Prétraitement - Normalisation

Amina, KACIMI Valentin, DEHAINAULT Minh-Hoang DANG

December 22, 2018

```
In [1]: -- connection: host='localhost' dbname='Photographies' user='postgres'
        password='postgres'
```

1 Pré-requis:

Pour pouvoir utiliser ce Notebook, il faut préparer votre environnement de travail avec certains outils ci-dessous.

1.1 pgAdmin:

La base `Photographies` doit être crée préalablement et les parametres de connection (premiere cellule) doivent etre adaptées.

1.2 PostgreSQL kernel:

La version experimentale doit être installé: `pip install git+https://github.com/mhoangvslev/postgres_kern`
Les extensions Jupyter sont optionels.

2 Importer les donées brutes dans le SGBD

On a la possibilité d'importer les données à partir d'un fichier CSV avec la requête `COPY ... FROM`
L'import des données va invoquer les règles définies sur la table cible.

```
COPY FROM will invoke any triggers and check constraints on the destination table.
However, it will not invoke rules.
```

Nous allons donc préparer deux tables vides, avec les attributs de types adéquats, puis les règles pour les données entrantes. La première table, nommée `CorrectionTemp` servira à la redirection des données vers la vraie table `Correction`, via une multitude de triggers.

```
In [2]: DROP EXTENSION IF EXISTS btree_gin CASCADE;
        --CREATE EXTENSION pg_trgm;
        CREATE EXTENSION btree_gin;

        /**
         * COMPOSITE TYPE: CoordLambert93
         * C'est un type qui contient un array de varchar
         * Le champ acceuillit les données préparées pour l'insertion
         */
        DROP TYPE IF EXISTS CoordLambert93 CASCADE;
        CREATE TYPE CoordLambert93 AS (
```

```
    CoordX numeric,
    CoordY numeric
);

DROP TABLE IF EXISTS CorrectionTemp;
CREATE TABLE CorrectionTemp(
    ReferenceCindoc varchar NULL ,
    Serie varchar NULL ,
    Article varchar NULL,
    Discriminant varchar NULL,
    Ville varchar NULL,
    Sujet varchar NULL,
    DescDet varchar NULL,
    Date varchar NULL,
    Notebp varchar NULL,
    FicNum varchar NULL,
    Idx_per varchar NULL,
    Idx_Ico varchar NULL,
    NbrCli varchar NULL,
    TailleCli varchar NULL,
    N_V varchar NULL,
    C_G varchar NULL,
    Remarques varchar NULL
);


DROP TABLE IF EXISTS Correction;
CREATE TABLE Correction(
    ReferenceCindoc varchar(6) NULL, --SEP |
    Serie varchar NULL ,
    Article int NULL, --AUTO INCREMENT, PRIMARY KEY
    Discriminant int NULL,
    Ville varchar NULL,
    Sujet varchar NULL, -- SEP ,
    DescDet varchar NULL,
    Date varchar NULL, -- remove first chunk, parse date from french
    NoteBP varchar NULL, -- SEP /
    FicNum varchar NULL, -- SEP /
    Idx_per varchar NULL,
    Idx_Ico varchar NULL, -- SEP /, regex CCCCDDD_[SERIE]_DDDDDD_D
    NbrCli varchar NULL, -- SEP /, |
    TailleCli varchar NULL, --SEP, par NbrCli
    N_V varchar(3) NULL, -- regex NEG|INV
    C_G varchar(3) NULL, -- regex 'GSC'|'CLR'
    Remarques varchar NULL, -- S
    Coordonnees CoordLambert93 NULL
);

DROP TABLE IF EXISTS VilleTemp;
CREATE TABLE VilleTemp (
    viReference varchar,
    viCodeInsee varchar,
    viCodePostal varchar,
    viVille varchar,
    viHabitants varchar,
    viDensite varchar,
    viLatitude varchar,
    viLongitude varchar,
    viLambertX numeric,
    viLambertY numeric
);

COPY VilleTemp (
    viReference,
    viCodeInsee,
    viCodePostal,
    viVille,
    viHabitants,
```

```
        viDensite,
        viLatitude,
        viLongitude,
        viLambertX,
        viLambertY

    ) FROM '/home/minhhoangdang/L3/S5/BD/TEA/pdfsrc/villes.csv' DELIMITER ';' CSV HEADER;

    /* ============================================
     * INDEX: Ville_Nom_Coords_Key
     * Optimiser: la requête sur la table ville comme on n'a besoin que de 2 attributs
     * 5 mins -> 28s
     * ============================================ */
    CREATE INDEX idx_Ville_Nom_Coords ON VilleTemp(viVille, viLambertX, viLambertY);
    CREATE INDEX idx_Ville_Patterns ON VilleTemp USING gin(viVille gin_trgm_ops);
```

```
NOTICE:  drop cascades to function traitement_coord(anyarray)
NOTICE:  table "correctiontemp" does not exist, skipping
NOTICE:  table "correction" does not exist, skipping
NOTICE:  table "villetemp" does not exist, skipping
```

3 Nettoyage de données

Comme nous l'avons discuté précédemment, nous allons écrire un `TRIGGER` pour chaque 'problème' ci-dessous. Afin d'obtenir le résultat le plus satisfaisant, nous avons besoins d'une librairie de fonctions qui permettent de traiter les données entrantes sous forme de `string`.

