertdt, from_unixtime(unix_timestamp()))  as insertdt
     , if(a.pk is not null, from_unixtime(unix_timestamp()), t.updatetdt) as updatetdt
     , coalesce(a.pk, t.pk)                                               as pk
     , if(a.pk is not null, a.col_1, t.col_1) as col_1
     , if(a.pk is not null, a.col_2, t.col_2) as col_1
     ...
from credit_cust.dwd_cc_template_a_d t
full join credit_cust.dwd_cc_template_i_d a
on a.dt='${start|yyyyMMdd}'
and t.pk=a.pk
where t.dt='${start-1d|yyyyMMdd}'
```

### 3.使用最新分区模板

当需要使用的表不定期月更新的时候，可以多选择几个分区，然后使用有数据的最新分区即可。

```
-- 功能能描述：使用最新分区模板
-- 注意：
-- 输入表：credit_cust.ods_cc_template（每月不定期更新）
-- 输出表：credit_cust.adm_cc_template_d（每日更新）
-- 主键   ：pk

select t.pk
     , t.col_1
     , t.col_2
     ...
from credit_cust.ods_cc_template t
where t.dt in('${mstart-1m-1d|yyyyMMdd}', '${mstart-1d|yyyyMMdd}')
and t.dt in (
               select max(if(t.num>0, t.dt, '0')) as dt
               from (
                         select count(*) as num
                              , t.dt     as dt
                         from credit_cust.ods_cc_template t
                         where t.dt in ('${mstart-1m-1d|yyyyMMdd}', '${mstart-1d|yyyyMMdd}')
                     ) t
           )
```

### 4.带顺序的字符串拼接

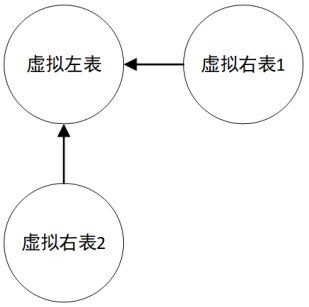
为了产出按照特定顺序拼接的新字段，可以在待拼接字段前加上特定的顺序，然后使用`collect_list`函数将带顺序的字符串变为字符串数组，随后使用`sort_array`函数对字符串数组进行排序，最后使用`concat_ws`函数对排序后的字符串数组进行拼接。

```
-- 功能能描述：带顺序的字符串拼接
-- 注意：
-- 输入表：credit_cust.dwd_cc_template
-- 输出表：credit_cust.dwd_cc_template_plus
-- 主键   ：pk

