ESA Sen4Stat

Sentinels for Agricultural Statistics

D7.0 – DDF - ATBD

"Algorithm Theoretical Basis Document: *In situ* Data Preparation"

Issue/Rev : 1.0

Date Issued : 27-08-21

Milestone: 2

Submitted to European Space Agency

Consortium Partners

Participant Organisation Name	Acronym	City, Country
Université catholique de Louvain	UCLouvain	Louvain-la-Neuve, Belgium
CS GROUP - Romania	CSG RO	Craiova, Romania
Systèmes d'Information à Référence Spatiale - Filiale CLS	SIRS-CLS	Villeneuve d'Ascq, France
Universidad Polytecnica de Madrid	UPM	Madrid, Spain

Contact

Université catholique de Louvain – Earth and Life Institute

Place de l'Université, 1 – B-1348 Louvain-la-Neuve – Belgium

Email : Sophie.Bontemps@uclouvain.be

Internet : https://uclouvain.be/en/research-institutes/eli/elie

Disclaimer

Document sheet

Authors and Distribution

Authors	Nicolas DEFFENSE, Cosmin CARA, Laurentiu NICOLA, Cosmin UDROIU, Fabrice DAZIN, Sadri HAOUET, Sophie BONTEMPS
Distribution	ESA - Benjamin KOETZ, Zoltan SZANTOI

Document Status

Issue/Rev.	Date	Reason
0.1	18/05/2021	Initial version of the deliverable
0.2	09/06/2021	Internal review of the deliverable
0.3	30/06/2021	2nd internal review of the deliverable
0.4	13/07/2021	Internal review with CLS
1.0	27/08/2021	1st version related to the Acceptance Review

Detailed record sheet

From version 1.0 to 1.x

Comment	Section	Problem description	Change

Table of contents

List of figures

List of tables

List of codes

Acronyms

Acronym	Definition
AD	Applicable Document
AOI	Area Of Interest
ATBD	Algorithm Theoretical Basis Document
DDF	Design Definition File
EO	Earth Observation
ESA	European Space Agency
ID	Identifier
LUT	Look-Up Table
L1, L2, L3, L4	Level 1, Level 2, Level 3, Level 4
L8	Landsat 8
NDVI	Normalized Difference Vegetation Index
NSO	National Statistical Office
S1, S2	Sentinel-1, Sentinel-2
Sen4Stat	Sentinels for Agricultural Statistics
UTM	Universal Transverse Mercator

1 Logical model and processor overview

In situ data collected from National Statistical Offices (NSO) must be prepared and quality controlled before being used for the production of Earth Observation (EO) products. This is the objective of this processor, which assumes that the *in situ* data are provided by the NSOs as polygons: it qualifies each polygon with a set of indicators or flags related to its geometry, area, quality and stratum.

The overview of the "*In situ* Data Preparation" processor is summarized in Figure 1-1 and detailed in the following sections of this document.

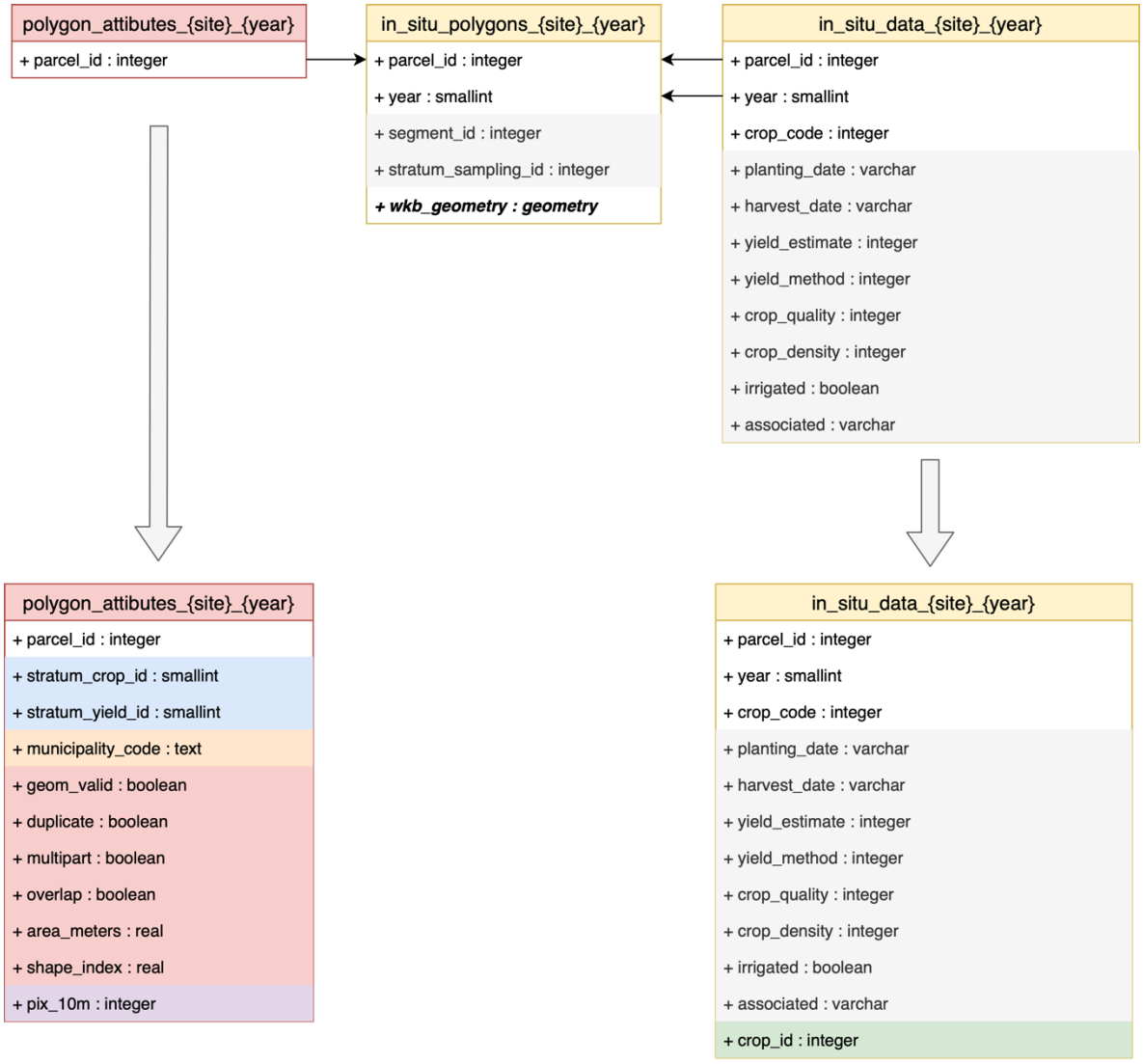

Figure 1-1. Workflow of the NSO declaration dataset preparation.

