# Forecast Flag Calculation: Step Description

Forecast Flag table contains information regarding full present lifecycle history for each quadruple product | location|customer|distr\_channel needed for handling forecast generation process. Forecast Flag has to be used in demand restoration step, Pre-ABT preparation step, forecast preparation and forecast consolidation steps, downstream integration step.

Regarding Service Model Forecasts Flag step is a subservice of the 1.0.1 Lifecycle Info processing.

## Code Realization Requirements

Use python

# Input Data

The initial data for the forecast flag definition are listed below. These tables should be present in the system before this step for example in STG or in DDS area.

## PRODUCT

|  |  |
| --- | --- |
| **DDS.IN\_PRODUCT** | |
| Column Name | Description |
| PRODUCT\_LVL\_ID1 | Hierarchy level ID 1 |
| PRODUCT\_LVL\_ID2 | Hierarchy level ID 2 |
| PRODUCT\_LVL\_ID3 | Hierarchy level ID 3 |
| PRODUCT\_LVL\_ID4 | Hierarchy level ID 4 |
| PRODUCT\_LVL\_ID5 | Hierarchy level ID 5 |
| PRODUCT\_LVL\_ID6 | Hierarchy level ID 6 |
| PRODUCT\_LVL\_ID7 | Hierarchy level ID 7 |
| PRODUCT\_ID | Product ID (the lowest level of the product hierarchy) |

## LOCATION

|  |  |
| --- | --- |
| **DDS.IN\_LOCATION** | |
| Column Name | Description |
| LOCATION\_LVL\_ID1 | Hierarchy level ID 1 |
| LOCATION\_LVL\_ID2 | Hierarchy level ID 2 |
| LOCATION\_LVL\_ID3 | Hierarchy level ID 3 |
| LOCATION\_LVL\_ID4 | Hierarchy level ID 4 |
| LOCATION\_LVL\_ID5 | Hierarchy level ID 5 |
| LOCATION\_ID | Location ID |

## CUSTOMER

|  |  |
| --- | --- |
| **DDS.IN\_CUSTOMER** | |
| Column Name | Description |
| CUSTOMER\_LVL\_ID1 | Hierarchy level ID 1 |
| CUSTOMER\_LVL\_ID2 | Hierarchy level ID 2 |
| CUSTOMER\_LVL\_ID3 | Hierarchy level ID 3 |
| CUSTOMER\_LVL\_ID4 | Hierarchy level ID 4 |
| CUSTOMER\_LVL\_ID5 | Hierarchy level ID 5 |
| CUSTOMER\_ID | Customer ID |

## DISTR\_CHANNEL

|  |  |
| --- | --- |
| **DDS.IN\_DISTR\_CHANNEL** | |
| Column Name | Description |
| DISTR\_CHANNEL\_LVL\_ID1 | Hierarchy level ID 1 |
| DISTR\_CHANNEL\_ID | Distribution CHANNEL ID |

## SALES (POS)

POS Sales information regarding the past till last known day of the history .

|  |  |
| --- | --- |
| **DDS.IN\_SALES** | |
| Column Name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location ID |
| **PERIOD\_DT** | Date of sales (calendar day) |
| **SALES\_QTY** | Total sales in units per day (w/o returns) |

## STOCK

Inventory history data containing the following fields is used as an input:

|  |  |
| --- | --- |
| **DDS.IN\_STOCK** | |
| Column Name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location ID |
| **PERIOD\_START\_DT** | Date of sales (calendar day) |
| **STOCK\_QTY** | Stock qty |