with dwd_cc_template as
(
    select 'id-1' as id, 'F' status_code, '20210609' as update_time union all
    select 'id-1' as id, 'S' status_code, '20210610' as update_time union all
    select 'id-1' as id, 'F' status_code, '20210611' as update_time union all
    select 'id-1' as id, 'F' status_code, '20210612' as update_time union all
    select 'id-2' as id, 'F' status_code, '20210609' as update_time union all
    select 'id-2' as id, 'F' status_code, '20210610' as update_time
)
select t.id
     , concat_ws('|', sort_array(collect_list(concat(t.rn, '-', coalesce(t.status_code,''))))) as status_cod
from (
          select t.*
               , lpad(row_number() over(partition by t.id order by t.update_time asc),10,'0') as rn
          from dwd_cc_template t
) t
group by t.id
```

## 5.反模式：`full join`多表关联

通过关联健进行`full join`关联，如果关联键既存在于左表又存在于右表，则会关联成一条记录。多表进行`full join`关联时，假设存在一个虚拟的左表，关联键存在于右表1，但是不存在于左表，该记录会被保留。关联键存在于右表2，不存在于左表，该记录会被保留。如果上述右表1的记录和右表2的记录的关联键相同，则最终结果会重复，如下图所示。




因此在设计结果表（一般会关联多张表）时，使用一张主表进行做关联是最简单的防止重复的设计模式。

## 6.反模式：不同粒度的表关联

不同粒度的表进行`full join`本质是不同的主键进行关联。例如，表A的粒度比较低（销售人员销售的不同类型的合同的数量，因此表A的粒度是销售员+签约类型），而表B的粒度更高（销售人员合同的推销次数，因为只有在签约的时候才会确定类型，因此表B的粒度是销售员），此时如果使用`full join`关联A表和B表，会导致统计出错，如下表所示。

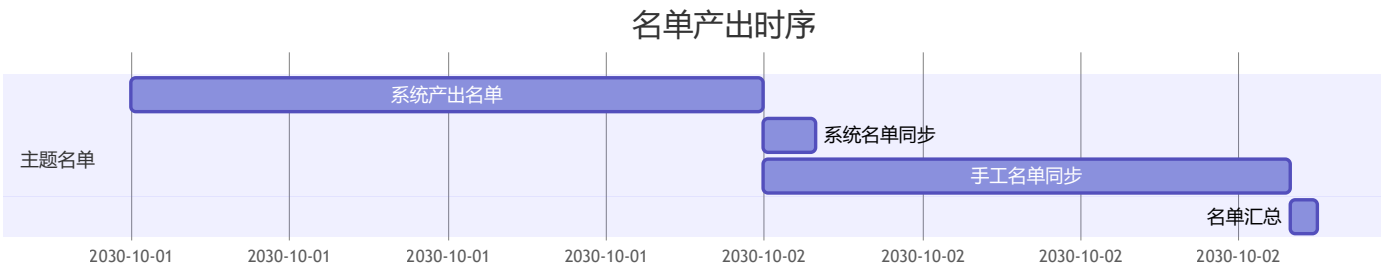
```
     签约统计表                                                    结果表
+----------+----+---+                                +----------+----+------+--------+
|sales_code|type|cnt|                 销售统计表      |sales_code|type|cnt_qy| cnt_xs |
+----------+----+---+            +----------+---+    +----------+----+------+--------+
|   S001   | c1 | 10|            |sales_code|cnt|    |   S001   | c1 |  10  |   100  |
+----------+----+---+ left join  +----------+---+  →  +----------+----+------+--------+
|   S001   | c2 | 15|            |   S001   |100|    |   S001   | c2 |  15  |   100  |
+----------+----+---+            +----------+---+    +----------+----+------+--------+
|   S001   | c3 | 20|                                |   S001   | c3 |  20  |   100  |
+----------+----+---+                                +----------+----+------+--------+
```

如上表所示，结果表中业务员的销售次数总计是300，c1类合同的签约数量是10，c2类合同的签约数量是15，c3类合同的签约数量是20。但是业务员的总的推销次数只有100次，因此正确的做法是对签约表进行汇总，提高粒度之后再关联，如下表。

```
    销售统计表                  签约统计表                  结果表
+----------+---+            +----------+---+     +----------+------+------+
|sales_code|cnt|            |sales_code|cnt|     |sales_code|cnt_xs|cnt_db|
+----------+---+  left join  +----------+---+  →  +----------+------+------+
|   S001   |100|            |   S001   | 45|     |   S001   |  100 |  45  |
+----------+---+            +----------+---+     +----------+------+------+
```

## 7.多通路查询中介模板

由于大数据会对接企业的几乎所有的应用系统，因此遇到多个部门都需要与外部组织通过文件进行数据交互时，需要将同一个主题的名单通过名单宽表进行汇总去重，同时各个通路的名单应该遵循以下标准，以满足后续的功能扩展。




1.名单来源表需要是分区表
2.使用证件号作为主键 Id_no
3.各个渠道需要约定优先级 Sort_prority
4.各个通路需要指定查询时间 Upload_date 且格式限定为 yyyy-MM-dd

示例.

| 名称 | 类型 | 约束 | 说明 |
| --- | --- | --- | --- |
| `cust_no` | `string` | 不允许为空 | 集团内统一客户号 |
| `id_no` | `string` | 不允许为空 | 主键，集团内外唯一标识 |
| `id_name` | `string` | 不允许为空 | 姓名 |
| `channel` | `string` | 不允许为空 | 渠道、部门 |
| `upload_date` | `string` | 不允许为空 | 查询时间：支持分批查询、当日查询 |
| `sort_prority` | `string` | 不允许为空 | 优先级，用于去重 |
| `reuse_day` | `string` | 不允许为空 | 复用天数（可选） |

不同渠道名单统一备份到ods层（手工表每日定时备份，不依赖），备份之后在ods层进行合并，左后在dwd层进行过滤，筛选需要当日查询的名单（即upload_date=dt+1day）然后按照优先级去重。