In [3]:
```
/**
 * COMPOSITE TYPE: insert_array
 * C'est un type qui contient un array de varchar
 * Le champ acceuillit les données préparées pour l'insertion
 */
DROP TYPE IF EXISTS insert_array CASCADE;
CREATE TYPE insert_array AS (
    field varchar[]
);


/**
 * FUNCTION: split_string(str, delim1, delim2)
 * Séparer un string par 2 delimiteurs en un ensemble de strings
 */
DROP FUNCTION IF EXISTS split_string;
CREATE OR REPLACE FUNCTION public.split_string(_str varchar, _delim1 varchar, _delim2
varchar)
RETURNS varchar[] AS $$
    DECLARE
        result varchar[] ;
    BEGIN
        result := ARRAY(
            SELECT unnest(string_to_array(a, _delim2))
            FROM   unnest(string_to_array(_str, _delim1)) a
        );
        --RAISE NOTICE 'split_string(): %', result;
        RETURN result;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: traitement_date(str)
 *  input format 'chunk: [month year | year ]
 */
DROP FUNCTION IF EXISTS traitement_date;
```

```
CREATE OR REPLACE FUNCTION traitement_date(_str varchar)
RETURNS varchar[] AS $$
    DECLARE
        res varchar[] := string_to_array(regexp_replace(_str, '.*:(.*)', '\1'), '/');
        temp varchar;
        i int;
    BEGIN
        FOR i IN 1..coalesce(array_length(res, 1), 0) LOOP
            -- Double jour: 29-30 Juin 2011
            IF res[i] ~* '[0-9]{1,2}-[0-9]{1,2}.*' THEN
                temp := res[i];
                res := array_append(
                    res, concat(
                        split_part(temp, '-', 1),
                        substring(temp from '\s*\w+\s*[0-9]{4}')
                    )::varchar
                );
                res := array_append(res, split_part(temp, '-', 2)::varchar);
                res := array_remove(res, res[i]);

            -- Double mois: 29 Juin-Juillet 2011
            ELSIF res[i] ~* '.*\w+-\w+.*' THEN
                temp := res[i];
                res := array_append(
                    res, concat(
                        split_part(temp, '-', 1),
                        substring(temp from '\s*[0-9]{4}')
                    )::varchar
                );
                res := array_append(res, split_part(temp, '-', 2)::varchar);
                res := array_remove(res, res[i]);
            END IF;
        END LOOP;

        /* FOR i IN 1..coalesce(array_length(res, 1), 0) LOOP
            res[i] := trim(BOTH FROM res[i]);
        END LOOP; */
        res := ARRAY(SELECT trim(BOTH FROM d) FROM unnest(res) d);

        RETURN res;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: traitement_n_v(str)
 *  négatif = NEG, inversible = INV, details += remarques
 */
DROP FUNCTION IF EXISTS traitement_n_v;
CREATE OR REPLACE FUNCTION traitement_n_v( strSet varchar[])
RETURNS varchar(3)[] AS $$
    DECLARE
        i int;
    BEGIN
        IF strSet is NULL THEN
            RETURN NULL;
        END IF;

        FOR i IN 1 .. array_length(strSet, 1) LOOP
            IF (strSet[i] LIKE '%négatif%') THEN
                strSet[i] := 'NEG';
            ELSE
                strSet[i] := 'INV';
            END IF;
        END LOOP;
        RETURN strSet;
    END;
$$ LANGUAGE plpgsql;
```

```
/**
 * FUNCTION: array_expand()
 *  négatif = NEG, inversible = INV, details += remarques
 */
DROP FUNCTION IF EXISTS array_expand;
CREATE OR REPLACE FUNCTION array_expand( arr anyarray, maxlength int, fill anyelement =
null )
RETURNS anyarray AS $$
    DECLARE
        i int;
        length int := coalesce(array_length(arr, 1), 0);
    BEGIN
        --RAISE NOTICE 'MaxLength: %, Length: %', maxlength, length;
        IF (maxlength > length) THEN
            FOR i IN 1 .. (maxlength - length) LOOP
                arr := array_append(arr, fill);
            END LOOP;
        END IF;
        RETURN arr;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: getMaxLength(VARIADIC arr insert_array[] )
 *  Pour chaque ligne d'insertion, on prend l'attribut avec le plus grand nombre de