2 Input data

2.1 NSO *in situ* statistical dataset

In situ statistical dataset used in the Sen4Stat system are data collected by the NSO. The dataset must be provided as a table with the fields listed in Table 2-1. If there are associated crops on the same parcel (i.e. several crops at the same time within the same parcel), several records in the *in sit*u statistical dataset may have the same "*parcel_id"*.

Table 2-1. NSO *in situ* statistical dataset. Optional fields are in grey

Field name	Role	Default value [format]
parcel_id	ID used to link the in-situ information to a polygon	[integer]
crop_code	Code of the crop (equivalent to SUBnum in Figure 2-1)	[integer]
year	Year of the survey	[integer]
planting_date	Start date of the crop period	[date]
harvest_date	End date of the crop period	[date]
yield_estimate	Yield estimation in kg/ha	[float]
yield_method	Method used to estimate yield (see Table 2-2)	[integer]
crop_quality	Crop quality (see Table 2-4)	[integer]
crop_density	Number of crop plants per unit area in nb/m²	[integer]
irrigated	Presence (=1) or absence (=0) of irrigation	[integer, binary]
associated	Single crop (=0) or associated crop[1] (=1)	[integer, binary]

By default, there are 4 main classes of yield estimation method: eye estimation, data provided by the farmer, crop cutting and other methods (Table 2-2). The user is allowed to add other classes to better specify the different yield estimation methods it uses. By default, only class 1 is used to calibrate the yield model (Table 2-3). Here again, the user is free to modify this parameterization and to consider other class(es) for the calibration of the yield model.

[1] Associated crops means more than one crop inside a plot. It could be different crops growing at the same time with superposition of the growth phases (mixed crops) or it could be successive crops. It could be also several turns of the same crops (i.e. rice).

Table 2-2. Yield estimation methods

Code	Description
1	Eye estimation
2	Data provided by the farmer
3	Crop-cutting
4	Other methods

Table 2-3. Yield methods selected for calibration

Parameters	Role	Default value [format]
yield_calibration	List of yield estimation methods used to calibrate the yield model	1 [integer]

Crop quality is an optional information that the user can add to possibly filter out certain parcels that could reduce the classification accuracy. By default, there are 2 classes: normal growing condition and abnormal growing condition (Table 2-4)). The user can add as many classes as desired, as for instances: crops with scattered or border trees, protected crop, crop already harvested, etc. By default, only parcels belonging to class 1 will be used for classification.

Table 2-4. Crop quality

Code	Description
1	Normal growing conditions
2	Abnormal growing conditions (abandoned crop, young plantation that does not produce, climatic damage, erosion damage, disease, parasite damage)

2.2 NSO *in situ* geographical dataset

In situ geometries with a unique "*parcel_id*" for each entity (polygon for plot boundary or point for plot location) is provided by the NSO as a shapefile, with the fields listed in Table 2-5.

In a later version, it will be necessary to foresee that the NSO data can be points. In this case, there will be a buffer step to get polygons.

Table 2-5. NSO *in situ* shapefile. Optional fields are in grey

Field name	Role	Default value [format]
parcel_id	Unique ID for each in-situ polygon	[integer]
year	Year of the survey	[integer]
segment_id	Sampling segment where the parcel is located	[integer]
stratum_sampling_id	Sampling stratum to which the parcel/segment belongs	[integer]

2.3 Strata boundaries

The user can import a stratification of his Area Of Interest (AOI) into the system, in order to improve the quality of crop mapping and/or or yield-related products. This is particularly necessary if the AOI is large and contains various agro-climatic zones. By default, the AOI will be considered as belonging to a single stratum and the "stratum_crop_id" and "stratum_yield_id" will be set to 1 for all parcels (Table 2-6).

Table 2-6. Strata boundaries

Field name	Role	Default value [format]
stratum_crop_id	Stratum ID for crop mapping products	1 [integer]
stratum_yield_id	Stratum ID for yield-related products	1 [integer]

2.4 Administrative limits

The ultimate goal of the Sen4Stat system is to use EO data to support the computation of agricultural statistics at national level but also at finer scales. To calculate these estimations over the various administrative entities, it is necessary to know the boundaries of each of the different units. From the largest to the smallest unit, these limits are :

1. Country – level 0;
2. Region – level 1;
3. Province – level 2;
4. Municipality – level 3.

For each of these administrative levels, the user must upload 4 shapefiles with the boundaries. If the user does not have this information, he must download the official boundaries himself through the website https://gadm.org/data.html. Each shapefile must contain specific fields to link administrative units of different levels to each other (Table 2-7).

Table 2-7. Administrative limits

Shapefile name	Field name	Example
country	country_name	*Ecuador*
	country_code	*EC*
region	region_name	*Chimborazo*
	region_code	*EC05*
	country_code	*EC*
province	province_name	*Alausi*
	province_code	*EC0501*

	region_code	*EC05*
municipality	municipality_name	*Guasuntos*
	municipality_code	*EC051003*
	province_code	*EC0501*

2.5 Crop Look-Up table

The Sen4Stat's Crop Look-Up table (LUT) is a **global** and **hierarchical** LUT based on "*JECAM Guidelines for cropland and crop type definition and field data collection*"[2].

This Crop LUT is **global** meaning that it includes all the classes identified in all Sen4Stat's pilot countries. It is unique. This Crop LUT has been defined in a generic way with the aim to be valid in other countries and to easily evolve if some crop types are missing.

The Crop LUT is **hierarchical** because it includes different levels of aggregation, from the most general to the most precise classes, there are:

1. Land Cover ("lc" in Figure 2-1)
2. Group ("grp" in Figure 2-1)
3. Class ("class" in Figure 2-1)
4. Sub-class ("sub" in Figure 2-1)

Figure 2-1 shows an example of the hierarchical classes for wheat.

land cover	land cover name	group	gourp name	class	class name	sub	sub name
1	Annual cropland	11	Cereals	111	Wheat	1111	Winter wheat
1	Annual cropland	11	Cereals	111	Wheat	1112	Spring wheat
1	Annual cropland	11	Cereals	111	Wheat	1113	Hard wheat
1	Annual cropland	11	Cereals	111	Wheat	1114	Soft wheat
1	Annual cropland	11	Cereals	111	Wheat	1115	Triticale

Figure 2-1. Example of Sen4Stat Crop LUT for the wheat

Other columns are present in this Crop LUT to facilitate the grouping of certain classes needed for certain products. For example, the column "**cm**" is useful to generate the binary crop mask (Figure 2-2). It is a column with '1' for all sub-classes that are cropland and '2' for all sub-classes that are non-cropland.