## SELL\_IN

Initial CPG sales history data

|  |  |
| --- | --- |
| **DDS.IN\_SELL\_IN** | |
| Column Name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location from ID |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **LOCATION\_TO\_ID** | Location to Id |
| **PERIOD\_DT** | Date of SELL-IN (calendar day) |
| **ORDER\_QTY** | Total ORDER in units per day (w/o returns) |
| **ORDER\_AMOUNT** | Total ORDER revenue (with VAT) per day (w/o returns) |
| **SHIPMENTS\_QTY** | Total SHIPMENT in units per day (w/o returns) |
| **SHIPMENTS\_AMOUNT** | Total SHIPMENT revenue (with VAT) per day (w/o returns) |
| **INVOICE\_QTY** | Total INVOICE in units per day (w/o returns) |
| **INVOICE\_AMOUNT** | Total INVOICE revenue (with VAT) per day (w/o returns) |
| **RETURNS\_QTY** | Total returns in units per day |
| **RETURNS\_AMOUNT** | Total returns amount (with VAT) per day |

## SELL\_OUT

Secondary retail sales history data

|  |  |
| --- | --- |
| **DDS.IN\_SELL\_OUT** | |
| Column Name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location from ID |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **LOCATION\_TO\_ID** | Location to Id |
| **PERIOD\_DT** | Date of SELL-OUT (calendar day) |
| **ORDER\_QTY** | Total ORDER in units per day (w/o returns) |
| **ORDER\_AMOUNT** | Total ORDER revenue (with VAT) per day (w/o returns) |
| **SHIPMENTS\_QTY** | Total SHIPMENT in units per day (w/o returns) |
| **SHIPMENTS\_AMOUNT** | Total SHIPMENT revenue (with VAT) per day (w/o returns) |
| **INVOICE\_QTY** | Total INVOICE in units per day (w/o returns) |
| **INVOICE\_AMOUNT** | Total INVOICE revenue (with VAT) per day (w/o returns) |
| **RETURNS\_QTY** | Total returns in units per day |
| **RETURNS\_AMOUNT** | Total returns amount (with VAT) per day |
| **PROMO\_FLG** | Promo flag |
| **PROMO\_ID** | Promo ID |
| **COST** | Unit cost |

## ASSORT\_MATRIX

Assortment matrix Sales information regarding the past till last known day of the history.

|  |  |
| --- | --- |
| **DDS.IN\_ASSORT\_MATRIX** | |
| Column Name | Description |
| **LOCATION\_ID** | Location from ID |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **START\_DT** | Assort Start date (e.g. sales for Retailer/ shipment for CPG /shelf life for Fashion) |
| **END\_DT** | Assort End date (e.g. sales for Retailer/ shipment for CPG /shelf life for Fashion) |
| **STATUS** | Status (Active, No longer merchandised etc.) |

## LOCATION\_LIFE

Information regarding lifecycle of each location. This table can be empty

|  |  |
| --- | --- |
| **DDS.IN\_LOCATION\_LIFE** | |
| Column Name | Description |
| **LOCATION\_ID** | Location ID |
| **CUSTOMER\_LVL\_ID** | Customer hierarchy element ID |
| **PRODUCT\_LVL\_ID** | Product hierarchy element ID |
| **DISTR\_CHANNEL\_LVL\_ID** | Distribution CHANNEL hierarchy element ID |
| **PERIOD\_START\_DT** | Period start date |
| **PERIOD\_END\_DT** | Period end date |
| **LOCATION\_SUCCESSOR\_ID** | Successor Location ID |
| **RELATION\_SHARE** | Number |
| **PERIOD\_TYPE** | Closure reason: reconstruction, re-branding etc. |

## PRODUCT\_LIFE

Promo history data containing the following fields is used as an input.

|  |  |
| --- | --- |
| **DDS.IN\_PRODUCT\_LIFE** | |
| Column Name | Description |
| **PRODUCT\_ID** | Product ID |
| **LOCATION\_LVL\_ID** | Location hierarchy element ID |
| **CUSTOMER\_LVL\_ID** | Customer hierarchy element ID |
| **DISTR\_CHANNEL\_LVL\_ID** | Distribution CHANNEL hierarchy element ID |
| **PERIOD\_START\_DT** | Period start date |
| **PERIOD\_END\_DT** | Period end date |
| **PRODUCT\_SUCCESSOR\_ID** | Successor Product ID |
| **RELATION\_SHARE** | Number |
| **PERIOD\_TYPE** | Active, blocked, end-of-life |