colonne
 */
DROP FUNCTION IF EXISTS getMaxLength;
CREATE OR REPLACE FUNCTION getMaxLength(VARIADIC arr insert_array[] )
RETURNS int AS $$
    DECLARE
        result int;
    BEGIN
        SELECT coalesce(max(array_length($1[i].field, 1)), 0)
        FROM generate_subscripts($1, 1) g(i)
        INTO result;
        --RAISE NOTICE 'maxLength: %', result;
        RETURN result;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: traitement_discriminant(arr anyarray)
 * Numéroter les photos d'un même article
 */
CREATE OR REPLACE FUNCTION traitement_discriminant(arr varchar[])
RETURNS int[] AS $$
    DECLARE
        i int;
        length int := coalesce(array_length(arr, 1), 0);
    BEGIN
        FOR i IN 1..length LOOP
            arr[i] = i;
        END LOOP;
        RETURN arr;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: pretty(strarr varchar[])
 * Rendre les attributs texts plus homogènes et présentables
 */
CREATE OR REPLACE FUNCTION pretty(strarr varchar[])
RETURNS varchar[] AS $$
    DECLARE
        i int;
        length int := coalesce(array_length(strarr, 1), 0);
    BEGIN
```

```
        FOR i IN 1..length LOOP
            strarr[i] = TRIM(BOTH ' ' FROM strarr[i]);
            strarr[i] = regexp_replace(strarr[i], '([a-z])([A-Z])', '\1-\2');
        END LOOP;
        RETURN strarr;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: traitement_c_g(str)
 *  coulieur = CLR, noir&blanc = GSC (greyscale)
 */
DROP FUNCTION IF EXISTS traitement_c_g;
CREATE OR REPLACE FUNCTION traitement_c_g( strSet varchar[] )
RETURNS varchar(3)[] AS $$
    DECLARE
        i int;
    BEGIN
        IF strSet is NULL THEN
            RETURN NULL;
        END IF;

        FOR i IN 1 .. array_length(strSet, 1) LOOP
            IF (strSet[i] ILIKE '%nb%') THEN
                strSet[i] := 'GSC';
            ELSE
                strSet[i] := 'CLR';
            END IF;
        END LOOP;
        RETURN strSet;
    END;
$$ LANGUAGE plpgsql;


/**
 * FUNCTION: traitement_c_g(str)
 *  coulieur = CLR, noir&blanc = GSC (greyscale)
 */
DROP FUNCTION IF EXISTS traitement_coord;
CREATE OR REPLACE FUNCTION traitement_coord( villeArr anyarray)
RETURNS CoordLambert93[] AS $$
    DECLARE
        res CoordLambert93 ARRAY;
        row CoordLambert93;
        i int;
        CoordX numeric;
        CoordY numeric;
    BEGIN
        FOR i IN 1..coalesce(array_length(villeArr, 1), 0) LOOP
            SELECT viLambertX, viLambertY INTO CoordX, CoordY
            FROM VilleTemp
            WHERE viVille ILIKE villeArr[i];

            row.CoordX := CoordX;
            row.CoordY := CoordY;

            res = array_append(res, row);
        END LOOP;
        RETURN res;
    END;
$$ LANGUAGE plpgsql;

/**
 * FUNCTION: traitement_oeuvre(str)
 */
DROP FUNCTION IF EXISTS traitement_oeuvre;
CREATE OR REPLACE FUNCTION traitement_oeuvre(str varchar)
RETURNS varchar[] AS $$
```

```
            BEGIN
                IF str IS NULL THEN RETURN NULL; END IF;
                RETURN split_string(
                    regexp_replace(
                        regexp_replace(str, '([A-Z])\s*,\s*([A-Z])', '\1 \2'),
                                ',\s*(\()', ' \1'
                    ), '|', '/'
                );
            END;
        $$ language plpgsql;

        /**
         * FUNCTION: traitement_cliche(str)
         */
        DROP FUNCTION IF EXISTS traitement_cliche;
        CREATE OR REPLACE FUNCTION traitement_cliche(str varchar)
        RETURNS varchar[] AS $$
            BEGIN
                RETURN string_to_array(
                    regexp_replace(str, '([0-9][0-9]*),([0-9])', '\1.\2', 'g'), ',');
            END;
        $$ language plpgsql;
```

```
NOTICE:  drop cascades to function getmaxlength(insert_array[])
NOTICE:  function getmaxlength() does not exist, skipping
NOTICE:  function traitement_coord() does not exist, skipping
```

