« Datavisualisation et webmapping sur les sports dans les intercommunalités d'Occitanie »

MASTER 2 SIGMA

23/02/2017

Enseignant-Encadrant: Laurent JEGOU

BOSQUE Sabrina DASQUET Florent KIOUSIS Panagiotis

Sommaire

1. Introduction

L'État Français via le Ministère de la Jeunesse et des Sports mobilise des financements pour que les enfants dans les écoles primaires aient un accès à la piscine et aux sports en général. Ces installations sont gérées par les communes sauf si l'intercommunalité (ou EPCI) a choisi de gérer la compétence « Sport ». Pour permettre une gestion commune des infrastructures, de nombreuses EPCI ont pris cette compétence et donc allouée un budget pour le sport au sein de leurs périmètres. Cependant les EPCI et leurs relations avec le sport sont mal connues en France, notamment sur le nombre de pratiquant, les financements alloués, le pourcentage de licenciés par fédération … C'est dans ce cadre-là que les étudiants de la Licence 3 APTER promotion 2015-2016 ont réalisés un travail de fiche de synthèse pour chacun des EPCI de la région Midi Pyrénées pour la Direction Régionale De la Jeunesse, des Sports et de la Cohésion Sociale encadré par M. Laurent JEGOU.

Un site internet a par la suite été réalisé par M. Laurent JEGOU, à partir de ces fiches de synthèses introduisant des éléments purement administratifs tels que le nom du président de l'EPCI, les quartiers prioritaires, le nombre d'écoles primaires, collèges et lycées, des éléments financiers avec la part de population imposable, les revenues moyens, le taux de pauvreté, le prélèvement des impôts locaux, les financements de l'état vers l'EPCI ainsi que des éléments sociodémographique sur les populations locales. Enfin des éléments liés aux pratiques sportives comme l'âge des équipements, le type de pratique, le nombre de licenciés, le nombre de pratiquant par sport et par EPCI composent ces fiches synthétiques. Ainsi toutes ces données sont accessibles en ligne sur le site http://www.geotests.net/test/sports/ .

Cependant, deux évènement sont modifiés les administrations territoriales avec d'abord la fusion des régions avec un passage de 27 à 18 région en 2016, puis la fusion de nombreux EPCI avec un passage de 2062 à 1266 d'EPCI en 2017.

Notre travail arrive après ces étapes et dans ce cadre-là. Il consiste dans un premier temps à mettre à jour les données statistiques notamment avec les nouvelles données INSEE et liées les sports. Dans un second temps, nous devions améliorer le rendu visuel du site en modifiant la cartographie et les diagrammes illustrant l'information statistique, afin de le rendre plus dynamiques et plus interactifs pour l'utilisateur.

2. Importation des données

L'objectif de cette étape est de faciliter la procédure de mise à jour de la base de données, c'est-à-dire permettre à quelqu'un de peu initié dans le domaine de la programmation mais ayant tout de même des connaissances en gestion de base de données de pouvoir mettre à jour des tables de façon simple et rapide.

Les bases sont gérées par le gestionnaire de base de données PostgreSQl.

Lors de la récupération des données en particulier sur le site de l'INSEE et sur data.gouv.fr, nous avons privilégiés le format CSV car nous verrons par la suite qu'il facilite la mise à jour sur les logiciels testés.

En comparant chaque processus, ainsi que la réalisation d'une estimation des besoins, combiné avec le coût, le temps imparti et aussi les résultats souhaités pour cette étape dans la réalisation du projet, nous choisissons un processus d'importation de données simple.

L'application des étapes du mini-tuto que nous allons présenter ci-dessous est suffisant pour mettre à jour la base de données.

1. Choix d'un logiciel

Cette procédure peut être réalisée à partir de différentes techniques. Pour cela, nous avons testés les mêmes manipulations sur quatre logiciel, trois libre et gratuit (PgAdmin 3, HeidiSQL, Adminer) et un payant (Navicat). D'autres logiciels existent mais sont plus complexes d'utilisations, SQuirrel a été éliminé assez rapidement au vu de sa complexité d'utilisation.

Pour cela, nous avons réalisés un tableau comparatif des manipulations utilisés comme un évaluateur de ces outils.

Figure 1: Tableau comparatif des logiciels

	PgAdmin3	Heidi SQL	Adminer	Navicat
Connexion au serveur	oui	oui	oui	oui
Insertion de données	oui (csv)	oui (csv; txt)	oui	oui (xls; xlsx; csv; txt)
Sauvegarde des options de données	non	non	non	oui
Mise à jour des données (sans création de nouvelle table)	non	non	non	oui
Modification des types de données	oui avec programmation	oui avec programmation	~	oui sans programmation
Visualisation des données	oui	oui	oui	oui
Version portable (USB)	non	oui	~	non
Version mobile (iOS, Android)	non	non	non	oui

Les trois premiers outils de gestion de base de données présentent tous des avantages. Bien qu'HeidiSQL se détache des autres par des options qui facilitent l'importation et la mise à jour des données. Nous pouvons émettre comme aspect négatif une interface un peu vieille et la nécessité de chercher les options. Néanmoins, cette outil est facile à installer, à connecter aux serveurs, et offre une gestion de fichier satisfaisante.

Navicat, le logiciel payant est bien plus adéquat que les solutions gratuites. Il permet des manipulations de base (importation et mise à jour) et des procédures bien plus performantes, son interface est facile à prendre en main. Le prix du logiciel Navicat pour PostgresSQL varie de 99$ pour l'Édition Non- commerciale jusqu'à 259.00$ pour l'Édition professionnelle.

2. Technique commune à l'importation des données

La première étape consiste à aller récupérer les données. Elles sont pour la plupart issues de l'INSEE pour les données sociodémographiques, DRJSCS ou du ministère de la ville de la jeunesse et des sports pour les équipements sportifs et les licences, et de nosdonnées.fr pour les finances des EPCI.

La seconde étape a pour but de sélectionner et connaître les données ainsi que le type de fichier à utiliser. Il est nécessaire pour cela de prendre connaissance du "type" des colonnes, par exemple varchar correspondra à une colonne de type texte, int à des nombres entiers, real à des nombres réels…, il existe de nombreux formats d'où l'importance d'en prendre connaissance.

Un travail a été amorcé afin de faciliter la mise à jour du code source du site internet, l'étape suivante réside en la normalisation des tables. Nous avons donc renommé et organisé les tables pour qu'elles suivent le modèle déjà présent sur le site internet afin de ne pas avoir à modifier le code par la suite.

Figure 2: Normalisation des tables pour les introduire dans la base de donnée

donnée de base	données mise à jour	normalisation des données
liste_EPCI_typo		epci_typologie_(année)
RES_France	RES_fiche_equipement	equipement_(année)
base_cc_emploi_pop_active_2007_coms	base_cc_emploi_pop_active_2008_coms	emploi_(année)
base_cc_emploi_pop_active_2012_coms	base_cc_emploi_pop_active_2013_coms	emploi_(année)
bpe14_enseignement_xy		ecole_(année)
codes_fedes	federation_et_codes_fichiers_club	federation_(année)
epci_competences_2015		epci_competences_(année)
epci_coms_mp_2015	epci_commune_2017	epci_commune_(année)
epci_finances_2012		epci_finances_(année)
epci_geojson		com_geojson
fin_communes		commune_finances_(année)
Licences_2012	licence_2014_occ	licences_(année)
naissances_deces_2004_2014	base_naissance_2015	accroissement_(année)
	base_deces_2015	accroissement_(année)
part_menages_fiscaux_2012	filo_revenu_pauvrete_menage_2015	part_fiscalite_(année)
pop_sexe_age_quinquennal2012		
pop_active_coms_2012		emploi_(année)
pop_coms_2007_2012	Pop_coms_2008_2013	Evolution_pop_13
pop_coms_hist		Commune_histo_(année)
qpv_mp		quartier_prioritaire_(année)
Rugby_2014	licence_2014_occ	licences_(année)

À partir de cette étape nous supposerons que la sélection des tables a été faite.

3. Gestion de la base de données avec le logiciel Heidi SQL-Solution Gratuite

“HeidiSQL est un outil d'administration de base de données possédant un éditeur SQL et un constructeur de requête. Il a été développé et optimisé pour être utilisé avec le SGBD relationnel MySQL disponible commercialement ou gratuitement”. (https://fr.wikipedia.org/wiki/HeidiSQL)

“C’est un logiciel gratuit qui permet de parcourir et de modifier des données, de créer et de modifier des tables, des vues, des procédures, des déclencheurs et des événements planifiés. En outre, vous pouvez exporter la structure et les données vers le fichier SQL, le presse-papiers ou vers d'autres serveurs.” (http://www.heidisql.com/).