## CUSTOMER\_LIFE

|  |  |
| --- | --- |
| **DDS.IN\_CUSTOMER\_LIFE** | |
| Column Name | Description |
| **CUSTOMER\_ID** | Customer ID |
| **LOCATION\_LVL\_ID** | Location hierarchy element ID |
| **PRODUCT\_LVL\_ID** | Product hierarchy element ID |
| **DISTR\_CHANNEL\_LVL\_ID** | Distribution CHANNEL hierarchy element ID |
| **PERIOD\_START\_DT** | Period start date |
| **PERIOD\_END\_DT** | Period end date |
| **CUSTOMER\_SUCCESSOR\_ID** | Successor Customer ID |
| **RELATION\_SHARE** | Number |
| **PRODUCT\_LVL\_ID** | Number |
| **PERIOD\_TYPE** | active, blocked, end-of-life |

## Forecast Flag

Table is prepared at previous runs

|  |  |  |
| --- | --- | --- |
| **FORECAST\_FLAG** | | |
| Column Name | Description | Key FLG |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) | Yes |
| **LOCATION\_ID** | Location ID | Yes |
| **CUSTOMER\_ID** | Customer ID | Yes |
| **DISTR\_CHANNEL\_ID** | Distribution Channel ID | Yes |
| **PERIOD\_START\_DT** | Period start date | Yes |
| **PERIOD\_END\_DT** | Period end date | No |
| **STATUS** | One of the following status: maturity, new, end-of-life (only active-like periods) | No |

## CONFIGURATION PARAMETERS

### CONFIG\_PARAMETERS

The following config parameters are used within demand restoration step.

|  |  |
| --- | --- |
| **CONFIG.CONFIG\_PARAMETERS** | |
| Column Name | Description |
| **IB\_HIST\_END\_DT** | Last known date (i.e. sales and stock information are known) |

### INITIAL\_GLOBAL parameters

All parameters are listed in initial\_global file.

|  |  |
| --- | --- |
| **INITIAL\_GLOBAL parameters init** | |
| Column Name | Description |
| **IB\_MAX\_DT** | Maximal date which is used for data preparation (e.g. 01/01/2100) |
| **IB\_UPDATE\_HISTORY\_DEPTH** | Number of days of historical information that should be considered within this step running, i.e. only dates since **IB\_HIST\_END\_DT - IB\_UPDATE\_HISTORY\_DEPTH** should be used within step |

## Other Dependencies

Forecast Flag should be updated each time when data uploaded to the system, but commonly values in it are changed only for 20% of pairs product | location. Optimal realization of Forecast Flag calculation algorithm highly depends on data uploading format (update/insert or delete/replace, etc.).

All comparisons within algorithm regarding string values must be case-insensitive (i.e. ‘CASE\_INSENSITIVE’ = ‘case\_insensitive’).

# Algorithm Definition

The forecast flag definition algorithm is described in detail in the sections below.

It includes the next steps:

1. Stock and Sales or Sell-in dates calculation
2. Assortment matrix dates calculation
3. Life-cycle information merging
4. Forecast flag and life-cycle status defining.

Output result of the algorithm is a table, which described in the section «2.4 Output data» above.

## Technical utility for time-intervals union

First, special utility macro must be developed for union time-intervals within fixed group of objects (products, locations, customers, distribution channels and any element from any combination for those hierarchies). It will be used in each step below.

Utility parameters are shown in the table below:

|  |  |  |
| --- | --- | --- |
| Parameter | Feasible values or examples | Description |
| mpTimeGranularity | day, week or month | Time granularity. Each interval will be led up and considered in terms of chosen granularity |
| mpDistanceTolerance | >= 0, integer | Using this parameter time-interval right bound (mpEndDt) is being extended. It should be in terms of chosen mpTimeGranularity. |
| mpStartDt | start\_dt | Column name which contains left interval bound |
| mpEndDt | end\_dt | Column name which contains right interval bound |
| mpGroupBy | product\_id location\_id | List of the columns separated with blanks, which are keys of the table |

Algorithm:

1. Sort input table &mpInTable by &mpGroupBy and &mpStartDt.
2. Redefine bounds (&mpStartDt and &mpEndDt) for each time-interval in accordance with required time-granularity (&mpTimeGranularity) and tolerant distance (&mpDistanceTolerance):

* Left bound := lead &mpStartDt up to the beginning of the day/week/month based on &mpTimeGranularity parameter
* Right bound := lead &mpEndDt up to the beginning of the day/week/month based on &mpTimeGranularity parameter + number of periods from &mpDistanceTolerance parameter

1. Sorted intervals within each by-group (which is defined using parameter mpGroupBy) consider consequently, calculate and output new intervals responding to the logic below:
2. Initial step: Assign left bound and right bound as values in the first row of by-group
3. If there are only one interval, then output it and go-to the next by-group element
4. If there are more than one interval than consequently consider the next interval, then check, whether left bound of current interval belongs to previous interval:

* If yes, then check whether current right bound is greater or not than previous right bound.
  + If yes, then re-write right bound with value from current bound and go-to the next interval and repeat step c.
  + If no, then go-to the next interval without changes and repeat step c.
* If no, then previous interval should be output, and go-to step a. to maintain remaining intervals within this by-group.

1. Final combined intervals should be output using table name from &mpOutTable parameter and include bounds those intervals for each &mpGroupBy element.

## Utility to unfold aggregated data to lower level of organizational hierarchy

|  |  |  |
| --- | --- | --- |
| Parameter | Feasible values or examples | Description |
| mpInTable | IN\_EVENTS | Input Table that must be unfold, this table contains all 4 dimensions in it named product\_lvl\_id, location\_lvl\_id, customer\_lvl\_id, distr\_channel\_lvl\_id accordingly. |
| mpInProduct | IN\_PRODUCT | Product Hierarchy |
| mpInLocation | IN\_LOCATION | Location Hierarchy |
| mpInCustomer | IN\_CUSTOMER | Customer Hierarchy |
| mpInDistrChannel | IN\_DISTR\_CHANNEL | Distr Channel Hierarchy |
| mpInQuadruple | IN\_ASSORT\_MATRIX | Table with quadruples product\_id location\_id customer\_id distr\_channel\_id |

Algorithm:

1. If mpInQuadruple is not empty (is not None) then unfold mpInTable using only those quadruples that are present in it

SELECT PRODUCT\_ID, LOCATION\_ID, CUSTOMER\_ID, DISTR\_CHANNEL\_ID

\*\*.

FROM mpInTable

LEFT JOIN mpInProduct on product\_lvl\_id

LEFT JOIN mpInLocation on location\_lvl\_id

LEFT JOIN mpInCustomer on customer\_lvl\_id

LEFT JOIN mpInDistrChannel on distr\_channel\_lvl\_id

INNER JOIN mpInQuadruple on (product\_id, location\_id, customer\_id, distr\_channel\_id)

1. Else mpInQuadruple is empty (is None) then unfold original table based on

SELECT PRODUCT\_ID, LOCATION\_ID, CUSTOMER\_ID, DISTR\_CHANNEL\_ID

mpInTable.\*\* /\*everything apart from product\_lvl\_id, location\_lvl\_id, customer\_lvl\_id, distr\_channel\_lvl\_id \*/

FROM mpInTable

LEFT JOIN mpInProduct on product\_lvl\_id

LEFT JOIN mpInLocation on location\_lvl\_id

LEFT JOIN mpInCustomer on customer\_lvl\_id

LEFT JOIN mpInDistrChannel on distr\_channel\_lvl\_id

## Incremental load preparation

**Inputs:** SALES, STOCK, SELL\_IN, SELL\_OUT, I1=ASSORT\_MATRIX, LOCATION\_LIFE, PRODUCT\_LIFE, CUSTOMER\_LIFE

**Transformation algorithm:**

1. Check whether previous version of FORECAST\_FLAG table is available (is stored at previous step)
   1. If not then parameter **IB\_UPDATE\_HISTORY\_DEPTH[[1]](#footnote-2) = 0 for all steps below.**
2. Select list of quadruples that have DELETED\_FLG = 1 in ASSORT\_MATRIX, LOCATION\_LIFE, PRODUCT\_LIFE, CUSTOMER\_LIFE
   1. Prepare list of quadruples:
      1. SELECT product\_id, location\_id, customer\_id, distr\_channel\_id

