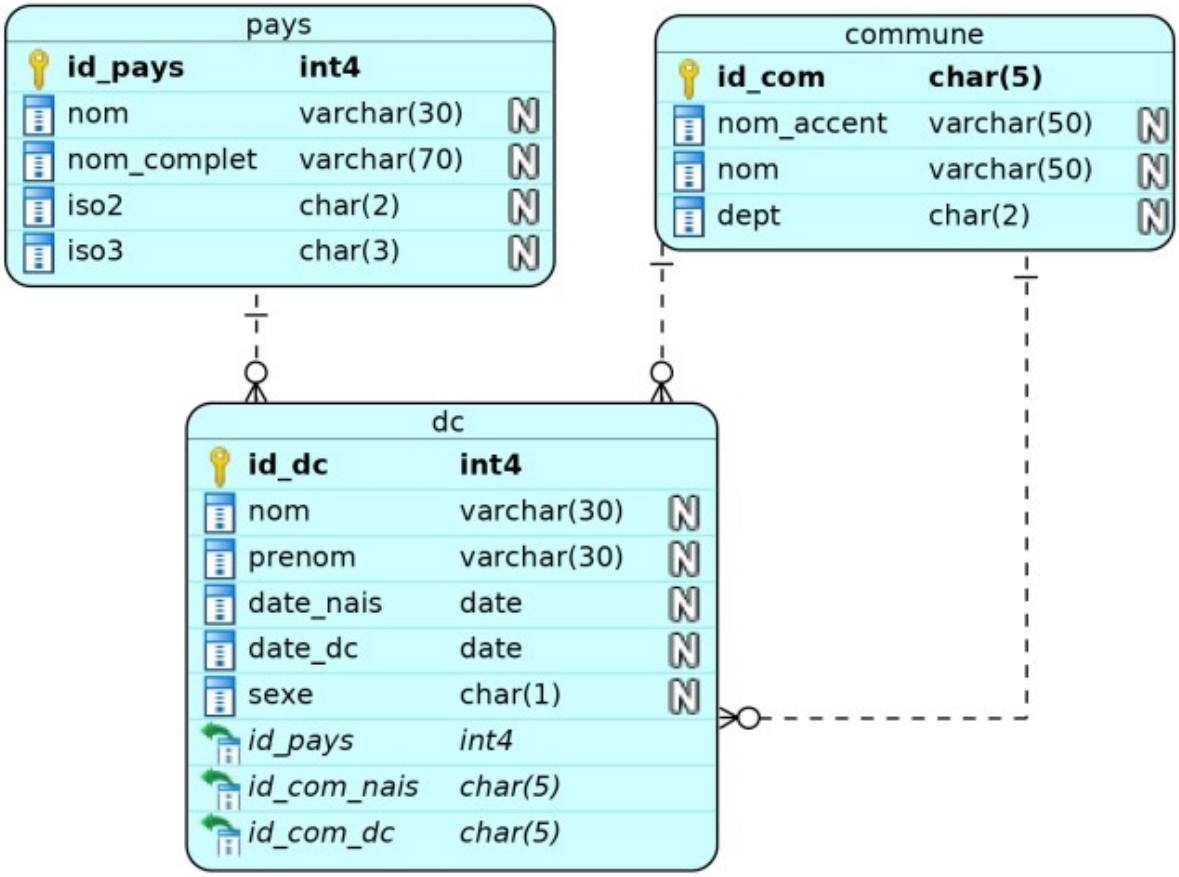
SUIVI DC

Mission n°6 – Portfolio – Antoine Gandelin

SUIVI DC

I. Migration dans une base de données

1. Modèle



2. Scripts de création

Voir les fichiers dc.admin.sql et dc_create.sql

1) Création de la base de données et du rôle d'administrateur

```
create database dc;
create role dc_admin password 'admin' login;
grant all on database dc to dc_admin;
```

2) Création de la structure de la base de données

```
BEGIN TRANSACTION;

CREATE TABLE pays
    (
    id_pays     SERIAL PRIMARY KEY,
    nom         varchar(30),
    nom_complet varchar(70),
    iso2        char(2),
    iso3        char(3)
    );

CREATE TABLE commune
    (
    id_com     char(5) PRIMARY KEY,
    nom_accent varchar(50),
    nom        varchar(50),
    dept       char(2)
   );

CREATE TABLE dc
    (
    id_dc       SERIAL PRIMARY KEY,
    nom         varchar(30),
    prenom      varchar(30),
    date_nais   date,
    date_dc     date,
    sexe        char(1),
    id_pays     integer REFERENCES pays (id_pays),
    id_com_nais char(5) REFERENCES commune (id_com),
    id_com_dc   char(5) REFERENCES commune (id_com)
    );

COMMIT;
```