Il faut noter que le logiciel offre une édition portable qui permet la gestion de base de tout ordinateur.

Avec HeidiSQL deux modes d’importation correspondent à notre cas. La première une création de tables manuelle et la seconde une création de table avec le code SQL.

Dans le premier cas, lors de l’importation des tables, Il est possible de sélectionner les colonnes désirées d’une table.

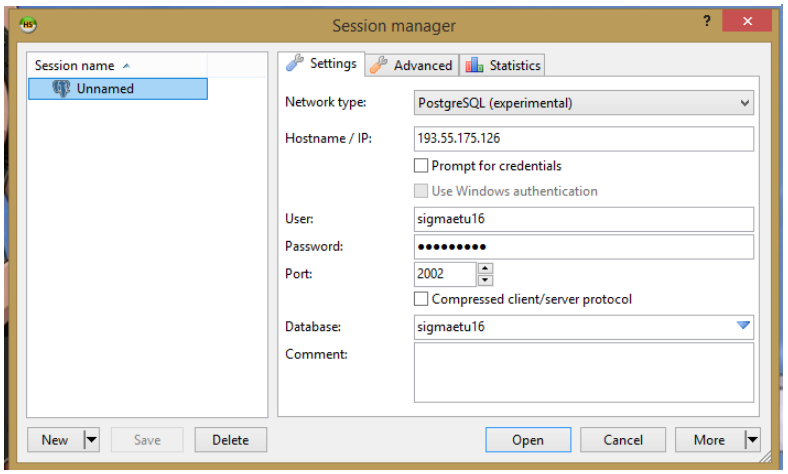
L’importation doit impérativement être réalisée à partir de table **au format .CSV** pour cela nous préconisons l’utilisation de Libre Office Calc qui permet facilement et rapidement de passer d'un format .XLS ou .XLSX en .CSV en réenregistrant le fichier avec un séparateur point-virgule. Il faut porter une attention particulière aux codes régionaux, départementaux et INSEE commençant par un 0 par exemple l’Ariège avec 09, sur certains cas il est nécessaire de forcer le passage en texte de la colonne avec le code INSEE.

ATTENTION. Il faut avoir une bonne correspondance par rapport au “type” et au nombre de colonne qui sont créées et celle choisi à l'importation.

Un autre moyen pour réussir l’importation des données est de faire une exportation de la forme de la table existante sur SQL par PhpPgAdmin et de faire après l’importation en changeant seulement le nom de la table.

Tutoriel d'importation avec HeidiSQL :

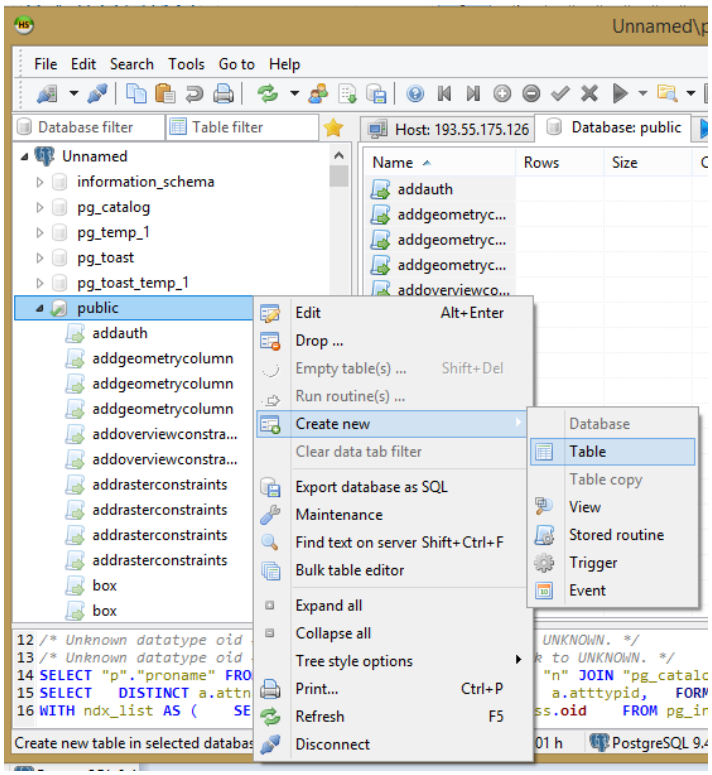
Figure 3: Etape 1: Connexion au serveur



Les champs à remplir pour la connexion sont les suivants Network type : PostgreSQL(experimental)

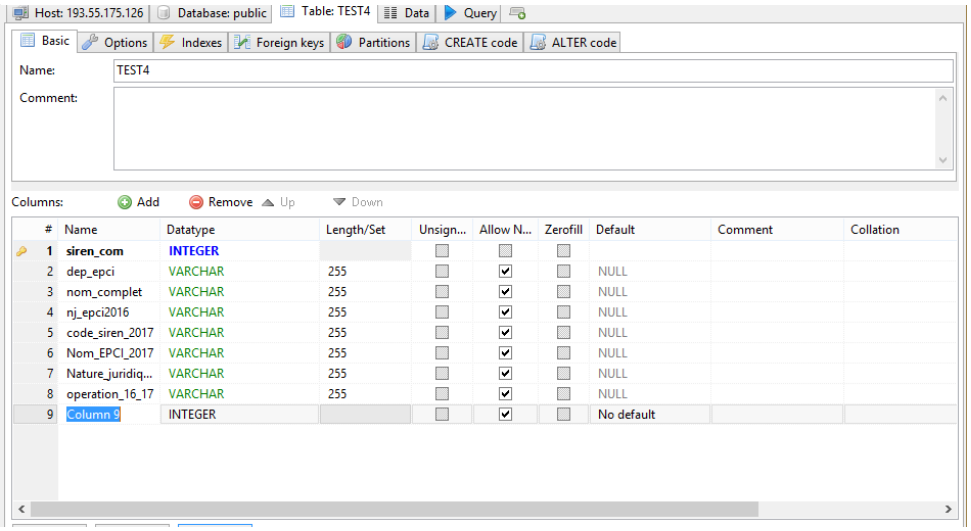
Hostname : L'IP du Server User : username Password : Mot de passe Port : ***** Database:******

Figure 4: Création de la table



Après la transformation d'un tableur au format Excel vers un format CSV (procédure décrite ci-dessus), il faut créer une table avec les champs qui vont nous être utile en définissant le type de chaque champ (Figure 4 & Figure 5).

Figure 5: Création de champs



Ensuite nous ajoutons la clé primaire :

Indexes-> Add-> (clic droit sur le Primary key) ->Add column Figure 6

La prochaine étape est l'importation du tableur en CSV.

Tools-> Import CSV file

Sur la fenêtre qui apparaît on choisit la forme d'importation et la table que nous avons créée avec les champs qui vont être utilisés (Figure 6).

Figure 6: Création de la clé primaire

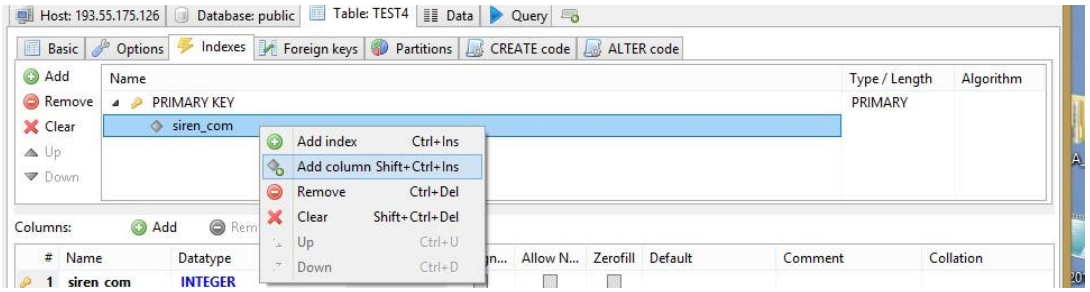
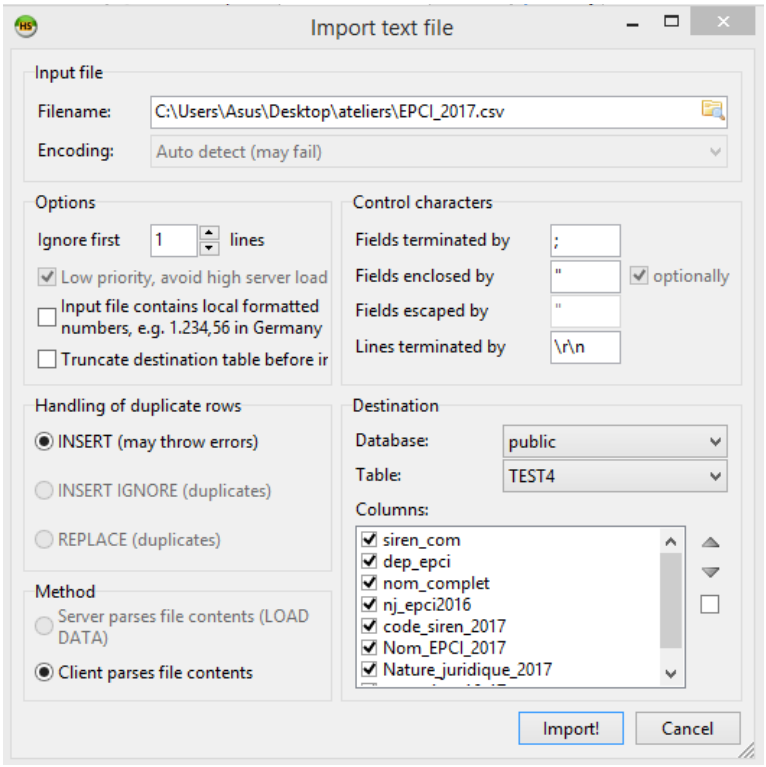


Figure 7: Importation du CSV



Si les tables dont nous avons besoin sont déjà présentes sur le serveur nous pouvons passer l'étape de la création normale.