FROM SALES

* 1. SELECT DISTINCT PRODUCT\_ID, LOCATION\_ID, CUSTOMER\_ID, DISTR\_CHANNEL\_ID

FROM ASSORT\_MATRIX WHERE DELETE\_FLG = 1

* 1. SELECT DISTINCT PRODUCT\_ID, LOCATION\_ID, CUSTOMER\_ID, DISTR\_CHANNEL\_ID

FROM LOCATION\_LIFE WHERE DELETE\_FLG = 1

Then unfold results from step 2. to the lowest level - product | location | customer | distr\_channel combination using

* 1. Union…

Name the list as QUADRUPLES\_DELETE

ASSUMPTION: there is no rows with delete\_flg = 1 in STOCK, SALES, SELL\_IN, SELL\_OUT tables earlier than **IB\_HIST\_END\_DT - IB\_UPDATE\_HISTORY\_DEPTH** (thus all changes in these tables can be considered with regular procedure).

1. Select only those triples/quintuples from a input table that relate to QUADRUPLES\_DELETE or date later then **IB\_HIST\_END\_DT - IB\_UPDATE\_HISTORY\_DEPTH** moment
   1. e.g. create table SALES\_UPDATE\_FF as

SELECT \*

FROM SALES

LEFT JOIN QUDRUPLES\_DELETE on PRODUCT\_ID, LOCATION\_ID

WHERE (**IB\_UPDATE\_HISTORY\_DEPTH<=0**

**OR** period\_dt **> IB\_HIST\_END\_DT - IB\_UPDATE\_HISTORY\_DEPTH**

**OR (**QUADRUPLES\_DELETE.PRODUCT\_ID is not missing**)**

**/\*Quadruples delete are added in order to delete them from output table\*/**

* 1. e.g. create table ASSORT\_MATRIX\_UPDATE\_FF as

SELECT \*

FROM ASSORT\_MATRIX

LEFT JOIN QUDRUPLES\_DELETE on PRODUCT\_ID, LOCATION\_ID, CUSTOMER\_ID, DISTR\_CHANNEL\_ID

WHERE (**IB\_UPDATE\_HISTORY\_DEPTH<=0**

**OR** period\_start\_dt **> IB\_HIST\_END\_DT - IB\_UPDATE\_HISTORY\_DEPTH OR** period\_end\_dt **> IB\_HIST\_END\_DT - IB\_UPDATE\_HISTORY\_DEPTH**

**OR** QUADRUPLES\_DELETE.PRODUCT\_ID is not missing**)**

All steps below use only updated tables (SALES\_UPDATE\_FF, STOCK\_UPDATE\_FF, etc).

## Add required fields into SALES and STOCK tables

**Inputs:** SALES = SALES\_UPDATE\_FF, STOCK=STOCK\_UPDATE\_FF, I1=ASSORT\_MATRIX\_UPDATE\_FF

**Transformation algorithm:**

1. Select LOCATION\_ID, PRODUCT\_ID and PERIOD\_DT from SALES and STOCK tables and union both results.

Assumption: STOCK table must not contain pairs LOCATION\_ID - PRODUCT\_ID, that are not present in the SALES table.

Assumption: no additional filtering (like WHERE SALES\_QTY>0) for SALES and STOCK tables is needed.

Next steps are needed if SALES and STOCK tables are needed and there are not CUSTOMER\_ID and DISTR\_CHANNEL\_ID columns in SALES and STOCK tables.

1. 1. left join ASSORT\_MATRIX on PRODUCT\_ID and LOCATION\_ID and PERIOD\_DT BETWEEN (I1.START\_DT and I1.END\_DT)

And add CUSTOMER and DISTR\_CHANNEL columns.