[2] http://jecam.org/wp-content/uploads/2018/10/JECAM_Guidelines_for_Field_Data_Collection_v1_0.pdf

land cover	land cover name	group	gourp name	class	class name	sub	sub name	cm	cm name
1	Annual cropland	11	Cereals	111	Wheat	1111	Winter wheat	1	Cropland
1	Annual cropland	11	Cereals	115	Barley	1151	Barley two-row	1	Cropland
2	Perennial crops	22	Vineyards	221	Vineyards	2211	Vineyards	1	Cropland
6	Forest	61	Conifers	611	Conifers	6111	Conifers	2	Non-cropland
8	Build-up surface	81	Urban	811	Urban	8111	Urban	2	Non-cropland

Figure 2-2. Complementary information included in the Sen4Stat Crop LUT

2.6 NDVI

NDVI time series are generated by the "Spectral Indices & Biophysical Indicators" processor and are used to control the quality of *in situ* data (Section 3.4).

3 Detailed workflow

3.1 Standardization

3.1.1 Conversion in PostgreSQL

The original *in situ* statistical dataset collected by NSO is stored into a PostgreSQL database (see Appendix 1). The PostGIS extension should be installed to facilitate and accelerate the further spatial processing. At least 3 types of data must be provided by the user:

- the main table with the *in situ* data (yellow table);
- the polygon boundaries from the field survey (red table);
- the delineations of the 4 main administrative levels: municipality, province, region and country (orange tables).

The schema provided in Figure 3-1 describes the Sen4Stat database structure and relations between tables.

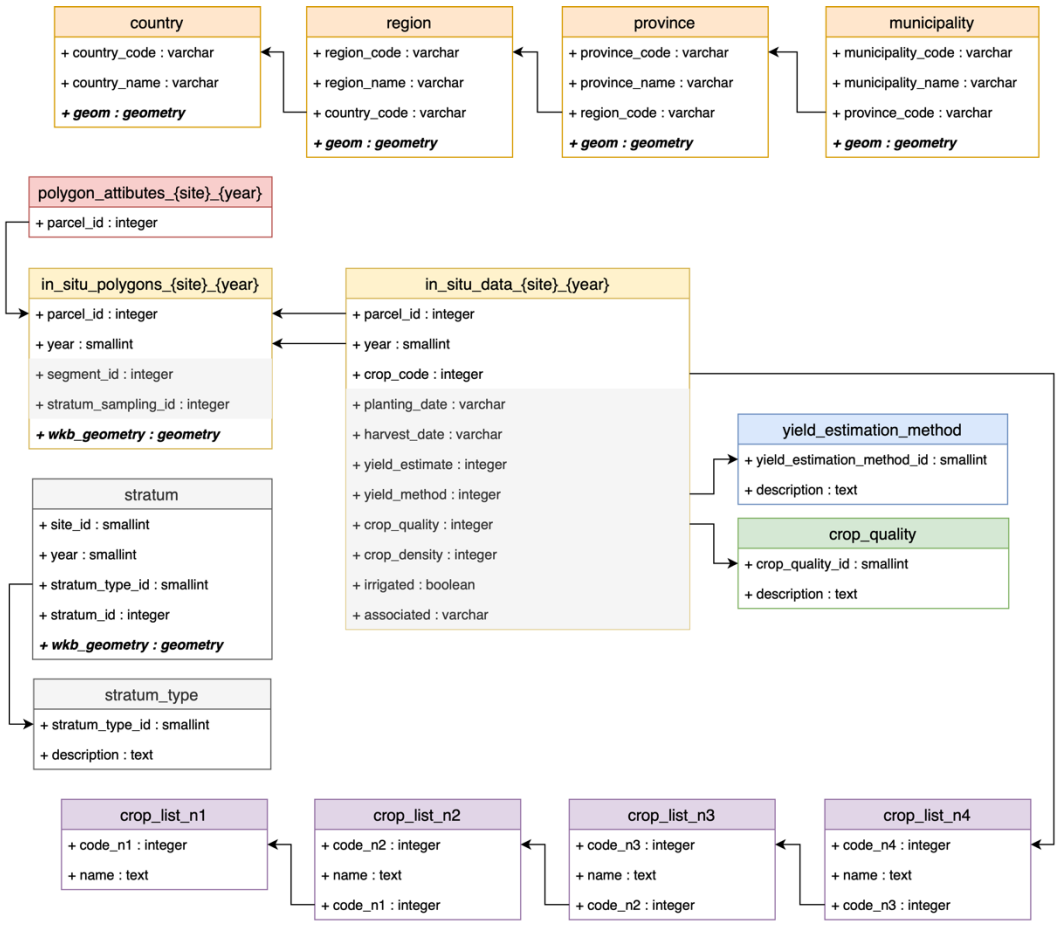

Figure 3-1. Original NSO datasets are converted into a PostgreSQL database with different tables. Optional fields are in grey.

3.1.2 Add unique crop ID

This step adds one new sequential ID named '*crop_id'* for each crop (= each row) of the "in_situ_data" table. The '*parcel_id'* used to link each crop information to a parcel geometry is not a unique ID because a parcel can have several crops simultaneously or successively.