3.1 Éliminer les incohérences dans les données

Nous avons remarqué une certaine incohérence dans les données: plusieures syntaxes pour exprimer une information (négatif/verre négatif), plusieurs types de tâches ou d'usure référencées, ... Ce traitement est chargé dans le trigger qui fait la séparation des lignes.

Pour supprimer les doublons, on procède de la manière suivante: On vérifie si deux lignes sont identiques, plus précisément si toutes les valeurs contenues dans une ligne sont identiques à celles d'une autre si c'est le cas on supprime.

3.2 Séparer les lignes combinées

Pour séparer les lignes combinées, nous avons crée un trigger qui détecte la présence des séparateurs (virgules, pipe, slash) selon chaque attribut du fichier CSV (donc la présence de multiples informations dans une même case). Avec les délimiteurs, on sépare les informations combinées dans un array correspondant à l'attribut concerné. Finalement, on passe les attributs dans la clause `VALUES` de `INSERT`.

3.3 Les Cantons associés aux communes

Après avoir récupéré un fichier csv d'une opendata contenant les coordonnées lambert 93 (lambertX,lambertY) de toutes les villes de france. - on mets à jour notre table pour ajouter deux colonnes LambertX et LambertY. - on crée une nouvelle table Villes qui contiendra les noms ainsi que les coordonnées Lambert de chaque ville. - on update notre table pour insérer les coordonnées.

```
In [4]: /**
         * TRIGGER: trigger_pretraitement
         * PROCEDURE: pretraitement
         */
```

```
CREATE OR REPLACE FUNCTION pretraitement()
RETURNS TRIGGER AS $$
    DECLARE
        ReferenceCindocVals insert_array;
        DiscriminantVals insert_array;
        VilleVals insert_array;
        SujetVals insert_array;
        DateVals insert_array;
        NoteBPVals insert_array;
        Idx_PerVals insert_array;
        FicNumVals insert_array;
        Idx_IcoVals insert_array;
        NbrCliVals insert_array;
        TailleCliVals insert_array;
        N_VVals insert_array;
        C_GVals insert_array;
        CoordVals CoordLambert93[];

        maxLength int;
        insertVals insert_array[];

    BEGIN
        ------------------------------------------------------------------------------
        -- Traitement des données
        ------------------------------------------------------------------------------
        ReferenceCindocVals.field := string_to_array(NEW.ReferenceCindoc, '|');
        DiscriminantVals.field := string_to_array(NEW.Discriminant, '|');
        VilleVals.field := pretty(string_to_array(NEW.Ville, ','));
        SujetVals.field := string_to_array(NEW.sujet, ',');
        DateVals.field := traitement_date(NEW.Date);
        NoteBPVals.field := split_string(NEW.NoteBP, '|', '/');
        FicNumVals.field := split_string(
            regexp_replace(NEW.FicNum, '(.*)\.(.*)', '\1'), '|', '/');
        Idx_PerVals.field := traitement_oeuvre(NEW.Idx_per);
        Idx_IcoVals.field := split_string(lower(NEW.Idx_ico), '|', '/');
        Idx_IcoVals.field := ARRAY(SELECT unnest(string_to_array(a, ',')) FROM
unnest(Idx_IcoVals.field) a);
        NbrCliVals.field := split_string(NEW.NbrCli, '|', '/');
        TailleCliVals.field := traitement_cliche(NEW.TailleCli);
        N_VVals.field := traitement_n_v(string_to_array(NEW.n_v, ','));
        C_GVals.field := traitement_c_g(string_to_array(NEW.c_g, ','));
        CoordVals := traitement_coord(VilleVals.field);

        maxLength := getMaxLength(
                ReferenceCindocVals,
                DiscriminantVals,
                VilleVals,
                SujetVals,
                DateVals,
                NoteBPVals,
                Idx_PerVals,
                FicNumVals,
                Idx_IcoVals,
                NbrCliVals,
                TailleCliVals,
                N_VVals,
                C_GVals
        );

        VilleVals.field := array_expand(VilleVals.field, maxLength, VilleVals.field[1]);

        ------------------------------------------------------------------------------
        -- Insérer dans la bonne table les données séparées
        ------------------------------------------------------------------------------
        INSERT INTO Correction (ReferenceCindoc, Serie, Article, Discriminant, Ville,
                Sujet, DescDet, Date, NoteBP, Idx_Per, FicNum, Idx_Ico, NbrCli,
                TailleCli, N_V, C_G, Remarques, Coordonnees)
            VALUES(
```

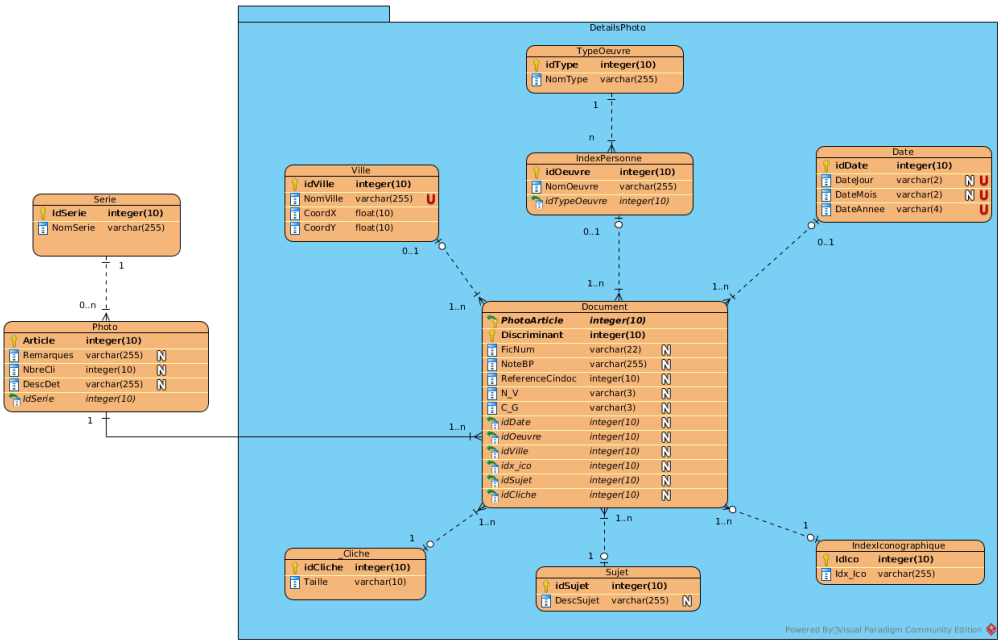
```
                unnest(
                    array_expand(ReferenceCindocVals.field, maxLength,
        ReferenceCindocVals.field[1])::int[]
                ),
                NEW.serie,
                cast(NEW.article as int),
                unnest(traitement_discriminant(
                    array_expand(DiscriminantVals.field, maxLength,
        DiscriminantVals.field[1]))
                ),
                unnest(VilleVals.field),
                unnest(pretty(array_expand(SujetVals.field, maxLength,
        SujetVals.field[1]))),
                NEW.DescDet,
                unnest(pretty(array_expand(DateVals.field, maxLength,
        DateVals.field[1]))),
                unnest(pretty(array_expand(NoteBPVals.field, maxLength,
        NoteBPVals.field[1]))),
                unnest(pretty(array_expand(Idx_PerVals.field, maxLength,
        Idx_PerVals.field[1]))),
                unnest(pretty(array_expand(FicNumVals.field, maxLength,
        FicNumVals.field[1]))),
                unnest(pretty(array_expand(Idx_IcoVals.field, maxLength,
        Idx_IcoVals.field[1]))),
                unnest(pretty(array_expand(NbrCliVals.field, maxLength,
        NbrCliVals.field[1]))),
                unnest(pretty(array_expand(TailleCliVals.field, maxLength,
        TailleCliVals.field[1]))),
                unnest(pretty(array_expand(N_VVals.field, maxLength,
        N_VVals.field[1]))),
                unnest(pretty(array_expand(C_GVals.field, maxLength,
        C_GVals.field[1]))),
                NEW.Remarques,
                unnest(array_expand(CoordVals, maxLength,
        CoordVals[1])::CoordLambert93[])
            );
        RETURN NULL;
    END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trigger_pretraitement ON CorrectionTemp;
CREATE TRIGGER trigger_pretraitement
    BEFORE INSERT ON CorrectionTemp
    FOR EACH ROW
        EXECUTE PROCEDURE pretraitement();
```

```
NOTICE:  trigger "trigger_pretraitement" for relation "correctiontemp" does not exist,
skipping
```