3. Scripts de migration communes et pays

Les deux premiers scripts sont assez simples à obtenir, la démarche est toujours la même :

1. Création du table tempo qui va contenir les champs du fichier à importer.
2. Requête sur la table tempo permettant d'alimenter la table voulue.

Voir migration_communes.sql et migration_pays.sql

1) Migration des communes

```
begin transaction;

create table tempo
    (
    typecom varchar(50),
    com varchar(50),
    reg varchar(50),
    dep varchar(50),
    arr varchar(50),
    tncc varchar(50),
    ncc varchar(500),
    nccenr varchar(500),
    libelle varchar(500),
    can varchar(50),
    comparent varchar(50)
    );

copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/communes2020.csv' with csv delimiter ',' NULL '' encoding 'utf8';

insert into commune(id_com, nom, nom_accent, dept)
select com, ncc, libelle, dep::char(2)
    from tempo
        where typecom <> 'COMD';

drop table tempo;
commit;
vacuum full;
```

2) Migration des pays

```
begin transaction;

create table tempo
    (
    cog varchar(50),
    actual varchar(50),
    capay varchar(50),
    crpay varchar(50),
    ani varchar(50),
    libcog varchar(50),
    libenr varchar(500),
    ancnom varchar(50),
    codeiso2 varchar(50),
    codeiso3 varchar(50),
    codenum3 varchar(50)
    );

copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/pays2020.csv' with csv delimiter ',' NULL '' encoding 'utf8';

insert into pays(nom, nom_complet, iso2, iso3)
select libcog::varchar(30), libenr, codeiso2::char(2), codeiso3::char(3)
    from tempo;

drop table tempo;
commit;
vacuum full;
```

4. Difficultés rencontrées

Les seules difficultés rencontrées sont les tailles de certaines colonnes de tempo qui sont supérieures à la table cible. Il a fallu remédier à ce problème en convertissant les colonnes. Par ailleurs, il a fallu penser à écarter les lignes qui ont comme type « DCOM » pour éviter des doublons dans la clef primaire de « commune ».

```
dep::char(2)
libcog::varchar(30)
```

5. Script de migration des décès

Cette étape a été beaucoup plus complexe car comme on souhaitait que notre base respecte l'intégrité référentielle, la mise en place des clés étrangères a posé quelques soucis.

Voir migration_dc.sql et fix2csv.cc

```
begin transaction;

create table tempo
    (
    nom varchar(100),
    prenom varchar(100),
    sexe varchar(100),
    date_nais varchar(100),
    code_com_nais varchar(100),
    pays_nais varchar(100),
    date_dc varchar(100),
    code_com_dc varchar(100)
    );

copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m01.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m02.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m03.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m04.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m05.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m06.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m07.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m08.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2021-m09.csv' with csv delimiter ';' NULL '' encoding 'utf8';
copy tempo from 'C:/wamp64/www/SLAM/2eme_annee/suivi_dc/migration_base/fic/deces-2020.csv' with csv delimiter ';' NULL '' encoding 'utf8';



insert into dc(nom,prenom,date_nais,date_dc,sexe,id_pays,id_com_nais,id_com_dc)
    select tempo.nom::varchar(30),prenom::varchar(30),date_nais::date,date_dc::date,sexe,id_pays,c1.id_com,c2.id_com
        from tempo
        left join pays on pays.nom = tempo.pays_nais
        left join commune c1 on c1.id_com = code_com_nais
        left join commune c2 on c2.id_com = code_com_dc;

drop table tempo;

commit;
vacuum full;
```

6. Difficultés

Les difficultés sont nombreuses :

- Code commune des fichiers décès qui ne sont pas dans la table commune
- Idem pour les noms de pays
- Dates incorrectes

Il a fallu régler ces problèmes en plusieurs étapes :

- Décider que lorsqu'un code commune n'existe pas, (cela concerne surtout les codes communes de naissance) on indique une clef étrangère nulle. Cela revient à perdre le nom de la commune, ce qui ne serait pas une solution acceptable dans le cadre d'un projet réel.
- Idem pour pays
- Pour les problèmes de date, il a fallu modifier le code du programme de transformation des fichiers à champ fixe vers des fichiers csv, notamment en modifiant le code de la fonction « transfo_date() ».