3.1.3 Add administrative levels

The 4 shapefiles containing the delineations of the 4 main administrative levels provided by the user are first integrated in 4 different tables named: municipality, region, province and country. Each table is linked to the upper level by a common field. For example, the municipality table contains the field *'province_code'* to identify the province to which each municipality belongs. Only the *'municipality_code'* is added in the "polygon_attributes" table, as the others can be computed using a couple of "join" operations.

```
CREATE TEMPORARY TABLE site_municipalities (
      municipality_code text NOT NULL,
      geom geometry NOT NULL
);
```

Code 3-1. Create temporary “site_municipalities” table

```
WITH polygons_srid AS (
          SELECT Find_SRID('public','in_situ_polygons_staging','wkb_geometry')
          AS srid
     ),
     polygons_extent AS (
         SELECT ST_Extent(wkb_geometry) AS extent
         FROM in_situ_polygons_staging
     ),
     polygons_extent_4326 AS (
         SELECT ST_Transform(ST_SetSRID(polygons_extent.extent,
                                        polygons_srid.srid),
                             4326) AS geog
         FROM polygons_extent,
              polygons_srid
     )
```

Code 3-2. Compute the overall extent of the polygons and reproject it to EPSG:4326

```
INSERT INTO site_municipalities
SELECT municipality.municipality_code,
       ST_Transform(municipality.geom, srid) AS geom
FROM municipality,
     polygons_extent_4326,
     polygons_srid
WHERE ST_Intersects(municipality.geom, polygons_extent_4326.geog);
```

Code 3-3. Find the municipalities that intersect the polygons, reproject them back to the CRS of the polygons, then insert them into the temporary table

```
CREATE INDEX ON site_municipalities USING gist (geom);
```

Code 3-4. Create a spatial index on the geometries from the temporary table

```
INSERT INTO polygon_attributes (
    municipality_code,
    ...
)
SELECT(
      SELECT municipality_code
      FROM site_municipalities
      WHERE ST_Intersects(geom, polygons.wkb_geometry)
      LIMIT 1
    ),
    ...
FROM in_situ_polygons_staging polygons;
```

Code 3-5. Match the polygons against these filtered geometries

3.1.4 Add strata — optional

If the user decides to import a stratification in the Sen4Stat system, two tables must be created: stratum and stratum_type (Code 3-6).

```
CREATE TABLE stratum_type(
      stratum_type_id smallint NOT NULL PRIMARY KEY,
      description text NOT NULL
);

INSERT INTO stratum_type
VALUES
      (1, 'Crop classification'),
      (2, 'Yield estimation');

CREATE TABLE stratum(
      stratum_type_id smallint NOT NULL,
      stratum_id int NOT NULL,
      year int NOT NULL,
      wkb_geometry geometry NOT NULL,
      PRIMARY KEY stratum_id
);
```

Code 3-6. Create stratum tables

This step adds a stratum ID to each polygon in the "polygon_attributes" table according to the stratum in which it is located (Code 3-7). A difference is done between strata used for crop mapping products and strata used for yield-related products.

```
INSERT INTO polygon_attributes (
    stratum_crop_id,
    stratum_yield_id,
    ...
)
SELECT
    COALESCE(
        (SELECT stratum_id
         FROM stratum
         WHERE (site_id, year, stratum_type_id) = (SITE_ID, YEAR, 1)
           AND ST_Intersects({}.wkb_geometry, stratum.wkb_geometry)
         LIMIT 1)
        , 0),
    COALESCE(
        (SELECT stratum_id
         FROM stratum
         WHERE (site_id, year, stratum_type_id) = (SITE_ID, YEAR, 2)
           AND ST_Intersects({}.wkb_geometry, stratum.wkb_geometry)
         LIMIT 1)
        , 0),
    ...
FROM in_situ_polygons;
```

Code 3-7. Populate "polygon_attributes" table with strata information

3.1.5 Standardization summary

Figure 3-2 shows the 2 main tables "polygon_attributes" and "in_situ_data" that contain the new fields created in this standardization step:

- *'crop_id'*: a unique ID for each crop;
- *'stratum_crop_id', 'stratum_yield_id'*: stratum to which each polygon belongs;
- *'municipality_code'* : municipality to which each parcel belongs.

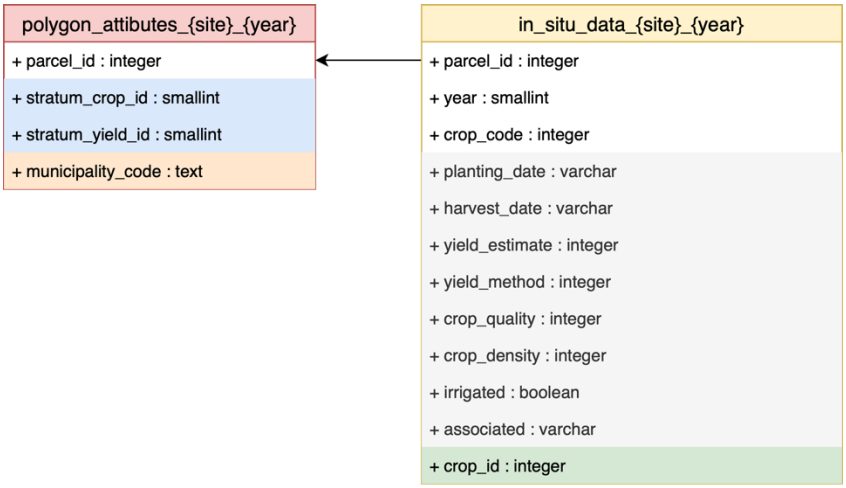

Figure 3-2. “polygon_attributes” and “in_situ_data” tables after the standardization step. Optional fields are in grey

3.2 Geometry flags

The full definition of each geometry flag is provided here below, as well as the spatial PostGIS queries to calculate them. Each geometry flag is computed and stored in the “polygon_attributes” table.

3.2.1 Check polygon geometry — geom_valid

This flag identifies the polygons:

- for which no geometry exists;
- that have a geometry which is not valid, like self-intersecting polygons.

This flag is binary:

- Value ‘1’ if the polygon has a geometry and the geometry is valid;
- Value ‘0’ if the polygon has no geometry or the geometry is not valid.

The “polygon_data” table is updated with a PostGIS query (Code 3-8).

```
UPDATE polygon_attributes
SET “geom_valid” = ST_IsValid(wkb_geometry)
;
```

Code 3-8. Check polygon geometry

3.2.2 Check duplicate polygons — duplicate

This flag identifies polygons that have the same geometry as another one.

This flag is binary:

- Value '1' if the polygon has the same geometry as another polygon;
- Value '0' if the polygon has a unique geometry.

The "polygon_data" table is updated with the PostGIS query provided in Code 3-9.

```
UPDATE polygon_attributes
SET "duplicate" = "parcel_id" IN (
    SELECT "parcel_id"
    FROM (
         SELECT "parcel_id",
                COUNT(*) OVER(PARTITION BY wkb_geometry) AS count
         FROM polygon_data
    ) t WHERE count > 1
);
```

Code 3-9. Check duplicate polygons

3.2.3 Check multipart polygons — multipart

This flag identifies polygons that have multipart geometry.