Note: It is assumed that true relations among locations and customers (distr\_channels) are provided be the client in ASSORT\_MATRIX. But if the awkward case where there are several customers/distr\_channels for the same LOCATION|PRODUCT takes place then all those relations product/location/customer (distr\_channel) will be present in output table.

1. Fill missing values in CUSTOMER and DIST\_CHANNEL columns, within each pair PRODUCT|LOCATION:
   1. Fill missing values with previous non-missing value
   2. If there are some missing values fill missing with next non-missing value
   3. If there is still missing values fill it with the minimal CUSTOMER\_ID (DIST\_CHANNEL\_ID).

ASSUMPTION: There is at least one element in CUSTOMER and DISTR\_CHANNEL dictionaries.

ASSUMPTION: This step assumes that there is only fake (dummy) elements in CUSTOMER and DIST\_CHANNEL dimensions and alignment among dimensions is not fulfilled on previous steps. But if there is several CUSTOMER (DISTR\_CHANNEL) in input data then all only a kind of random relation will be present in output table.

Output: As a result of this step, a table of the following structure is constructed, *T1.*

|  |  |
| --- | --- |
| Column name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location ID |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **PERIOD\_DT** | Date of sales (calendar day) |

## Fact dates calculation

**Inputs:** T1, SELL\_IN = SELL\_IN\_UPDATE\_FF, SELL\_OUT = SELL\_OUT\_UPDATE\_FF

**Transformation algorithm:**

1. UNION T1, SELL\_IN, SELL\_OUT with fields PRODUCT\_ID, LOCATION\_ID, CUSTOMER\_ID, DISTR\_CHANNEL\_ID, PERIOD\_DT

And select DISTINCT rows in it.

Assumption: no additional filtering (like WHERE SELL\_IN\_QTY>0) for SELL\_IN and SELL\_OUT tables is needed.

1. Call utility 4.0 for each product | location | customer | distr\_channel combination from 1. with the next parameters:

|  |  |
| --- | --- |
| Parameter | Value |
| mpTimeGranularity | Day |
| mpDistanceTolerance | 365 |
| mpStartDt | period\_dt |
| mpEndDt | period\_dt |
| mpGroupBy | product\_id location\_id customer\_id distr\_channel\_id |

Output: As a result of this step, a table of the following structure is constructed, *T2.*

|  |  |
| --- | --- |
| Column name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location ID |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **PERIOD\_START\_DT** | Start date of a period (calendar day) |
| **PERIOD\_END\_DT** | End date of a period (calendar day) |

## Assortment Matrix Calculation

**Inputs:** ASSORTMENT\_MATRIX = ASSORTMENT\_MATRIX\_UPDATE\_FF

**Transformation algorithm:**

1. Fill missing values in ASSORTMENT\_MATRIX.END\_DT
   1. Order by START\_DT in ascending order within each group **product\_id, location\_id, customer\_id, distr\_channel\_id**
   2. Fill missing in END\_DT with a START\_DT from the next row within **product\_id, location\_id, customer\_id, distr\_channel\_id or with IB\_MAX\_DT**:

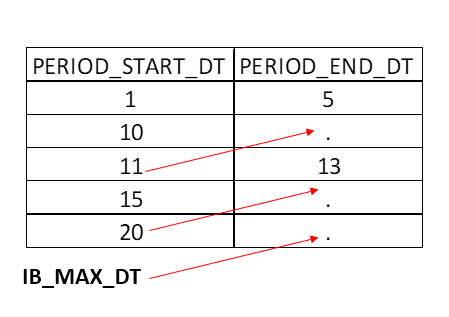
END\_DT = COALESCE(END\_DT,

NEXT(START\_DT), IB\_MAX\_DT).

See picture Figure 1below for clarification.

Figure 1: Fill missing values in PERIOD\_END\_DT field

1. Select distinct combinations – product\_id | location\_id | customer\_id | distr\_channel\_id - using filter Status = ‘Active’ with extracting START\_DT and END\_DT dates to the temporary table FF\_ASSORT\_DATES.