Par le site http://176.31.110.159/phppgadmin nous pouvons faire un export du code SQL de tables. Ensuite nous pouvons gérer le code SQL directement sur logiciel de HeidiSQL. Nous pouvons effectuer la même démarche sur le logiciel pgAdmin III.

À ce stade, il convient de souligner que l'utilisation du programme HeidiSQL bien qu'il soit meilleur que les autres procès, n'a pas offert beaucoup plus de fonctionnalités que PgAdmin III. Nous pouvons constater également que l'import de gros tableur (supérieur à 1millions de ligne) entraîne des plantages récurrents de ces programmes. Il est donc conseillé de les ajouter sur le serveur directement par le Shell de Postgres.

4. NAVICAT - Solution Payant

Navicat est une suite logicielle graphique de gestion et de développement de bases de données produites par PremiumSoft CyberTech Ltd pour MySQL, MariaDB, Oracle, SQLite, PostgreSQL et Microsoft SQL Server.

Son interface utilisateur graphique est simple et similaire à celle d'Explorer, il supporte des connexions multiples vers des bases de données locales et distantes. Dans notre cas, nous allons utiliser l'édition trial de postgreSQL.

L'utilisation de ce logiciel permet d'éviter bon nombre d'étapes fastidieuses, comme la transformation d'une feuille Excel en CSV ou la création d'une table séparément sur le logiciel. De plus, la connaissance du langage informatique SQL n'est pas nécessaire, il suffit d'avoir quelques bases en gestion de base de données.

L'installation du logiciel est très simple et intuitive.

La première étape consiste à connecter le logiciel au serveur

Connexion

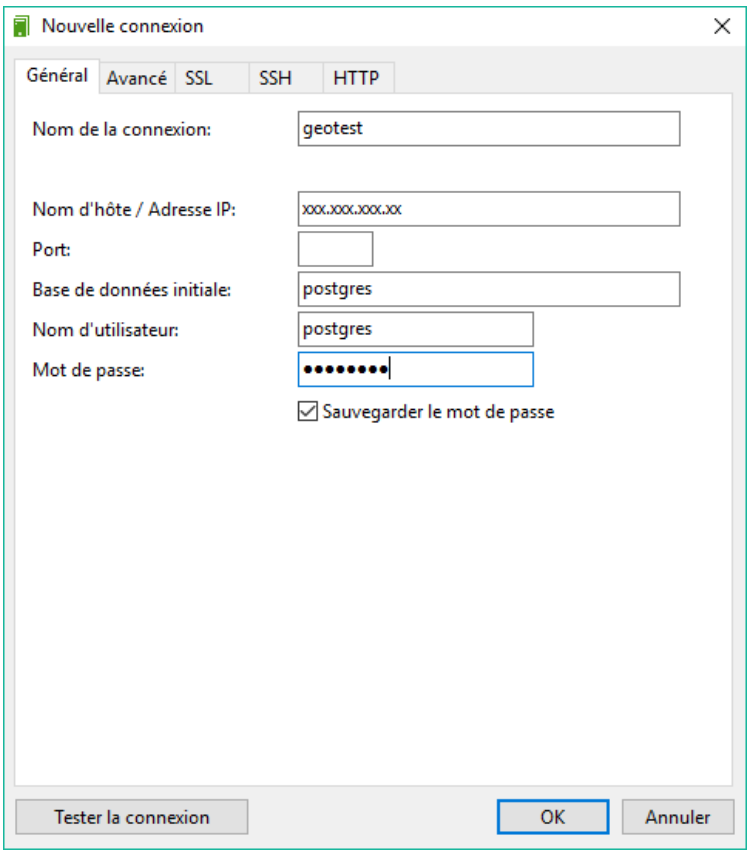


Figure 8: Connexion

Import des tables

La seconde étape consiste à importer la table.

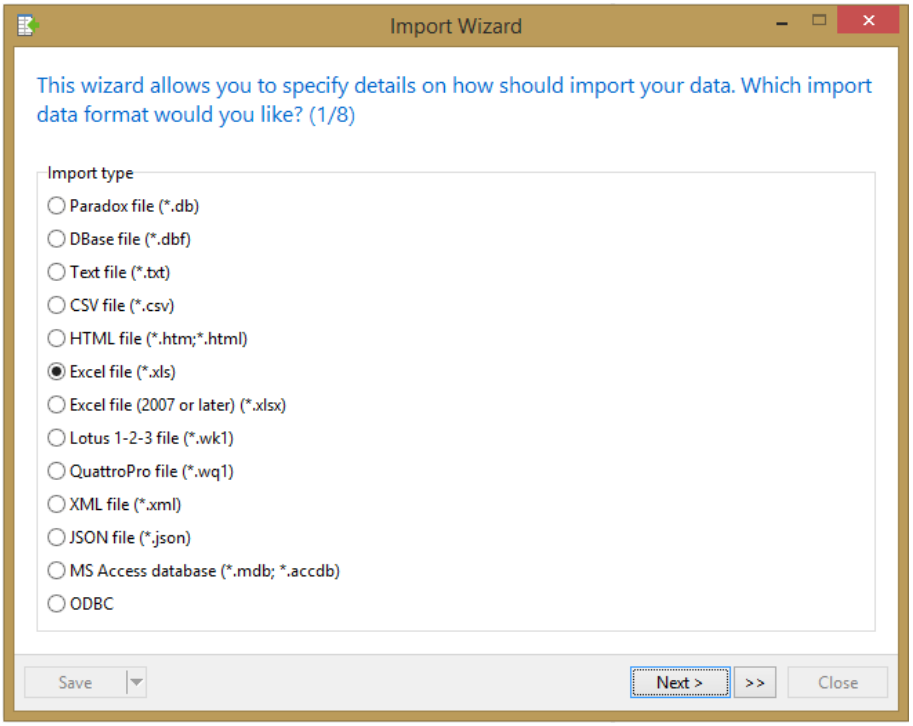


Figure 9: Choix du format (Etape 1/8)

L'importation des données se déroule en huit phases. La première phase consiste à sélectionner le type de fichier que nous voulons ajouter.

Les formats compatibles sont : *.db, *.dbf, *.txt, *.csv, *.htm, *.html, *.xls, *.xlsx, *.wk1, *.wq1,

*.xml, *.json, *.mdb, *.accdb, connexion ODBC. (Image 9).

Dans le cadre de la mise à jour des données, les formats utilisés sont Excel (*.xls, *.xlsx) , DBase (*.db) et du CSV (*.csv).