This flag is binary:

- Value '1' if the polygon is multipart.
- Value '0' if the polygon is not multipart.

The "polygon_data" table is updated with the PostGIS query from Code 3-10.

```
UPDATE polygon_attributes
SET "multipart" = 1
WHERE ST_NumGeometries(t.wkb_geometry) > 1)
;
```

Code 3-10. Check multipart polygons

3.2.4 Compute polygon area — area_meters

This flag gives the polygon area computed based on the geometry stored in "polygon_data" table projected in the national projection. It is calculated in m^2 and is stored as an float value (Code 3-11).

```
UPDATE polygon_attributes
SET "area_meters" = ST_Area(wkb_geometry)
;
```

Code 3-11. Compute polygon area

3.2.5 Check polygon overlap — overlap

This flag identifies the polygons which overlap neighbouring polygons with more than 10 % of their area.

This flag is binary:

- Value '1' if the polygon overlaps with other polygons with more than 10% of its area;
- Value '0' if there is not the case.

The "polygon_data" table is updated with the PostGIS query in Code 3-12.

```
UPDATE polygon_data
SET "overlap" = 1
WHERE "geom_valid"
AND EXISTS (
    SELECT 1
    FROM {} polygon_data
    WHERE t."parcel_id" != {}."parcel_id"
    AND t."geom_valid"
    AND ST_Intersects(t.wkb_geometry, polygon_data.wkb_geometry)
    HAVING
SUM(ST_Area(ST_Intersection(t.wkb_geometry,{}.wkb_geometry)))/nullif({}."area_
meters",0)
> 0.1
);
```

Code 3-12. Check polygon overlap

3.2.6 Compute shape index — shape_index

This flag characterizes the compactness of the polygons. It is calculated with the following equation:

$$shape\_index = \frac{Perimeter}{2 * \sqrt{\pi * Area_{UTM}}}$$

This flag is stored as a float value. The Shape Index value equals to 1 when the patch is circular and increases without limit as patch becomes more irregular.

The "polygon_data" table is updated with the PostGIS query from Code 3-13.

```
UPDATE polygon_data
SET "shape_index" = ST_Perimeter(wkb_geometry)/(2 * sqrt(pi() *
nullif(ST_Area(wkb_geometry), 0))
);
```

Code 3-13. Compute shape index

3.2.7 Geometry flags summary

Table 3-1 summarizes the new fields added to "polygon_attributes" table, while Figure 3-3 illustrates the evolution of this table.

Table 3-1. New fields added to the "polygon_attributes" table during the geometry flags step

Field name	Role	Default value [format]
geom_valid	Identify polygons for which no geometry exists in the *in situ* statistical dataset or with a not valid geometry	[boolean]
duplicate	Identify polygons that have the exact same geometry as another	[boolean]
multipart	Identify multi parts polygons	[boolean]
area_meters	Polygon area in national projection (m^2)	[real]
overlap	Identify polygons which overlaps with neighbouring polygons	[boolean]
shape_index	Shape index of the polygon	[real]

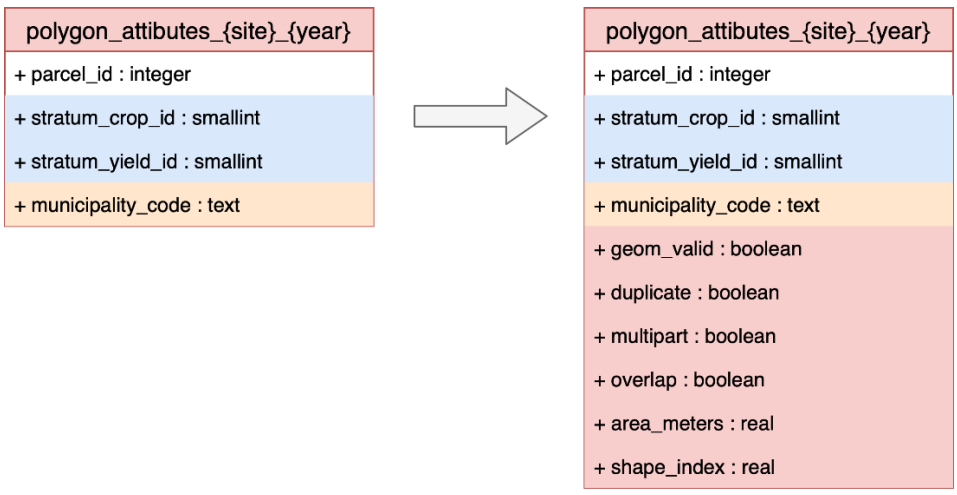

Figure 3-3. "polygon_attributes" table after the geometry flags step

3.3 Polygons rasterization and pixels number flags

During this third part of the *in situ* data pre-processing, the polygons are rasterized to make the computing of their area in pixels at the target resolution (10m) easier. The numbers of 10-meters pixels - both Sentinel-2 (S2) and Sentinel-1 (S1) data are at 10-m resolution - are added as one additional quality flag to the "polygon_attributes" table.

The rasterization process also generates polygons buffer layers.

The approach adopted in Sen4Stat to count the numbers of S2 10-m pixels that are used by polygon for any feature extraction responds to two objectives:

- to keep - as much as possible - only the pixels that belong entirely to the polygon (i.e. that do not partly cover its surroundings);
- to maximise - as much as possible - the number of pixels that are used by polygon to extract the spectral values of the S2 and S1 data.

The developed solution consists in (Figure 3-4):

- first, applying an inner buffer to each polygon with a distance of 10m;
- second, selecting only the S2 pixels that have their centroid inside this buffer.