4 Normalisation de la base de données

Comme nous avons demontré dans ce document, il est préférable que la base soit en 3FN, c'est à dire structurée comme ci:



Nous allons créer une trigger qui, avant d'insérer dans la table correction, distribue les données dans la bonne tables. Les procédés sont similaires à ceux réalisé précédemment.

4.1 Création des tables receptrices:

```
In [5]: DROP TABLE IF EXISTS Document, Photo, Sujet, IndexIconographique, Cliche, TypeOeuvre,
        IndexPersonne, DatePhoto, Serie, Ville CASCADE;

        /* ========================================
         * TABLE: Ville
         * Description:
         * ======================================== */
        CREATE TABLE Ville(
            idVille integer,
            NomVille varchar UNIQUE,
            CoordX numeric NULL,
            CoordY numeric NULL
        );
        ALTER TABLE Ville ADD CONSTRAINT pk_ville PRIMARY KEY(idVille);

        /* ========================================
         * TABLE: Serie
         * Description:
         * ======================================== */
        CREATE TABLE Serie(
            idSerie integer,
            NomSerie varchar UNIQUE
        );
        ALTER TABLE Serie ADD CONSTRAINT pk_serie PRIMARY KEY(idSerie);

        /* ========================================
         * TABLE: DatePhoto
         * Description:
         * ======================================== */
        CREATE TABLE DatePhoto(
            idDate integer,
            DateJour varchar(2) NULL,
            DateMois varchar(2) NULL,
            DateAnnee varchar(4) NOT NULL,
            UNIQUE(DateJour,DateMois, DateAnnee)
        );
```

```
ALTER TABLE DatePhoto ADD CONSTRAINT pk_datephoto PRIMARY KEY(idDate);

/* ======================================
 * TABLE: IndexPersonne, TypeOeuvre
 * Description:
 * ======================================*/
CREATE TABLE TypeOeuvre(
    idType varchar,
    NomType varchar UNIQUE
);
ALTER TABLE TypeOeuvre ADD CONSTRAINT pk_typeOeuvre PRIMARY KEY(idType);
INSERT INTO TypeOeuvre (idType, NomType) VALUES (0, '');

CREATE TABLE IndexPersonne(
    idOeuvre integer ,
    NomOeuvre varchar,
    TypeOeuvre varchar,
    UNIQUE(NomOeuvre, TypeOeuvre)
);
ALTER TABLE IndexPersonne ADD CONSTRAINT pk_IndexPers PRIMARY KEY(idOeuvre);
ALTER TABLE IndexPersonne ADD CONSTRAINT fk_typeOeuvre FOREIGN KEY(TypeOeuvre)
REFERENCES TypeOeuvre(idType);

/* ======================================
 * TABLE: Cliche
 * Description:
 * ======================================*/

CREATE TABLE Cliche(
     idCliche integer ,
     Taille varchar(15) UNIQUE
);
ALTER TABLE Cliche ADD CONSTRAINT pk_cliche PRIMARY KEY(idCliche);

/* ======================================
 * TABLE: IndexIconographique
 * Description:
 * ======================================*/

CREATE TABLE IndexIconographique(
     idIco integer ,
     Idx_Ico varchar UNIQUE
);
ALTER TABLE IndexIconographique ADD CONSTRAINT pk_indexico PRIMARY KEY(idIco);

/* ======================================
 * TABLE: Sujet
 * Description:
 * ======================================*/

CREATE TABLE Sujet(
     idSujet integer ,
     DescSujet varchar UNIQUE
);
ALTER TABLE Sujet ADD CONSTRAINT pk_sujet PRIMARY KEY(idSujet);

/* ======================================
 * TABLE: Photo
 * Description:
 * ======================================*/
CREATE TABLE Photo(
    Article integer,
    Remarques varchar NULL,
    NbrCli varchar NULL,
    DescDet varchar NULL,
    idSerie integer
);
ALTER TABLE Photo ADD CONSTRAINT pk_photo PRIMARY KEY(Article);
```

```
ALTER TABLE Photo ADD CONSTRAINT fk_photo_serie FOREIGN KEY(idSerie) REFERENCES
Serie(idSerie);

/* ======================================
 * TABLE: Document
 * Description:
 * ======================================*/
CREATE TABLE Document(
    PhotoArticle integer,
    Discriminant integer,
    FicNum varchar(28) NULL,
    NoteBP varchar NULL,
    ReferenceCindoc varchar(6) NULL,
    N_V varchar(3) NULL,
    C_G varchar(3) NULL,
    idVille integer NULL,
    idDate integer NULL,
    idOeuvre integer NULL,
    idSujet integer NULL,
    idIco integer NULL,
    idCliche integer NULL
);
ALTER TABLE Document ADD CONSTRAINT pk_document PRIMARY KEY(PhotoArticle, Discriminant);

ALTER TABLE Document ADD CONSTRAINT fk_document_photo FOREIGN KEY(PhotoArticle)
REFERENCES Photo(Article);
ALTER TABLE Document ADD CONSTRAINT fk_document_ville FOREIGN KEY(idVille) REFERENCES
Ville(idVille);
ALTER TABLE Document ADD CONSTRAINT fk_document_date FOREIGN KEY(idDate) REFERENCES
DatePhoto(idDate);
ALTER TABLE Document ADD CONSTRAINT fk_document_oeuvre FOREIGN KEY(idOeuvre) REFERENCES
IndexPersonne(idOeuvre);

ALTER TABLE Document ADD CONSTRAINT fk_document_sujet FOREIGN KEY(idSujet) REFERENCES
Sujet(idSujet);
ALTER TABLE Document ADD CONSTRAINT fk_document_idxico FOREIGN KEY(idIco) REFERENCES
IndexIconographique(idIco);
ALTER TABLE Document ADD CONSTRAINT fk_document_cliche FOREIGN KEY(idCliche) REFERENCES
Cliche(idCliche);
```