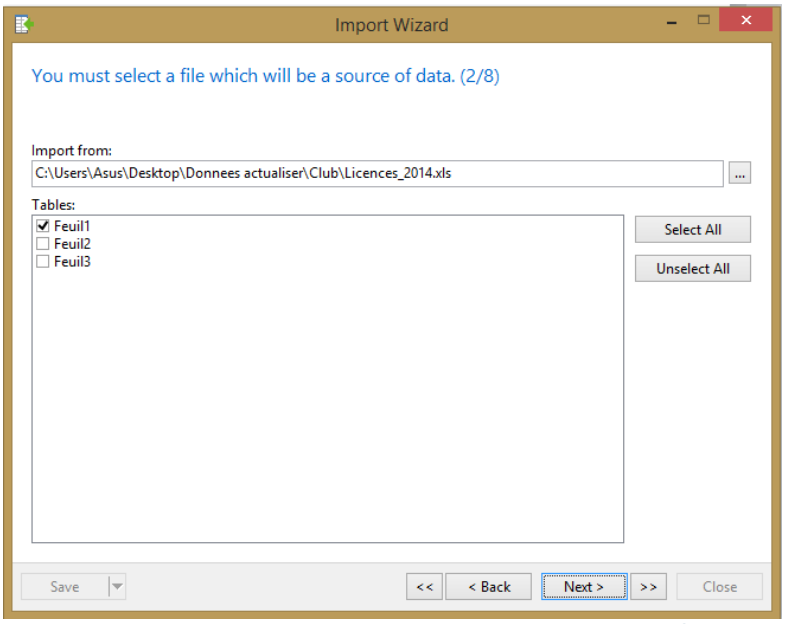


Figure 9: Choix de la feuille de calcul à l'importation(Etape 2/8)

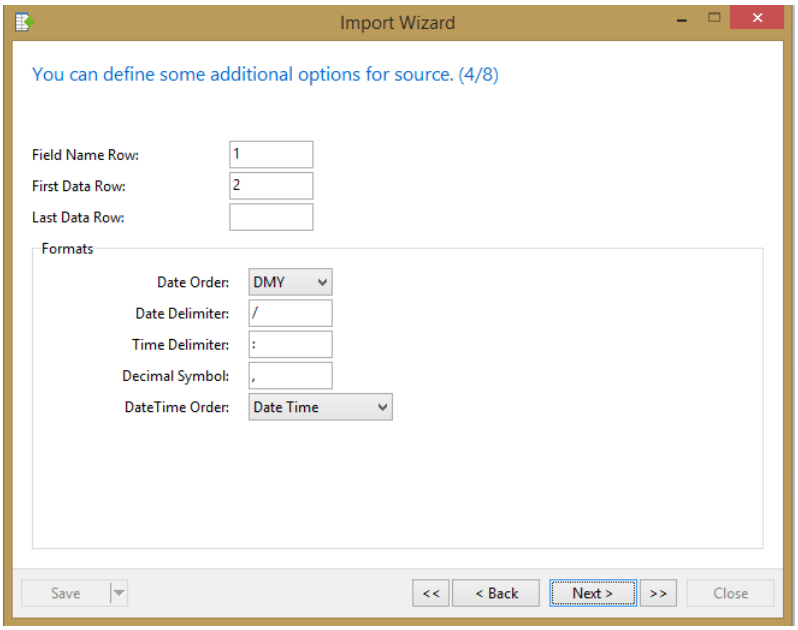


Figure 10: Choix des options du fichier-source (Etape 4/8)

Ensuite, nous choisissons des options supplémentaires, comme la ligne à partir de laquelle l'importation de la table commence, le délimiteur utilisé ou la date de création de la table.

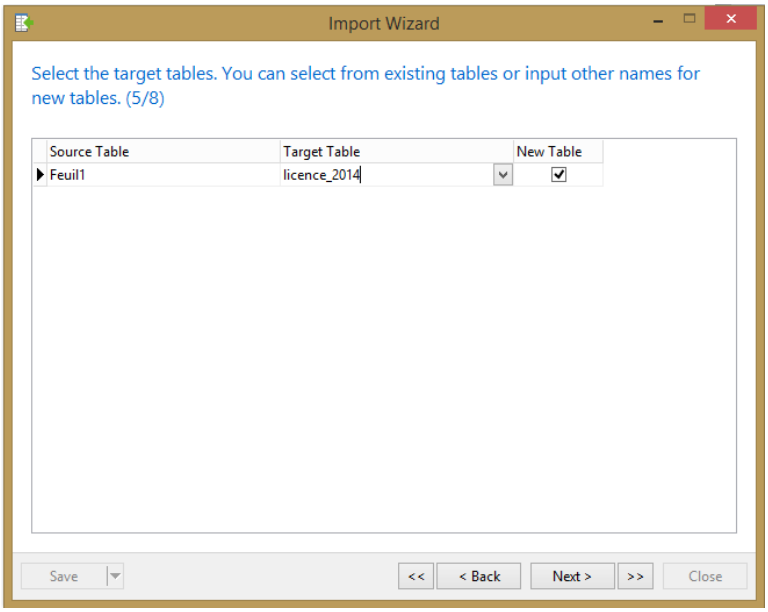


Figure 11:- Table source/Table cible(Etape 5/8)

Sur l'image 11, deux options sont possibles : importer les données d'une table existante ou créer une nouvelle.

Le logiciel nous demande ensuite le type de champs, la longueur de celui-ci, et la clé primaire (image 12).

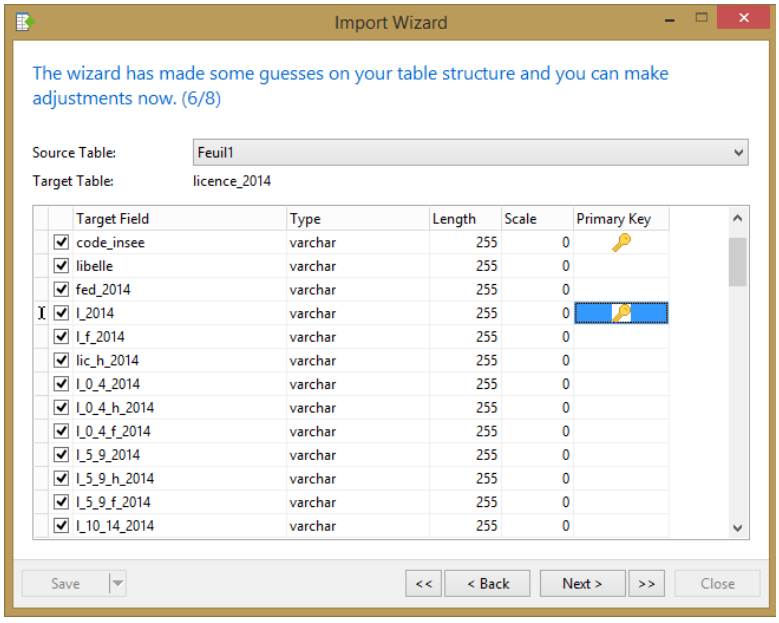


Figure 12: Choix de type/ Clé primaire (Etape 6/8)

Il est ensuite possible de configurer l'option de mise à jour Update, Append/Update, Delete, ou Copy.

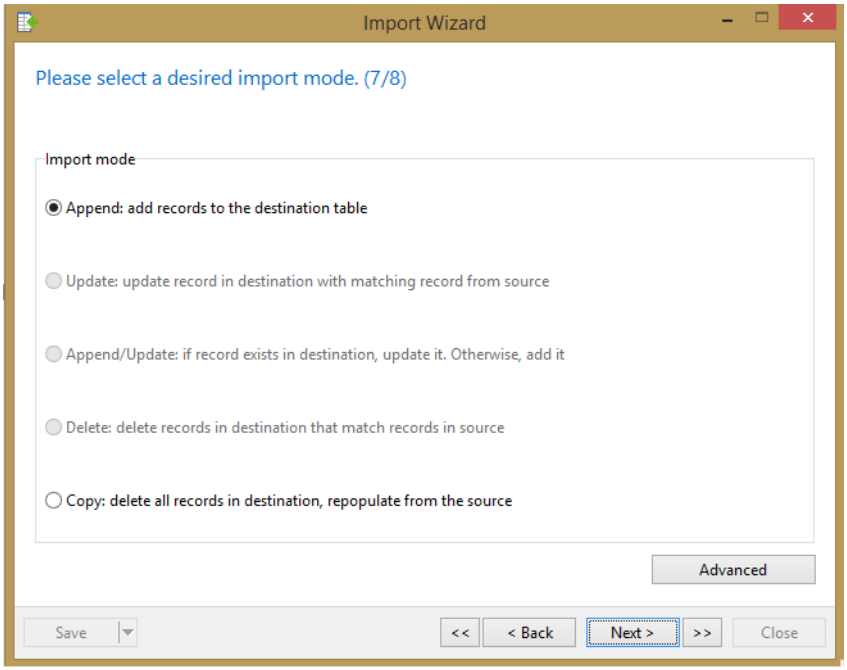


Figure 13 : Mode d'importation (Etape 7/8)

Le logiciel offre la possibilité de réaliser une mise à jour des tables existantes. Pour cela, la table existante et le tableau doivent posséder est le même nombre de colonnes.

Sauvegarder votre importation

Le logiciel de Navicat permet de sauvegarder les options choisies pendant les étapes de l'importation. Cette opération nous donne la possibilité de garder une trace de nos activités et de les réutiliser à l'avenir pour faciliter l'importation. De plus, dans le cas d'une panne pendant l'importation les données, nous pouvons ré-ouvrir le programme et poursuivre à partir du processus ponctuel que nous avions quitté. Il y a également la possibilité de sauvegarder aussi le log. (Image 14, étape 8/8).