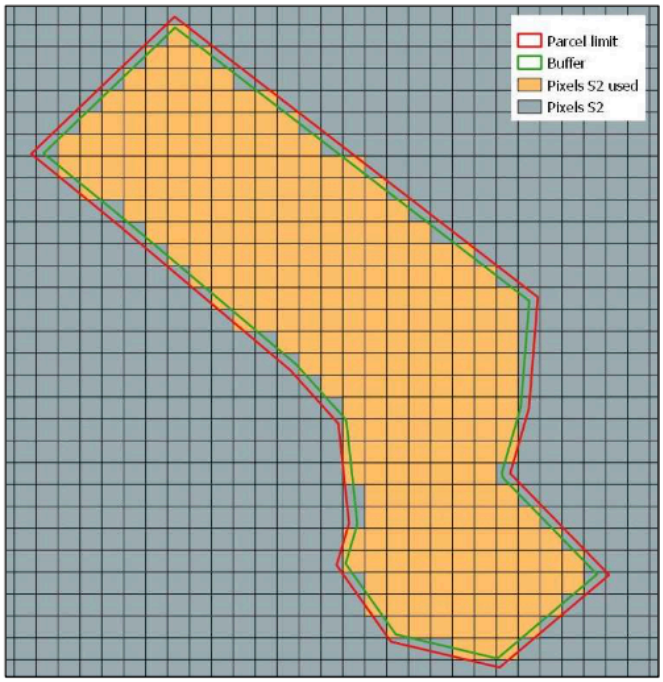

Figure 3-4. S2 pixels selection approach for the spectral values extraction by parcel

The rasterization process is done in 5 steps:

1. Reprojection: the "polygon_data" table is re-projected in one or several UTM zone(s) that cross(es) the country;
2. Inner buffer: one inner buffer layer is created for each of the re-projected "polygon_data" at 10 meters;
3. Clip by tile: the buffer layers are clipped by tile, using the UTM zone projection corresponding to the tile;
4. Rasterization: the products of the last steps are rasterized using the '*parcel_id'* as the value;
5. Pixels number flags — pix_10m: the number of S2 pixels is calculated for each polygon.

In the implementation, the first three steps are merged into a single PostGIS query and are therefore described in a single sub-section (3.3.1).

3.3.1 Reprojection, inner buffers and clip by tile

- **Rationale for reprojection**

Sen4Stat "polygon_data" table is reprojected into the same projection as the Sentinel data (which is WGS 84 / UTM zone {x}). The "polygon_data" table is thus re-projected in the one or several UTM zone projections that cross(es) the country. The list of these projections is derived from the name of the S2 tiles.

- **Rationale for inner buffers**

A 10-m inner buffer is applied to each polygon of the reprojected "polygon_data" tables.

- **Rationale for clip by tile**

The reprojected buffer layers are clipped in multiple parts corresponding to each S2 tile extent (bounding box). Depending on the UTM zone in which the S2 tile is located, the reprojected buffer layer with the corresponding projection is used. The S2 tiles are identified using their reference name made of two digits and three letters (e.g. 31UFS).

- **Implementation**

These three steps are summarized in one single PostGIS query which is passed in "gdal_rasterize" command (Code 3-14). This query is applied for each S2 tile of the country.

```
WITH transformed AS (
   SELECT   epsg_code,   ST_Transform(shape_tiles_s2.geom,   Find_SRID('public',
'in_situ_polygons', 'wkb_geometry')) as geom
    FROM shape_tiles_s2
    WHERE tile_id = {}
)
SELECT parcel_id, ST_Buffer(ST_Transform(wkb_geometry, epsg_code), buffer_size)
FROM in_situ_polygons, transformed
WHERE ST_Intersects(wkb_geometry, transformed.geom);
```

Code 3-14. Reprojection, inner buffers and clip by tile

3.3.2 Rasterization

The reprojected buffer polygons (clipped by S2 tile), given by the previous PostGIS query (Code 3-14) are rasterized using the '*parcel_id'* field as key value in order to obtain a one-band raster with each pixel having the value of the '*parcel_id'* of the polygons. The tile extents are used as reference frame. The *'parcel_id'* value of the polygon is assigned to a pixel only if its centroid is located inside of the corresponding reprojected and clipped by S2 tile buffer layer.

The Rasterization parameters are:

- value = *parcel_id*;
- resolution = 10m;
- PostGIS query = query described in Code 3-14.
- tile extent used as reference frame (*Xmin Ymin Xmax Ymax*);
- rule = centroid of the pixel inside the inner buffer.

The rasterization is done using **gdal_rasterize**, as follows (Code 3-15).

```
gdal_rasterize
   -q
   -a parcel_id
   -a_srs {epsg}
   -te {XMIN YMIN XMAX YMAX}
   -tr 10 10
   -sql {PostGIS_query}
   -ot Int32
   -co COMPRESS=DEFLATE
   -co PREDICTOR=2
   {country}_{year}_InSitu_S4S_buffer_{buffer_size}m_{tile}.tif
```

Code 3-15. Rasterization

3.3.3 Pixels number flags — pix_10m

The number of S2 pixels that will be used to extract spectral values from the S2 and S1 data is calculated by polygon. This information is added to the "polygon_data" table (Code 3-16).

```
for tile in s2_tile_list:

    compute the histogram of the rasterization output

    sort the histogram by 'parcel_id'

merge histograms, summing the pixel counts across tiles

update the PostGIS "polygon_attributes" table with the resulting counts
```

Code 3-16. Add S2 pixels to polygons

3.3.4 Pixels number flag summary

The input and output of this step are detailed in Table 3-2.

Table 3-2. Input and output data of pixels number flags step

Input data	Description	Default value [format]
in_situ_polygons	Standardized Sen4Stat table with polygon geographical data	[PostGIS table]
polygon_attributes	Standardized Sen4Stat table with polygon attributes	[PostGIS table]
shape_tiles_s2	Table with the bounding box of each S2 tile	[PostGIS table]
buffer_size	Inner buffer size in meters	10 [float]
Output data	**Description**	**Default value [format]**
in_situ_polygons_{site}_{year} _{tile}_{epsg}_buf_{buffer_size}m	Reprojected Sen4Stat *in situ* data with buffer –{buffer_size}m	[shapefile]
in_situ_polygons_{site}_{year} _{tile}_{buffer_size}m	Raster of all the used S2 pixels by polygon (value as parcel_id); resolution = 10 m; {tile} = S2 tile name	[GeoTIFF]

The following fields have been added to "polygon_attributes" table (Table 3-3), as shown also in Figure 3-5.