II. Récapitulatif de la migration

On peut donc maintenant réaliser la migration complète vers la base pour préparer la prochaine étape qui consistera à développer une application web de gestion de ces données.

1. Étape n°1

Lancer le programme « fix2csv » à travers le script bash « process.sh ». On considère que tous les fichiers à champs fixes sont dans le répertoire « fic ».

```
./process.sh fic/*.txt
```

2. Étape n°2

Lancer les scripts de création de la base. On se connecte en DBA (postgres) :

```
$ su postgres
```

On lance le client de Postgresql :

```
$ psql
```

Sous psql, on lance le script de création de la base et du rôle :

```
postgres=# \i /chemin_ou_se_trouve_le_script/dc_admin.sql
CREATE DATABASE
CREATE ROLE
GRANT
```

On quitte le client psql, on quitte la session bash postgres et on lance le client psql avec les valeurs voulues :

```
$ psql dc -U dc_admin -h 127.0.0.1

dc=> \i /chemin_ou_se_trouve_le_script/dc_create.sql
BEGIN
CREATE TABLE
CREATE TABLE
CREATE TABLE
COMMIT
```

On vérifie que les tables ont été correctement crées :

```
dc=> \d
                  Liste des relations
 Schéma  |       Nom        |   Type   | Propriétaire
 --------+------------------+----------+--------------
 public  | commune          | table    | dc_admin
 public  | dc               | table    | dc_admin
 public  | dc_id_dc_seq     | séquence | dc_admin
 public  | pays             | table    | dc_admin
 public  | pays_id_pays_seq | séquence | dc_admin
```

```
(5 lignes)
```

3. Étape n°3

On lance maintenant les scripts de migration. Mais attention, ceux-ci ne peuvent s'exécuter qu'en tant que DBA (postgres).

On peut déplacer avant les fichiers csv dans le répertoire migration_base/fic/ de ou créer un lien symbolique sur le répertoire migration_fic/fic.

On modifie ensuite les chemins des fichiers dans les scripts de migration pour tenir compte de l'emplacement des fichiers csv.

Par exemple, la ligne COPY pourrait prendre cette forme :

```
copy tempo from
     '/chemin/suivi_dc_4/migration_base/fic/communes2020.csv'
     with csv
     delimiter ','
     NULL ''
     encoding 'utf8';
```

Faire de même pour tous les scripts de migration (utiliser le chercher/remplacer sur tous les documents ouverts avec Bluefish est une bonne idée).

Une fois les scripts modifiés, nous pouvons les lancer en tant que DBA.

On se connecte en DBA (postgres) :

```
$ su postgres
```

On lance le client de Postgresql :

```
$ psql dc
```

Puis on lance les scripts :

```
dc=# \i /chemin/suivi_dc_4/migration_base/migration_commune.sql
BEGIN
CREATE TABLE
COPY 37902
INSERT 0 35560
DROP TABLE
COMMIT
VACUUM

dc=# \i /chemin/suivi_dc_4/migration_base/migration_pays.sql
BEGIN
CREATE TABLE
COPY 283
INSERT 0 283
DROP TABLE
COMMIT
VACUUM
```

```
dc=# \i /home/ant/public_html/suivi_dc/migration_base/migration_dc.sql
BEGIN
CREATE TABLE
COPY 66106
COPY 59039
COPY 61197
COPY 58482
COPY 51278
COPY 50387
COPY 49617
COPY 49864
COPY 53219
COPY 679917
INSERT 0 1179106
DROP TABLE
COMMIT
VACUUM
```

Notre base est maintenant effective, on peut vérifier par quelques requêtes si les tables sont remplies correctement :

```
dc=# select count(*) from commune ;
 count
-------
 35560
(1 ligne)

dc=# select count(*) from pays ;
 count
-------
  283
(1 ligne)

dc=# select count(*) from dc ;
  count
---------
 1179106
(1 ligne)
```

L'ensemble des fichiers se trouvent sur :
http://www.sio-reims.fr/periodes/suivi_dc_4.zip