4.2 Insertion des données dans les tables

In [6]:
```
/* ======================================
 * FONCTIONS UTILES
 * Description:
 * ======================================*/

DROP FUNCTION IF EXISTS month_to_num;
CREATE OR REPLACE FUNCTION month_to_num(mois varchar(2))
RETURNS numeric(2) AS $$
    BEGIN
        IF mois ILIKE 'Janvier' THEN RETURN 1;
        ELSIF mois ILIKE 'Février' THEN RETURN 2;
        ELSIF mois ILIKE 'Mars' THEN RETURN 3;
        ELSIF mois ILIKE 'Avril' THEN RETURN 4;
        ELSIF mois ILIKE 'Mai' THEN RETURN 5;
        ELSIF mois ILIKE 'Juin' THEN RETURN 6;
        ELSIF mois ILIKE 'Jui{0,1}llet' THEN RETURN 7;
        ELSIF mois ILIKE 'Août' THEN RETURN 8;
        ELSIF mois ILIKE 'Septembre' THEN RETURN 9;
        ELSIF mois ILIKE 'Octobre' THEN RETURN 10;
        ELSIF mois ILIKE 'Novembre' THEN RETURN 11;
        ELSIF mois ILIKE 'Décembre' THEN RETURN 12;
        ELSE RETURN -1;
        END IF;
    END;
$$ language plpgsql;
```

```
DROP FUNCTION IF EXISTS num_to_month;
CREATE OR REPLACE FUNCTION num_to_month(num varchar(2))
RETURNS varchar AS $$
    BEGIN
        IF num = '1' THEN RETURN 'Janvier';
        ELSIF num = '2' THEN RETURN 'Février';
        ELSIF num = '3' THEN RETURN 'Mars';
        ELSIF num = '4' THEN RETURN 'Avril';
        ELSIF num = '5' THEN RETURN 'Mai';
        ELSIF num = '6' THEN RETURN 'Juin';
        ELSIF num = '7' THEN RETURN 'Juillet';
        ELSIF num = '8' THEN RETURN 'Août';
        ELSIF num = '9' THEN RETURN 'Septembre';
        ELSIF num = '10' THEN RETURN 'Octobre';
        ELSIF num = '11' THEN RETURN 'Novembre';
        ELSIF num = '12' THEN RETURN 'Décembre';
        ELSE RETURN '';
        END IF;
    END;
$$ language plpgsql;

-------------------------------------------------------------
-- combine_date_string: combiner les colonnes dans DatePhoto
-------------------------------------------------------------
CREATE OR REPLACE FUNCTION combine_date_string(d varchar[])
RETURNS varchar AS $$
    DECLARE
        res varchar;
    BEGIN
        IF d[2] = '' THEN d[2] := '-1';
        END IF;
        res := array_to_string(ARRAY[d[1], num_to_month(d[2]), d[3]]::varchar[], ' ',
'');
        res := regexp_replace(res, '^\s*(.*)', '\1');
        RETURN trim(BOTH from res);
    END;
$$ language plpgsql;

-------------------------------------------------------------
-- split_date: séparer NEW.Date en 3 colonnes
-------------------------------------------------------------
DROP FUNCTION IF EXISTS split_date;
CREATE OR REPLACE FUNCTION split_date(str varchar)
RETURNS varchar[] AS $$
    DECLARE
        res varchar[] = ARRAY['', '', '']::varchar[];
        arr varchar[] := string_to_array(str, ' ');
        length integer := array_length(arr, 1);
    BEGIN

        IF (str IS NULL) or (char_length(str) = 0) THEN
            RETURN NULL;
        END IF;

        IF length = 1 THEN res[3] := arr[1];
        ELSIF length = 2 THEN
            res[2] := month_to_num(arr[1])::varchar;
            res[3] := arr[2];
        ELSE
            res[1] := arr[1];
            res[2] := month_to_num(arr[2])::varchar;
            res[3] := arr[3];
        END IF;

        RETURN res;
    END;
$$ language plpgsql;
```

```
-------------------------------------------------------------
-- split_oeuvre: séparer NEW.Idx_per en nom et type
-------------------------------------------------------------
DROP FUNCTION IF EXISTS split_oeuvre;
CREATE OR REPLACE FUNCTION split_oeuvre(str varchar)
RETURNS varchar[] AS $$
    DECLARE
        res varchar[] = ARRAY['', '']::varchar[];
    BEGIN

        IF str ~* '.*,\s([A-zÀ-ÿ]+\s*[A-zÀ-ÿ]+$)' THEN
            str := regexp_replace(str, '(.*),\s([A-zÀ-ÿ]+\s*[A-zÀ-ÿ]+$)', '\1|\2');
            res[1] := split_part(str, '|', 1);
            res[2] := split_part(str, '|', 2);
        ELSIF str ~* '^([A-zÀ-ÿ]+\s*,(?:\s*[A-zÀ-ÿ]+\s*))\(([A-zÀ-ÿ]+\s*[A-zÀ-ÿ]+)\)'
THEN
            str := regexp_replace(str,
'^([A-zÀ-ÿ]+\s*,(?:\s*[A-zÀ-ÿ]+\s*))\(([A-zÀ-ÿ]+\s*[A-zÀ-ÿ]+)\)', '\1|\2');
            res[1] := split_part(str, '|', 1);
            res[2] := split_part(str, '|', 2);
        ELSE
            res[1] := str;
        END IF;

        res[1] := trim(BOTH from res[1]);
        res[2] := trim(BOTH from res[2]);

        RETURN res;
    END;
$$ language plpgsql;

/**
 * TRIGGER: trigger_normalisation