Table 3-3. New fields added to the "polygon_data" table during the pixel number flag step

Field name	Role	Default value [format]
pix_10m	Indicates the number of used S2 pixels in the polygon	[integer]

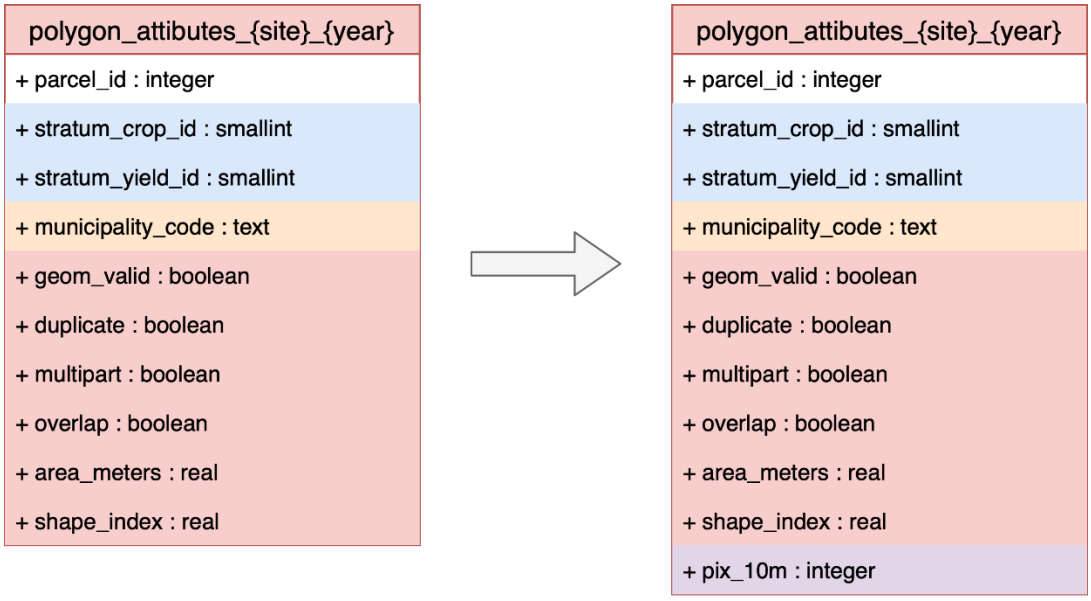

Figure 3-5. "polygon_attributes" table after the pixel number flag step.

3.4 Quality control flags

Coming in the next version

/!\ This step will become more complex. Others input data are expected /!

Jolan Wolter: NDVI time series

Sophie Lox: OpenStreetMap roads intersection with polygon

CLS: polygons without LC

The quality of each polygon must be controlled to avoid that the data of lower quality impact the performances of further processing.

The input and output of this step are detailed in Table 3-4. The new field added to "polygon_data" table by this step is given in Table 3-5.

Table 3-4. Input and output data of quality control flags step

Input data	Description	Default value [format]
polygon_data	Standardized Sen4Stat table with polygon geographical data	[PostGIS table]
NDVI time series	NDVI at each acquisition dates	[GeoTIFF]
Output data	**Description**	**Default value [format]**
polygon_data	Standardized Sen4Stat table with polygon geographical data	[PostGIS table]

Table 3-5. New fields added to the NSO declaration dataset during the quality control flags step

Field name	Role	Default value [format]
quality_control	Indicates if the polygon is homogeneous and of good quality	[integer]

4 Output

4.1 PostGIS tables

The main output of the "*In situ* Data Preparation" processor is a set of three standardized Sen4Stat PostGIS tables (Figure 4-1):

1. in_situ_data:
    a. crop code
    b. year
    c. parcel ID (not necessarily unique) to link to "in_situ_polygon" and "polygon_data"
    d. unique crop ID
    e. optional statistical data (yield, crop quality, irrigation, …)

2. in_situ_polygon:
    a. unique parcel ID
    b. year
    c. optional survey information (segment, stratum used for sampling design)
    d. parcel geometry

3. polygon_attributes:
    a. unique parcel ID
    b. optional stratum information (mapping product and/or yield-related product)
    c. municipality code
    d. geometry flags
    e. number of 10m pixels

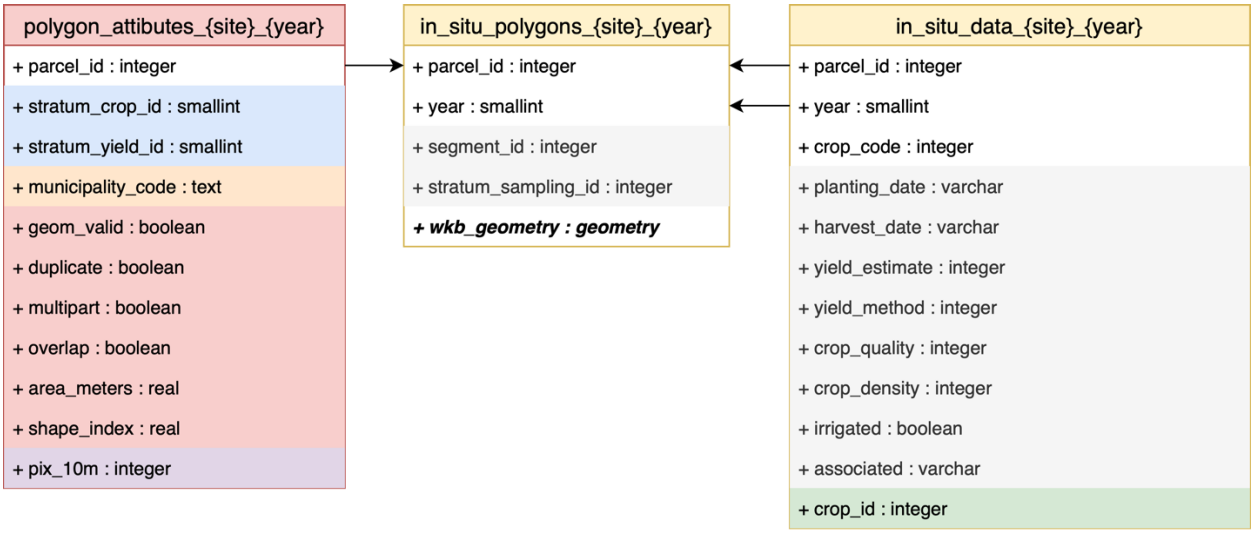

Figure 4-1. Final tables of the "*In situ* Data Preparation" processor

4.2 Polygons buffer shapefiles

The second outputs are the polygons reprojected buffer layers (10m inner buffers - see section3.3.1). These buffers are reprojected in the WGS 84 / UTM zone {x} projections that correspond to all the UTM zones that cross the country. It only contains one field, the *'parcel_id'* of the parcel (Table 4-1).

The parcels buffer layers:

- are shapefiles;
- are produced by UTM zones that cross the country;
- are projected in WGS 84 / UTM zone {x}.