Output: As a result of this step, a table of the following structure is constructed, *FF\_ASSORT\_DATES.*

|  |  |
| --- | --- |
| Column name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location ID |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **START\_DT** | Start date of a period (calendar day) |
| **END\_DT** | End date of a period (calendar day) |

## Life-cycle information merging

**Inputs:** IN\_<name>\_LIFE\_UPDATE\_FF, where <name> in { CUSTOMER , LOCATION , PRODUCT }

**Transformation algorithm:**

1. Transform lifecycles tables from the successor – predecessor interval format to the single interval format.
2. Extract distinct from each table <name>\_LIFE key fields, PERIOD\_START\_DT and PERIOD\_END\_DT using filter:
   * 1. <name>\_SUCCESSOR\_ID is missing.
3. Extract distinct from each table <name>\_LIFE key fields, PERIOD\_START\_DT and PERIOD\_END\_DT using filter:
   * 1. <name>\_ID is NOT equal <name>\_SUCCESSOR\_ID

But instead of <name>\_ID use <name>\_SUCCESSOR\_ID. In other words, <name>\_SUCCESSOR\_ID will be part of the key instead of <name>\_ID.

1. Extract distinct from each table <name>\_LIFE key fields, PERIOD\_START\_DT and PERIOD\_END\_DT using filter:
   * 1. <name>\_ID is equal <name>\_SUCCESSOR\_ID
2. Append all 3 extracted tables to the WORK.FF\_COMBINED\_<name>\_LIFE independently.
3. Call utility macro 3 times for each combined <name>\_LIFE tables with the next parameters:

|  |  |
| --- | --- |
| Parameter | Value |
| mpTimeGranularity | Day |
| mpDistanceTolerance | 1 |
| mpInTable | WORK.FF\_COMBINED\_<name>\_LIFE |
| mpStartDt | period\_start\_dt |
| mpEndDt | period\_end\_dt |
| mpGroupBy | customer\_id location\_lvl\_id product\_lvl\_id distr\_channel\_lvl\_id  or  location\_id customer\_lvl\_id product\_lvl\_id distr\_channel\_lvl\_id  or  product\_id location\_lvl\_id customer\_lvl\_id distr\_channel\_lvl\_id |
| mpOutTable | WORK.FF\_PROCESSED\_<name>\_LIFE |

1. Unfold results from step 2. to the lowest level - product | location | customer | distr\_channel combination and append them to unified table WORK.FF\_LIFE\_DATES.

**Output**: As a result of this step, a table of the following structure is constructed

|  |  |
| --- | --- |
| Column name | Description |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) |
| **LOCATION\_ID** | Location ID |
| **CUSTOMER\_ID** | Customer ID |
| **DISTR\_CHANNEL\_ID** | Distribution CHANNEL ID |
| **PERIOD\_START\_DT** | Start date of a period (calendar day) |
| **PERIOD\_END\_DT** | End date of a period (calendar day) |

## Forecast Flag Calculation

**Inputs:** Temporary tables as results from the steps 4.2-4.4: FF\_FACT\_DATES (T2), FF\_ASSORT\_DATES, FF\_LIFE\_DATES, Forecast\_Flag, QUADRUPLES\_DELETE

**Transformation algorithm:**

1. Append input tables to unified table WORK.FF\_DATES.
2. Call utility macro with the next parameters for 1:

|  |  |
| --- | --- |
| Parameter | Value |
| mpTimeGranularity | Day |
| mpDistanceTolerance | 1 |
| mpInTable | WORK.FF\_DATES |
| mpStartDt | start\_dt |
| mpEndDt | end\_dt |
| mpGroupBy | product\_id location\_id customer\_id distr\_channel\_id |
| mpOutTable | WORK.FF\_DELTA\_OUTPUT |

1. Create new field STATUS with constant value ‘active’.
2. Update the result to the FORECAST\_FLAGtable.
   1. Get rid off all rows from rows from FORECAST\_FLAG that contain QUADRUPLES\_DELETE
   2. Add rows from 3 to a.