 * PROCEDURE: normalisation
 */
CREATE OR REPLACE FUNCTION normalisation()
RETURNS TRIGGER AS $$
    DECLARE
        v_idVille integer;
        v_idSerie integer;
        v_idDate integer;
        v_date varchar[] := split_date(NEW.Date);
        v_idOeuvre integer;
        v_idCliche integer;
        v_idSujet integer;
        v_idIco integer;
        v_idxPer varchar[];
    BEGIN
        -- INSERT dans Ville
        IF (NEW.Ville IS NOT NULL ) THEN
            SELECT INTO v_idVille COUNT(idVille) FROM Ville;
            INSERT INTO Ville(idVille, NomVille, CoordX, CoordY)
            VALUES(
                (SELECT COUNT(idVille) FROM Ville)+1,
                NEW.Ville, (NEW.Coordonnees).CoordX, (NEW.Coordonnees).CoordY)
            ON CONFLICT(NomVille) DO NOTHING;

        END IF;

        -- INSERT dans serie;
        IF (NEW.Serie IS NOT NULL ) THEN
            INSERT INTO Serie(idSerie, NomSerie)
            VALUES(
                (SELECT COUNT(idSerie) FROM Serie)+1,
                NEW.Serie)
            ON CONFLICT(NomSerie) DO NOTHING;
        END IF;
```

```
-- INSERT dans date;
IF (v_date[3] IS NOT NULL) THEN
    --RAISE EXCEPTION 'Inserting % from %', v_date, NEW.Date;
    INSERT INTO DatePhoto(idDate, DateJour, DateMois, DateAnnee)
    VALUES(
        (SELECT COUNT(idDate) FROM DatePhoto)+1,
        v_date[1], v_date[2], v_date[3])
    ON CONFLICT(DateJour, DateMois, DateAnnee) DO NOTHING;
END IF;


-- INSERT dans IdxPers;
v_idxPer := split_oeuvre(NEW.idx_per);
IF (NEW.Idx_per IS NOT NULL ) THEN
    IF v_idxPer[2] != '' THEN
        INSERT INTO TypeOeuvre(idType, NomType)
        VALUES(
            (SELECT COUNT(idType) FROM TypeOeuvre),
            v_idxPer[2]
        ) ON CONFLICT(NomType) DO NOTHING;
    END IF;

    INSERT INTO IndexPersonne(idOeuvre, NomOeuvre, TypeOeuvre)
    VALUES(
        (SELECT COUNT(idOeuvre) FROM IndexPersonne)+1,
        v_idxPer[1],
        (SELECT idType FROM TypeOeuvre WHERE NomType = v_idxPer[2])
    )
    ON CONFLICT(NomOeuvre, TypeOeuvre) DO NOTHING;
END IF;

-- INSERT dans Cliche
IF (NEW.TailleCli IS NOT NULL ) THEN
    INSERT INTO Cliche(idCliche, Taille)
    VALUES (
        (SELECT COUNT(idCliche) FROM Cliche)+1,
        NEW.TailleCli)
    ON CONFLICT DO NOTHING;
END IF;

-- INSERT dans IndexIco
IF (NEW.Idx_Ico IS NOT NULL ) THEN
    INSERT INTO IndexIconographique(idIco, Idx_Ico)
    VALUES(
        (SELECT COUNT(idIco) FROM IndexIconographique)+1,
        NEW.Idx_Ico)
    ON CONFLICT DO NOTHING;
END IF;

-- INSERT dans Sujet
IF (NEW.Sujet IS NOT NULL ) THEN
    INSERT INTO Sujet(idSujet, DescSujet)
    VALUES(
        (SELECT COUNT(idSujet) FROM Sujet)+1,
        NEW.Sujet)
    ON CONFLICT DO NOTHING;
END IF;

-- INSERT dans Photo;
SELECT INTO v_idSerie idSerie FROM Serie WHERE NomSerie = NEW.Serie;
INSERT INTO Photo (Article, Remarques, NbrCli, DescDet, idSerie)
VALUES (NEW.Article, NEW.Remarques, NEW.NbrCli, NEW.DescDet, v_idSerie)
ON CONFLICT DO NOTHING;

-- INSERT dans Document
SELECT INTO v_idVille idVille FROM Ville WHERE NomVille = NEW.Ville;
SELECT INTO v_idDate idDate FROM DatePhoto
```

```
            WHERE DateJour = v_date[1] and DateMois = v_date[2] and DateAnnee =
    v_date[3];


        SELECT INTO v_idOeuvre idOeuvre FROM IndexPersonne i, TypeOeuvre t
        WHERE NomOeuvre = v_idxPer[1] and NomType = v_idxPer[2] and i.TypeOeuvre =
    t.idType;

        SELECT INTO v_idCliche idCliche FROM Cliche WHERE Taille = NEW.TailleCli;
        SELECT INTO v_idIco idIco FROM IndexIconographique WHERE Idx_Ico = NEW.Idx_Ico;
        SELECT INTO v_idSujet idSujet FROM Sujet WHERE DescSujet = NEW.Sujet;

        INSERT INTO Document(PhotoArticle, Discriminant, FicNum, NoteBP,
    ReferenceCindoc, N_V, C_G, idVille, idOeuvre, idDate, idCliche, idIco, idSujet)
        VALUES(NEW.Article, NEW.Discriminant, NEW.FicNum, NEW.NoteBP,
    NEW.ReferenceCindoc, NEW.N_V, NEW.C_G, v_idVille, v_idOeuvre, v_idDate, v_idCliche,
    v_idIco, v_idSujet)
        ON CONFLICT DO NOTHING;

        RETURN NULL;
    END;
    $$ LANGUAGE plpgsql;

    DROP TRIGGER IF EXISTS trigger_normalisation ON Correction;
    CREATE TRIGGER trigger_normalisation
        BEFORE INSERT ON Correction
        FOR EACH ROW
            EXECUTE PROCEDURE normalisation();
```

```
NOTICE:  trigger "trigger_normalisation" for relation "correction" does not exist,
skipping
```