Table 4-1. Output 2 — Polygons buffer layers

Output data	Description	Default value [format]
in_situ_polygons_{site}_{year} _{tile}_{epsg}_buf_{buffer_size}m	Reprojected Sen4Stat *in situ* data with buffer – {buffer_size}m	[shapefile]

4.3 Polygons raster layers

The third outputs are the raster files produced from the polygons buffer layers by tile, with the *'parcel_id'* as value (see section 3.3.2). Only the pixels that have their centroid located in the buffer layer have been assigned the *'parcel_id'* value of the parcel (Table 4-2).

These layers:

- are GeoTIFF files;
- are produced by S2 tile;
- are projected in the WGS 84 / UTM zone {x} corresponding to the UTM zone of the S2 tile;
- value = *'parcel_id'* of the parcels.

Table 4-2. Output 3 — Polygons raster layers

Output data	Description	Default value [format]
in_situ_polygons_{site}_{year} _{tile}_{buffer_size}m	Raster of all the used S2 pixels by polygon (value as parcel_id); resolution = 10 m; {tile} = S2 tile name	[GeoTIFF]

5 Appendix 1. Conversion of the original dataset in PostgreSQL

Before processing, create two PostGis schemas named “schema_name” and “sen4stat”.

Import all input NSO shapefiles into “schema_name”.

```
All the scripts are generate using ECUADOR as an example.
--Insert Country
-- For administrative boundaries coming from https://gadm.org/data.html
-- “libelle” = “NAME_0”
-- “code” = first left 2 digits of field GID_0

--Insert Region
-- For administrative boundaries coming from https://gadm.org/data.html
-- “libelle” = “NAME_1”
-- “code” = first left 2 digits of field GID_1 + digit 5 and 6 from the left.
-- ex GID_1 = ECU.23_1 then code = EC23
-- the field gid_1_new is created in advance to calculate and store EC23
-- the field gid_0_new is created in advance to calculate and store EC

--Insert province
-- same principle as above

--Insert municipality
-- same principle as above

--                Intermediate stages

--       Creation of codes for the integration of administrative units

  -- Ecuador region
alter table schema_name.ecuador_region add column gid_1_new character varying;
update schema_name.ecuador_region
set gid_1_new =
```

```
case when length(split_part(split_part(gid_1, '.', 2), '_', 1)) < 2 then
 'EC0' || split_part(split_part(gid_1, '.', 2), '_', 1)
 else 'EC' || split_part(split_part(gid_1, '.', 2), '_', 1)
 end;

  --Province Ecuador
alter table schema_name.ecuador_province add column gid_1_new character varying;
alter table schema_name.ecuador_province add column gid_2_new character varying;

  --Region_code
update schema_name.ecuador_province
  set gid_1_new =
  case when length(split_part(split_part(gid_1, '.', 2), '_', 1)) < 2 then
  'EC0' || split_part(split_part(gid_1, '.', 2), '_', 1)
  else 'EC' || split_part(split_part(gid_1, '.', 2), '_', 1)
  end;

  --Province code
update schema_name.ecuador_province
set gid_2_new =
case when length(split_part(split_part(gid_2, '.', 3), '_', 1)) < 2 then
 gid_1_new || '0'|| split_part(split_part(gid_2, '.', 3), '_', 1)
 else gid_1_new || split_part(split_part(gid_2, '.', 3), '_', 1)
 end;

 --Municipality ecuador
alter table schema_name.ecuador_municipality add column gid_1_new character
varying;
alter table schema_name.ecuador_municipality add column gid_2_new character
varying;
alter table schema_name.ecuador_municipality add column gid_3_new character
varying;

  --Region_code
update schema_name.ecuador_municipality
  set gid_1_new =
```

```
  case when length(split_part(split_part(gid_1, '.', 2), '_', 1)) < 2 then
  'EC0' || split_part(split_part(gid_1, '.', 2), '_', 1)
  else 'EC' || split_part(split_part(gid_1, '.', 2), '_', 1)
  end;

  --Province code
update schema_name.ecuador_municipality
set gid_2_new =
case when length(split_part(split_part(gid_2, '.', 3), '_', 1)) < 2 then
gid_1_new || '0'|| split_part(split_part(gid_2, '.', 3), '_', 1)
else gid_1_new || split_part(split_part(gid_2, '.', 3), '_', 1)
end;

  --Municipality code
update schema_name.ecuador_municipality
set gid_3_new =
case when length(split_part(split_part(gid_3, '.', 4), '_', 1)) < 2 then
 gid_2_new || '00'|| split_part(split_part(gid_3, '.', 4), '_', 1)
  when length(split_part(split_part(gid_3, '.', 4), '_', 1)) < 3 then
   gid_2_new || '0'|| split_part(split_part(gid_3, '.', 4), '_', 1)
 else gid_2_new || split_part(split_part(gid_3, '.', 4), '_', 1)
 end;



-- Allocation of the municipality in the segment according to the highest
recovery rate
drop table if exists schema_name.segment_maxarea;
create table schema_name.segment_maxarea as
select s.gid, (select max(st_area(st_intersection(s1.geom, p1.geom)))
                              From sen4stat.segment s1
                              Join      sen4stat.municipality      p1      on
st_intersects(s1.geom, p1.geom)
                              Where s1.gid = s.gid )
From sen4stat.segment s;

drop table if exists schema_name.temp_segment_import;
```

```
create table schema_name.temp_segment_import as
select s.gid, p.code
From sen4stat.municipality p
Join sen4stat.segment s on st_intersects(s.geom, p.geom)
Join schema_name.segment_maxarea ss on s.gid = ss.gid
where st_area(st_intersection(s.geom, p.geom)) = ss.max;
--               END Intermediate stages
```

Code 5-1. Conversion of the original dataset in PostgreSQL

6 Appendix 2. Creation of a single crop LUT

```
SELECT crop_list_n1.code AS lc_num,
       crop_list_n1.libelle AS lc,
       crop_list_n2.code AS grp_num,
       crop_list_n2.libelle AS grp,
       crop_list_n3.code AS class_num,
       crop_list_n3.libelle AS class,
       crop_list.code AS sub_num,
       crop_list.libelle AS sub
INTO sen4stat.crop_lut
FROM sen4stat.crop_list_n1
INNER JOIN sen4stat.crop_list_n2 ON crop_list_n2.code_n1 = crop_list_n1.code
INNER JOIN sen4stat.crop_list_n3 ON crop_list_n3.code_n2 = crop_list_n2.code
INNER JOIN sen4stat.crop_list ON crop_list.code_n3 = crop_list_n3.code;
```

Code 6-1. Merging by-level crop LUT into a single crop LUT