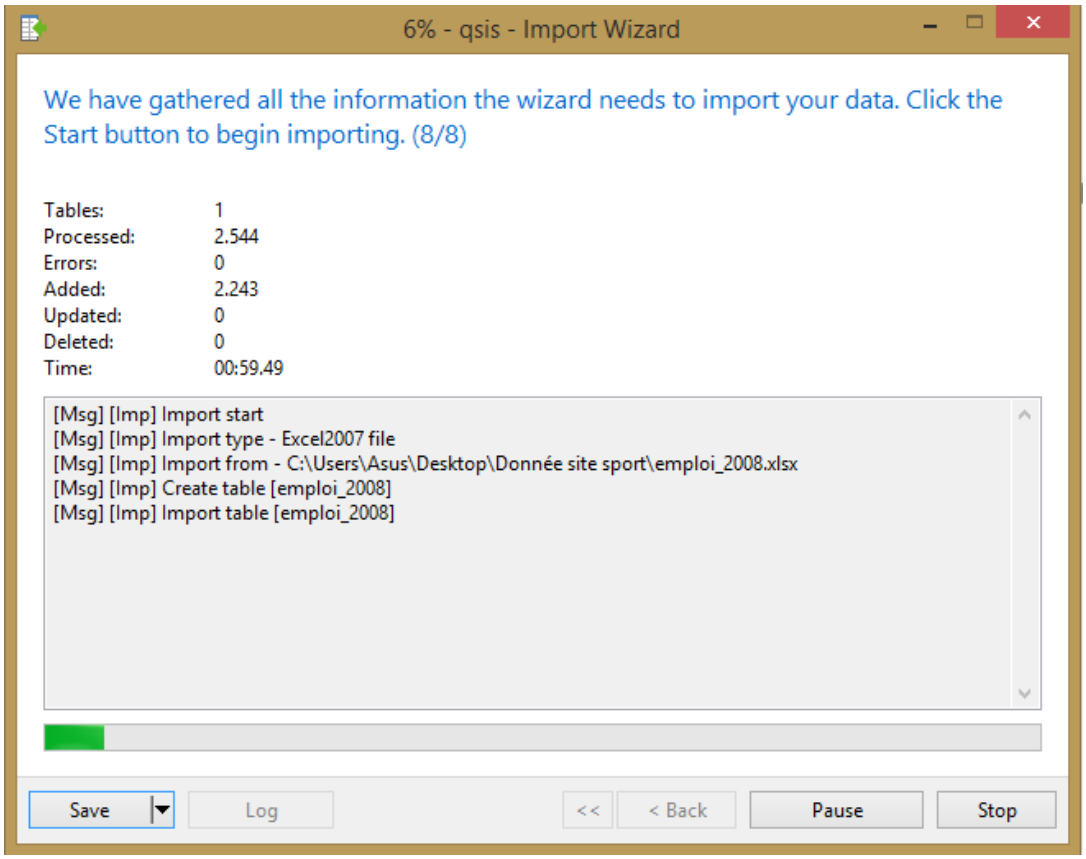


Figure 14: Processus de l'importation (Etape 8/8)

5. Conclusion pour l'importation et la mise à jour des données

Pour conclure, le logiciel gratuit pgAdmin III et HeidiSQL sont les deux solutions gratuites les plus performantes. Le logiciel NAVICAT commercialisé à un prix de 99$ pour la version non commerciale facilite considérablement l'import de données sur divers types de fichiers.

Pour l'automatisation des données, nous avons réfléchis à un programme php récupérant la table dont la valeur année est la plus haute. Ce programme fonctionne avec deux fichiers mais pas avec plus. Faute de temps nous n'avons pas pu poursuivre ce travail d'automatisation des données.

```
//connexion à la base PostGres require("connecter.php");
//Selectionne toutes les années trié par ordre décoissant avec 1 affichage
$r_years = "SELECT annee FROM pannee ORDER BY annee DESC LIMIT 5";
$matrice_years = pg_query($r_years)or die ("erreur requête years");
//Récupère les résultats de la requete et les places dans une matrice
```

```
// avec l'affichage de l'année
while ($ligne_y=pg_fetch_array($matrice_years))
{
$annee=$ligne_y[annee]; print"$annee";
}
//Requete sur les genres de musique où année est le résultat de la requete précedente
$requete = "SELECT * FROM pgenre_".$annee." ORDER BY idgenre";
$matrice = pg_query($requete) or die ("erreur requête");

while ($ligne=pg_fetch_array($matrice))
{
$idgenr=$ligne[idgenre];
$nomgenre=$ligne[nomgenre]; print"<tr><td>$idgenr</td> <td>$nomgenre</td></tr>";
}
pg_close($connexion);
//Les données récupérées sont celle de la dernière année trouvé
echo"</table></center>";
?>
</body></html>
```

3. Mise à jour des éléments graphiques

Dans le cadre des ateliers du master SIGMA, deux missions nous ont été confiées, la première consiste à mettre à jour des données du site de la DRJSCS, cette étape est décrite dans la première partie. L'objectif de la deuxième tâche est d'améliorer graphiquement le site internet, en modifiant la cartographie, mais aussi les diagrammes présents, tout en préservant une lecture de l'information simplifiée et rapide.

Ce document présente les fonctionnalités ajoutées sur le prototype web disponible à l'adresse suivante : http://geotests.net/ficheepci/. Les principales étapes techniques sont également abordées, permettant un suivi du travail établi en cas de futurs travaux sur ce site.

Pour la réalisation des graphiques nous avons utilisé la bibliothèque D3.js utilisant « le langage JavaScript [et permettant] l'affichage de données numériques sous une forme graphique et dynamique. Il s'agit d'un outil important pour la conformation aux normes W3C qui utilise les technologies courantes SVG, JavaScript et CSS pour la visualisation de données.“[1]

Le site web est constitué de deux fichiers décrits ci-dessous:

- index.php: ce fichier nous permet de réaliser les requêtes SQL dont nous avions besoin pour créer les variables. Ces variables sont ensuite stockées dans un supra tableau nommé $params faisant le lien entre les deux fichiers, et appelé dans la maquette sous une forme bien particulière {{variable}}

- maquette.phtml: ce fichier appelle les variables stockées dans l'index, nous permettant de réaliser toute la mise en page et l'aspect graphique. Ce fichier est lui-même connecté à un fichier Twig qui est un template conçu pour être dynamique. Il permet donc de construire la page sur demande sans avoir toutes les fiches EPCI stockées en dur sur le serveur. La maquette est divisée en trois partie, la première permet de faire la connexion avec l'index, la seconde concerne toute la mise en page générale de la page et la mise en forme des tableaux, enfin la troisième partie est la construction de tous les éléments graphiques (cartographie, histogramme, treemap, graphique donuts).

Ainsi pour bien comprendre, nos ajouts il faut réaliser que les deux fichiers sont complémentaires et travaillent ensemble, car nous allons faire des aller-retour entre eux dans ce document.

[1] Wikipedia : article D3.js disponible : https://fr.wikipedia.org/wiki/D3.js

1. Les menus déroulants

Pour l'actualisation du menu déroulant, nous avons dû changer de table référence.

```
SELECT num_siren_epci,
       nom_epci,
       substring(code_insee_commune_siege from 1 for 2) as dept
FROM public.epci_competences_2015
ORDER BY nom_epci
```

Sur l'ancienne version du site, le nom, et le département provenaient de la table epci_competences_2015, aucune autre version de cette table n'est actuellement disponible. De plus, elle recense les anciennes EPCI, et non celle de 2017 uniquement pour l'ex région Midi-Pyrénées.