SELECT PRODUCT\_ID, LOCATION\_ID, DISTR\_CHANNEL\_ID, CUSTOMER\_ID, period\_start\_dt,

FROM 3 UNION a

* 1. Call utility macro with the next parameters for b to union intersective intervals.

|  |  |
| --- | --- |
| Parameter | Value |
| mpTimeGranularity | Day |
| mpDistanceTolerance | 1 |
| mpInTable | Results of step 4.b |
| mpStartDt | period\_start\_dt |
| mpEndDt | period\_end\_dt |
| mpGroupBy | product\_id location\_id customer\_id distr\_channel\_id |
| mpOutTable | WORK.FF\_OUTPUT |

**Output**: As a result of this step, a table of the following structure is constructed, (see 2.4)

## Filter out excessive and obsolete elements of dictionaries

**Inputs:** Forecast\_Flag, PRODUCT, LOCATION, CUSTOMER, DISTR\_CHANNEL.

**Transformation algorithm:**

1. Leave only those elements of the lowest level that are present in Forecast\_Flag table
   1. PRODUCT join FORECAST\_FLAG on product\_id
   2. LOCATION join FORECAST\_FLAG on location\_id
   3. CUSTOMER join FORECAST\_FLAG on customer\_id
   4. DISTR\_CHANNEL join FORECAST\_FLAG on distr\_channel\_id

**Output**: As a result of this step, updated dictionaries should be stored: PRODUCT\_FILTERED, LOCATION\_FILTERED, DISTR\_CHANNEL\_FILLTERED, CUSTOMER\_FILTERED (see 2.4)

## Output Data

Final output table should look like as follows (keys are PRODUCT\_ID, LOCATION\_ID):

|  |  |  |
| --- | --- | --- |
| **FORECAST\_FLAG** | | |
| Column Name | Description | Key FLG |
| **PRODUCT\_ID** | Product ID (the lowest level of the product hierarchy) | Yes |
| **LOCATION\_ID** | Location ID | Yes |
| **CUSTOMER\_ID** | Customer ID | Yes |
| **DISTR\_CHANNEL\_ID** | Distribution Channel ID | Yes |
| **PERIOD\_START\_DT** | Period start date | Yes |
| **PERIOD\_END\_DT** | Period end date | No |
| **STATUS** | One of the following status: maturity, new, end-of-life (only active-like periods) | No |

# Ideas for next steps

|  |  |
| --- | --- |
| № | OQ or NS |
| 1 | Future possibility to improve Forecast Flag Algorithm: enhance forecast flag status using IN\_PRODUCT\_LIFE and IN\_LOCATION\_LIFE information |
| 2 | Do we need to provide FF on aggregated level (e.g. SKU+1 level)?  If yes, what is that level? The most detailed level of the further data flow (including final output table).  Default the lowest level of each dimension |
| 3 | It looks like we need to have special global parameter for data-type – CPG or RETAIL (see 3.1) |
| 4 | Current design considers combining method of time-intervals from life-cycle tables as union.  But there are some points which points out necessity of applying intersection method – but in one life-cycle table can be more than one interval for one by-group.  In particular, retailer’s source data can exist like independent data for locations and for products, which should be intersected to get information of product – location level.  Maybe it’s better to add parameter mpMode = ‘intersect’ or ‘union’ to the next release of the utility macro. |
| 5 | Provide diversity in lifecycle statuses into final table. Different phases in product life cycle like introduction, maturity, end-of-life are called ‘active’ (there is no difference in current version). |
| 6 | During this algorithm we’ve lost ratio coefficients from the lifecycle tables. Do we need to save them some way too? |
| 7 | Evolve SALES and STOCK extention algorithm: if there is several CUSTOMERS (DISTR\_CHANNELS) in input data and no relation between location and CUSOMTER (DISTR\_CHANNEL) is provided in ASSORT\_MATRIX. |

1. It doesn’t mean that global parameter must by rewritten, it means that parameter value must be different for particular run. [↑](#footnote-ref-2)