4.3 Vérifier si on a des données perdues.

```
In [7]: CREATE OR REPLACE FUNCTION verifier_normalisation()
    RETURNS bool AS $$
        DECLARE
            bVille bool;
            bIndexPersonne bool;
            bDate bool;
            bSujet bool;
            bCliche bool;
            bIco bool;

            nbVilleDoc integer := (SELECT COUNT(DISTINCT idVille) FROM Document);
            nbVille integer := (SELECT COUNT(DISTINCT idVille) FROM Ville);
            nbOeuvreDoc integer := (SELECT COUNT(DISTINCT idOeuvre) FROM Document);
            nbOeuvre integer := (SELECT COUNT(DISTINCT idOeuvre) FROM IndexPersonne);
            nbDateDoc integer:= (SELECT COUNT(DISTINCT idDate) FROM Document);
            nbDate integer := (SELECT COUNT(DISTINCT idDate) FROM DatePhoto);
            nbSujetDoc integer := (SELECT COUNT(DISTINCT idSujet) FROM Document);
            nbSujet integer := (SELECT COUNT(DISTINCT idSujet) FROM Sujet);
            nbClicheDoc integer := (SELECT COUNT(DISTINCT idCliche) FROM Document);
            nbCliche integer := (SELECT COUNT(DISTINCT idCliche) FROM Cliche);
            nbIdoDoc integer := (SELECT COUNT(DISTINCT idIco) FROM Document);
            nbIdo integer := (SELECT COUNT(DISTINCT idIco) FROM IndexIconographique);

        BEGIN
            SELECT INTO bVille, bIndexPersonne, bDate, bSujet, bCliche, bIco
                    (nbVilleDoc = nbVille),
                    (nbOeuvreDoc = nbOeuvre),
                    (nbDateDoc = nbDate),
                    (nbSujetDoc = nbSujet),
                    (nbClicheDoc = nbCliche),
                    (nbIdoDoc = nbIdo);
```

```
            IF bVille and bIndexPersonne and bDate and bSujet and bCliche and bIco THEN
                DROP TABLE IF EXISTS Correction;
                RETURN true;
            ELSE
                IF not bVille THEN
                    RAISE NOTICE 'Perte de donnée dans Ville: actuel(%) vs attendu(%)',
        nbVilleDoc, nbVille;
                END IF;

                IF not bIndexPersonne THEN
                    RAISE NOTICE 'Perte de donnée dans IndexPersonne: actuel(%) vs
        attendu(%)', nbOeuvreDoc, nbOeuvre;
                END IF;

                IF not bDate THEN
                    RAISE NOTICE 'Perte de donnée dans Date: actuel(%) vs attendu(%)',
        nbDateDoc, nbDate;
                END IF;

                IF not bSujet THEN
                    RAISE NOTICE 'Perte de donnée dans Sujet: actuel(%) vs attendu(%)',
        nbSujetDoc, nbSujet;
                END IF;

                IF not bCliche THEN
                    RAISE NOTICE 'Perte de donnée dans Cliche: actuel(%) vs attendu(%)',
        nbClicheDoc, nbCliche;
                END IF;

                IF not bIco THEN
                    RAISE NOTICE 'Perte de donnée dans IndexIco: actuel(%) vs attendu(%)',
        nbIdoDoc, nbIdo;
                END IF;

                RETURN false;
            END IF;
        END;
        $$ language plpgsql;
```

In [8]:
```
COPY CorrectionTemp( ReferenceCindoc, Serie, Article, Discriminant, Ville, Sujet,
DescDet, Date, Notebp, Idx_per, FicNum, Idx_Ico, NbrCli, TailleCli, N_V, C_G, Remarques
)
FROM '/home/minhhoangdang/L3/S5/BD/TEA/pdfsrc/data.csv' DELIMITER '     ' CSV HEADER;
DROP TABLE IF EXISTS CorrectionTemp;
DROP TABLE IF EXISTS VilleTemp;

SELECT verifier_normalisation();
```

1 row(s) returned.

verifier_normalisation
1