Nous avons donc effectué la même requête mais sur la table actualisée des EPCI:

```
SELECT siren_epci,
       min (nom_epci) as nom_epci,
       dep_epci
FROM epci_commune_2017
       codgeo
WHERE dep_epci = '09' or dep_epci ='11'or dep_epci ='12' or dep_epci ='30' or
dep_epci ='31' or dep_epci ='32' or  dep_epci ='34' or dep_epci ='46' or dep_epci
='48' or dep_epci ='65' or dep_epci ='66' or dep_epci ='81' or dep_epci ='82'
GROUP BY siren_epci,
       dep_epci;
```

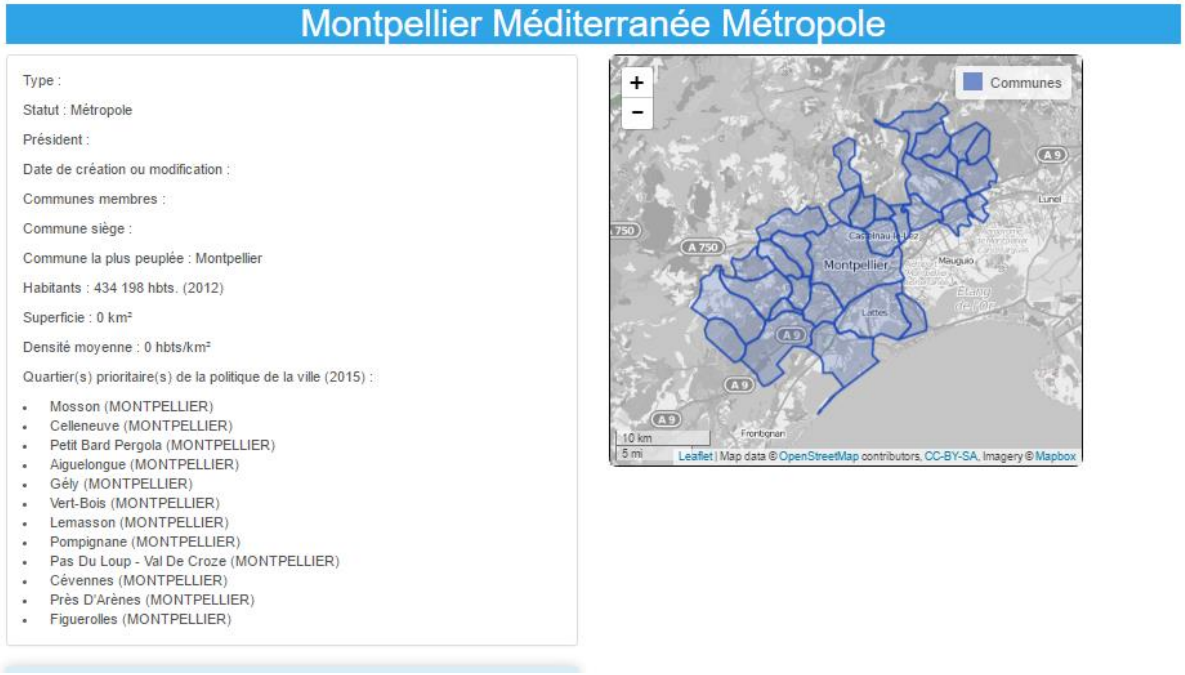
Nous avons également effectué cette opération pour le menu déroulant des départements où nous avons effectué une jointure avec la table équipement pour récupérer le nom du département.

```
SELECT distinct epci_commune_2017.dep_epci,
       equipement_2017.\"DepLib\"
FROM epci_commune_2017
INNER JOIN equipement_2017 ON epci_commune_2017.dep_epci =
equipement_2017.\"DepCode\"
ORDER BY \"dep_epci\";
```

Ces opérations nous ont permis de récupérer l'ensemble des départements de l'actuelle région Occitanie, mais aussi d'actualiser le nom des EPCI sur les listes déroulantes.

Ce genre d'opérations ont été effectuées pour le nom de l'EPCI placé en titre de la fiche, et pour le statut de l'EPCI, comme sur la figure ci-dessous.

Figure 8: Actualisation des données sur l'Occitanie (Midi Pyrénées et Languedoc Roussillon)



En revanche, la liste des quartiers politiques de la ville, la communes la plus peuplé, et le nombre d'habitants sont des procédures à part, n'utilisant pas la table compétences.

2. La cartographie

Création de la table Geojson

Pour la réalisation cartographique, nous avons dans un premier temps récupéré les données communales sur le sites IGN, où nous avons téléchargé la dernière base GEOFLA datant de 2016. Nous avons ensuite, sous le logiciel QGIS sélectionné la géométrie des communes de la région Occitanie, puis nous avons créé deux champs pour récupérer les coordonnées du centroïde en y entrant les formules suivantes :

x : x(centroid($geometry))

y : y(centroid($geometry))

Par la suite, une concaténation de ces deux champs a été effectuée afin de créer un champs centroid_inv. C'est également sous le logiciel QGIS que nous avons créé le champ géométrie GeoJson. Seul les champs code INSEE, géométrie et centroid_inv ont été conservé.

Requêtes sur le fichier index

```
    SELECT
                count(equipement_2017.\"ComInsee\"),
               com_geojson.geojson,
               epci_commune_2017.siren_epci,
               insee_com,
               epci_commune_2017.libgeo,
               commune_histo_2013.\"PMUN13\",
               centr_inv_2017.centr_inv
         FROM
              com_geojson
   Left JOIN epci_commune_2017 ON epci_commune_2017.codgeo = com_geojson.insee_com
   Left JOIN equipement_2017 ON com_geojson.insee_com =
equipement_2017.\"ComInsee\"
   LEFT JOIN commune_histo_2013 ON equipement_2017.\"ComInsee\" =
commune_histo_2013.\"DEPCOM\"
   Left JOIN centr_inv_2017 ON epci_commune_2017.siren_epci =
centr_inv_2017.epci_com_2
       WHERE
               epci_commune_2017.siren_epci = :code_epci
       GROUP BY
               com_geojson.geojson,
               epci_commune_2017.siren_epci,
               com_geojson.insee_com,
               epci_commune_2017.libgeo,
               commune_histo_2013.\"PMUN13\",
               centr_inv_2017.centr_inv;
```

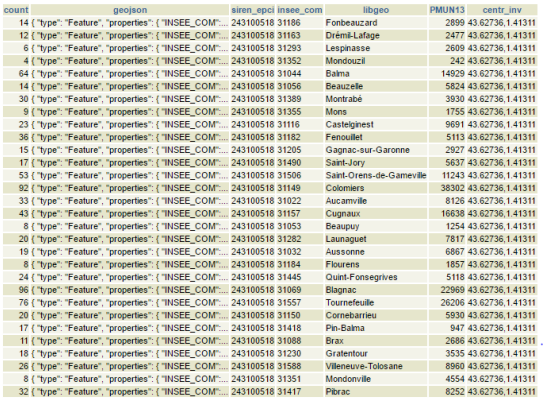
Afin de sélectionner les communes d'un EPCI, nous avons effectué une jointure entre la table geojson et la table epci_commune_2017 grâce au code INSEE de la commune. Les résultats de la requête ont ensuite été stocké sous forme de tableau dans la variable $params.

En cas de modification des EPCI, et de modification de la table epci_commune_2017, la requête SQL serait alors modifiée et l'affichage de la carte actualisé.

Stockage des informations

Ici nous stockons les informations de façon dynamique dans la base donnée PostGreSQL

Figure 9: Stockage des informations



count	geojson	siren_epci	insee_com	libgeo	PMUN13	centr_inv
14	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31186	Fonbeauzard	2899	43.62736,1.41311
12	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31163	Drémil-Lafage	2477	43.62736,1.41311
6	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31293	Lespinasse	2609	43.62736,1.41311
4	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31352	Mondouzil	242	43.62736,1.41311
64	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31044	Balma	14929	43.62736,1.41311
14	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31056	Beauzelle	5824	43.62736,1.41311
30	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31389	Montrabé	3930	43.62736,1.41311
9	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31355	Mons	1755	43.62736,1.41311
23	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31116	Castelginest	9691	43.62736,1.41311
36	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31182	Fenouillet	5113	43.62736,1.41311
15	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31205	Gagnac-sur-Garonne	2927	43.62736,1.41311
17	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31490	Saint-Jory	5637	43.62736,1.41311
53	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31506	Saint-Orens-de-Gameville	11243	43.62736,1.41311
92	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31149	Colomiers	38302	43.62736,1.41311
33	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31022	Aucamville	8126	43.62736,1.41311
43	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31157	Cugnaux	16638	43.62736,1.41311
8	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31053	Beaupuy	1254	43.62736,1.41311
20	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31282	Launaguet	7817	43.62736,1.41311
19	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31032	Aussonne	6867	43.62736,1.41311
8	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31184	Flourens	1857	43.62736,1.41311
24	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31445	Quint-Fonsegrives	5118	43.62736,1.41311
96	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31069	Blagnac	22969	43.62736,1.41311
76	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31557	Tournefeuille	26206	43.62736,1.41311
20	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31150	Cornebarrieu	5930	43.62736,1.41311
17	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31418	Pin-Balma	947	43.62736,1.41311
11	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31088	Brax	2686	43.62736,1.41311
18	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31230	Gratentour	3535	43.62736,1.41311
26	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31588	Villeneuve-Tolosane	8960	43.62736,1.41311
8	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31351	Mondonville	4554	43.62736,1.41311
32	{"type": "Feature", "properties": {"INSEE_COM":...	243100518	31417	Pibrac	8252	43.62736,1.41311

```
$Geo = array();
    foreach($rows as $row) {
        $les_c = array();
        $les_c['geojson'] =
htmlspecialchars_decode($row['geojson']);
        $les_c['centro_inv'] =
$row['centr_inv'];
        $les_c['equip'] = $row['count'];
        $les_c['nom'] = $row['libgeo'];
        $les_c['pop'] = $row['PMUN13'];
        array_push($Geo, $les_c)
    }
    $listgeojson = array();
      $listgeojson['listgeojson'] = $Geo;
    array_push($Geo, $listgeojson);
    $params = array_merge($params, $listgeojson);
    $params['centr_inv'] = $rows[0]['centr_inv'];
```

L'affiche de la carte se faisant dans la maquette nous avons ensuite effectué une boucle for de la longueur du tableau comme sur la figure ci-dessous :

```
$Geo = array();
    foreach($rows as $row) {
        $les_c = array();
        $les_c['geojson'] = htmlspecialchars_decode($row['geojson']);
        $les_c['centro_inv'] = $row['centr_inv'];
        $les_c['equip'] = $row['count'];
        $les_c['nom'] = $row['libgeo'];
        $les_c['pop'] = $row['PMUN13'];
        array_push($Geo, $les_c);
    }
    $listgeojson = array();
      $listgeojson['listgeojson'] = $Geo;
    array_push($Geo, $listgeojson);
    $params = array_merge($params, $listgeojson);
    $params['centr_inv'] = $rows[0]['centr_inv'];
```

Une fois l'affichage réussi, la carte était centrée sur le centroïde de la première EPCI, ce qui pouvait être dans certain cas problématique. Une nouvelle table centr_inv_2017 reprenant le centroïde des intercommunalités de 2017 a donc été créé, puis jointe dans la requête grâce au champ siren_epci. Même en cas de changement d'EPCI de quelques communes la valeur du centroïde resterait relativement juste.

Fenêtre pop-up

Pour l'affichage de la fenêtre pop-up, nous souhaitions afficher le nom de la commune, la population et le nombre d'équipements. Pour cela, nous avons privilégié une jointure avec la table équipement, rassemblant les équipements sportifs dans cette table le code équipement est la clé primaire, puis un count a été effectué par commune.

```
onEachFeature:  function (epci_json ,layer) {
 layer.bindPopup ('<font size=3><center><b>{{Geo.nom}}</b></center></font>
<i>Population:</i> {{Geo.pop}}<br> <i>Équipement(s):</i> {{Geo.equip}}'); } })
.addTo(mymap);
```

Pour l'affichage des populations, nous avons sélectionné le champs PMUN13 dans la table_histo_2013, enfin le nom de la commune provient du champ libgeo de la table epci_commune_2017. La partie du code qui permet l'affichage du pop-up avec les informations est ci-dessous.

Figure 10: Carte du site de base

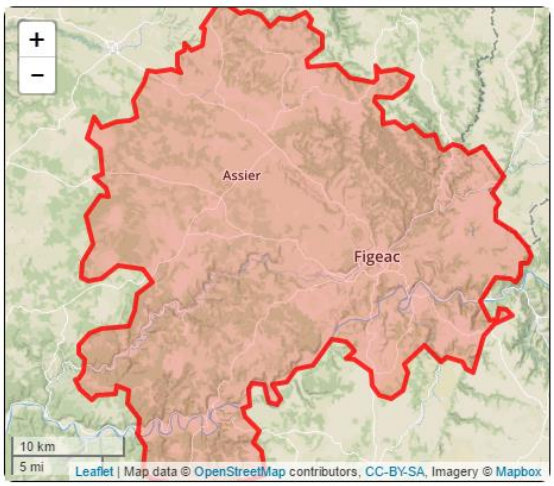
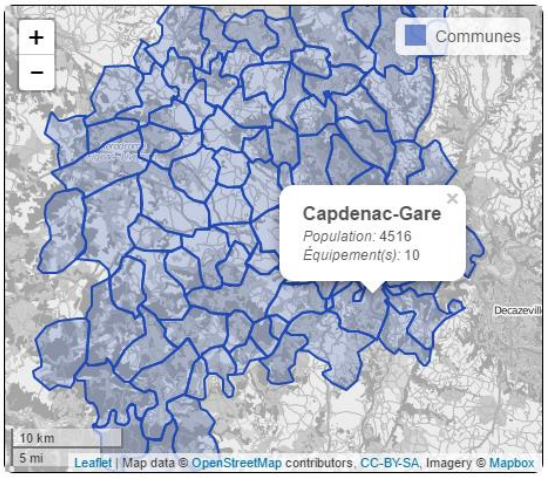


Figure 11: Carte interactive avec pop up



Ces nombreuses jointures nous permettent d'aller chercher les informations dans d'autres tables, et ne sont pas figés en cas de changement de table. Nous aurions pu stocker ces informations sur la table geojson, mais dans ce cas il aurait été impossible de faire évoluer ces valeurs.

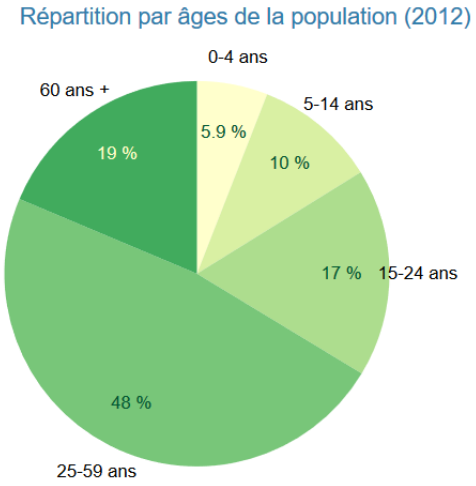
3. L'utilisation de la bibliothèque D3JS

L'histogramme

Pour mettre à jour le graphique de la population en 2012.

Ce graphique montre la répartition de la population en pourcentage par catégorie d'âge en 2012. Sur ce diagramme, nous n'avons pas d'information sur la répartition par sexe de la population. Certaines EPCI présentent des écarts importants sur la répartition des hommes et des femmes selon les tranches d'âge.

Figure 12: Ancien diagramme de la répartiton par âge de la population en 2012



C'est pour cela que nous avons choisi de mettre en place une pyramide des âges afin d'avoir la répartition des âges des hommes et des femmes par EPCI.

Mais cela implique de partir sur un nouveau code, nous avons donc choisi de partir de l'exemple:

http://jsfiddle.net/bahanson/UfWV3/

Tout d'abord, il est nécessaire d'importer les données mise à jour pour l'année 2013 dans l'index avec la requête suivante:

```
SELECT epci.siren_epci,
       sum(p.ageq_rec01s2rpop2013) AS pf0_4ans,
       sum(p.ageq_rec02s2rpop2013 + p.ageq_rec03s2rpop2013) AS pf5_14ans,
       sum(p.ageq_rec04s2rpop2013 + p.ageq_rec05s2rpop2013) AS pf15_24ans,
       sum(p.ageq_rec06s2rpop2013 + p.ageq_rec07s2rpop2013 +
p.ageq_rec08s2rpop2013 + p.ageq_rec09s2rpop2013 + p.ageq_rec10s2rpop2013 +
p.ageq_rec11s2rpop2013 + p.ageq_rec12s2rpop2013) AS pf25_59ans,
       sum(p.ageq_rec13s2rpop2013 + p.ageq_rec14s2rpop2013 +
p.ageq_rec15s2rpop2013 + p.ageq_rec16s2rpop2013 + p.ageq_rec17s2rpop2013 +
p.ageq_rec18s2rpop2013 + p.ageq_rec19s2rpop2013 + p.ageq_rec20s2rpop2013) AS
```

```
pf60ansplus,
       sum(p.ageq_rec01s1rpop2013) AS ph0_4ans,
       sum(p.ageq_rec02s1rpop2013 + p.ageq_rec03s1rpop2013) AS ph5_14ans,
       sum(p.ageq_rec04s1rpop2013 + p.ageq_rec05s1rpop2013) AS ph15_24ans,
       sum(p.ageq_rec06s1rpop2013 + p.ageq_rec07s1rpop2013 +
p.ageq_rec08s1rpop2013 + p.ageq_rec09s1rpop2013 + p.ageq_rec10s1rpop2013 +
p.ageq_rec11s1rpop2013 + p.ageq_rec12s1rpop2013) AS ph25_59ans,
       sum(p.ageq_rec13s1rpop2013 + p.ageq_rec14s1rpop2013 +
p.ageq_rec15s1rpop2013 + p.ageq_rec16s1rpop2013 + p.ageq_rec17s1rpop2013 +
p.ageq_rec18s1rpop2013 + p.ageq_rec19s1rpop2013 + p.ageq_rec20s1rpop2013) AS
ph60ansplus
FROM \"pop_sexe_age_quinquennal_2013\" as p,
       epci_commune_2017 as epci
WHERE epci.siren_epci = :code_epci AND
       epci.codgeo = p.code_insee
GROUP BY epci.siren_epci;
```

Ainsi cette requête permet de récupérer les données par sexe et par classe d'âge indispensable à la création de la pyramide des âges.

Il faut ensuite importer les données dans la maquette:

```
var exampleData = [
  {"group": "0-4", "male": "{{ph0_4ans_b}}", "female": "{{pf0_4ans_b}}"},
  {"group": "5-14", "male": "{{ph5_14ans_b}}", "female": "{{pf5_14ans_b}}"},
  {"group": "15-24", "male": "{{ph15_24ans_b}}", "female": "{{pf15_24ans_b}}"},
  {"group": "25-59", "male": "{{ph25_59ans_b}}", "female": "{{pf25_59ans_b}}"},
  {"group": "60+", "male": "{{ph60ansplus_b}}", "female": "{{pf60ansplus_b}}"},
]
    gdata = exampleData;
```

Nous avons ensuite récupéré la valeur maximum pour chaque sexe permettant ensuite de dessiner les barres de l'histogramme. Par l'indicateur "parseInt" on force la variable "male" qui était stockée en texte à être transformé en entier pour pouvoir définir un ordre de grandeur.

```
var maxValue = Math.max(
  d3.max(exampleData, function(d) { return parseInt(d.male); }),
  d3.max(exampleData, function(d) { return parseInt(d.female); })
);
```

Nous avons ensuite déterminé les axes en "x" et en "y" à partir de la valeur maximum pour chaque sexe.

```
var xScale = d3.scale.linear()
  .domain([0, maxValue])
  .range([0, regionWidth])
  .nice();

var xScaleLeft = d3.scale.linear()
  .domain([0, maxValue])
  .range([regionWidth, 0]);

var xScaleRight = d3.scale.linear()
  .domain([0, maxValue])
  .range([0, regionWidth]);


var yScale = d3.scale.ordinal()
  .domain(exampleData.map(function(d) { return d.group; }))
  .rangeRoundBands([h,0], 0.05);
```

Nous avons ensuite configuré les titres, les axes. Nous allons maintenant voir plus en détail la construction des barres de l'histogramme avec l'exemple des valeurs pour les femmes.

Cette première partie consiste en la création graphique des barres, on récupère ici les valeurs maximum pour les femmes établies préalablement correspondant dans le code à *"xScale(d.female)"* et *"yScale(d.group)".*

```
rightBarGroup.selectAll('.bar.right')
  .data(exampleData)
  .enter().append('rect')
       .attr('class', 'bar right')
       .attr('x', 0)
       .attr('y', function(d) { return yScale(d.group); })
       .attr('width', function(d) { return xScale(d.female); })
       .attr('height', yScale.rangeBand())
```

La fonction *"mouseover"* permet de contrôler les évènements qui se passe sur les barres lors du passage de la souris. Ainsi lors du passage de la souris le contour de la barre se met en gris clair.

```
    .on("mouseover", function(d) {
                    console.log(d);
                    d3.select(this)
                    .style("stroke", "gray");
```

Il a fallu ensuite configurer les éléments de texte à ajouter qui s'afficheront au passage de la souris, nous avons donc choisi d'y écrire la valeur correspondant à la barre. On gère également le style du texte: couleur, position...

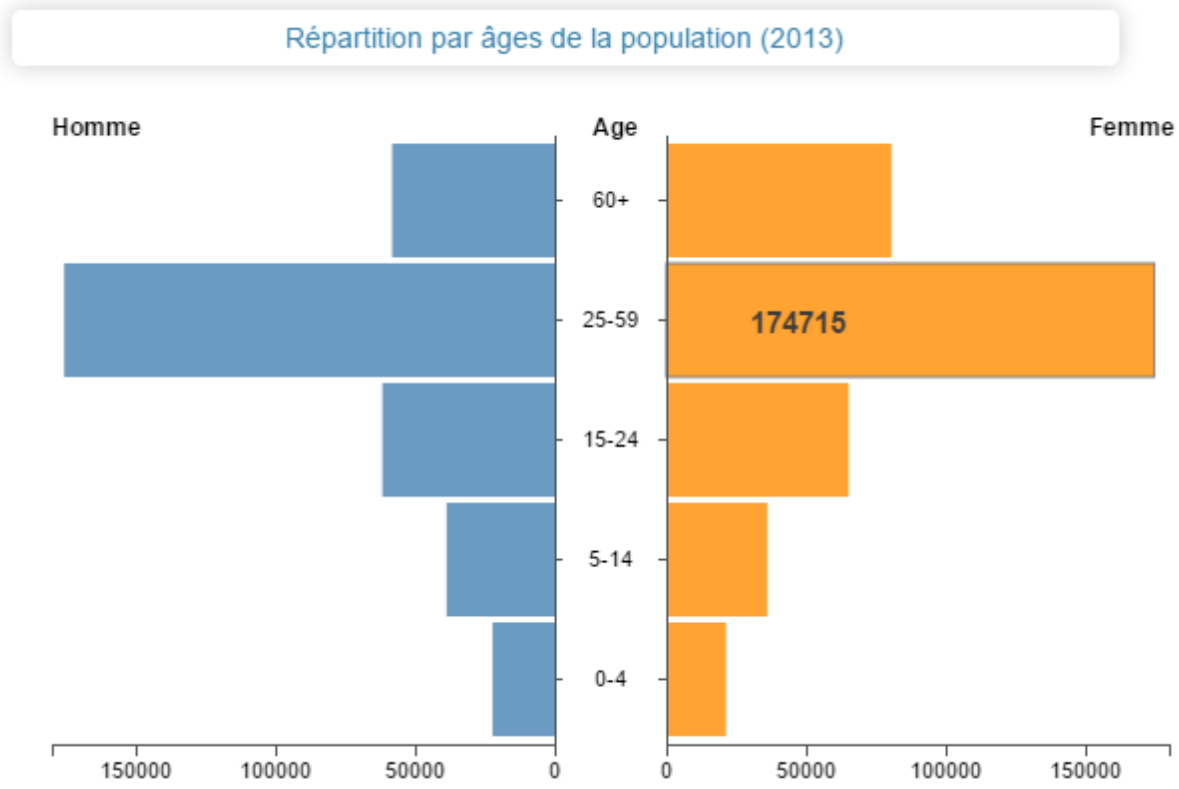
```
svg.append("text")
       .attr("y",yScale(d.group)+35)
       .attr("x", 350)
       .style("font-size", 20)
       .attr("class","label")
       .style("fill", function(d){return "#3c3c3c";})
       .text(function() {
              return (d.female); })
                     })
```

La fonction “*mouseout*” gère la fin de l'événement du passage de la souris, on demande donc que les évènements présents soient arrêtés.

```
.on("mouseout", function() {
svg.select(".label").remove();
d3.select(this).style("stroke", "none");
});
```

Voici le résultat de la représentation graphique de l’histogramme.

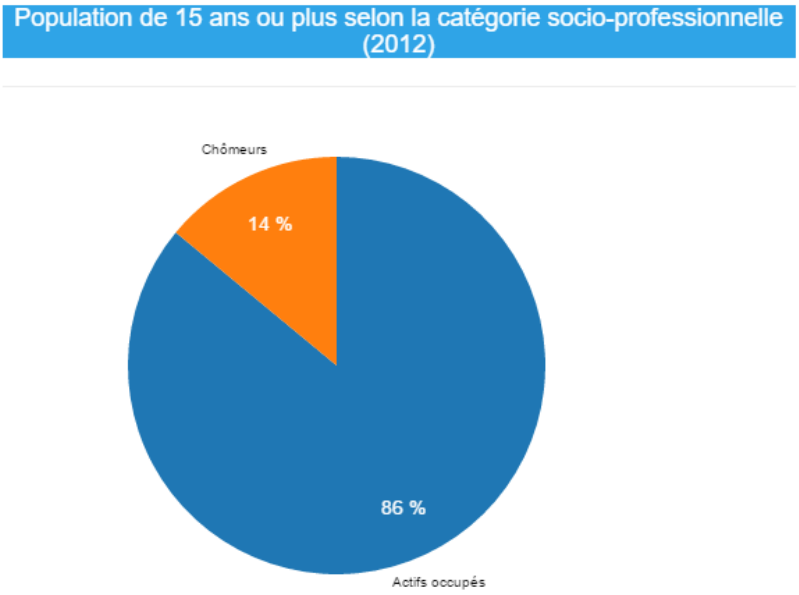
Figure 13: Histogramme de la répartition par âges de la population en 2013



Le treemap

Le “Treemap” ou carte proportionnelle permet de représenter les données de façon hiérarchiques. Il a été inventé par Ben Shneiderman et repris en langage D3JS par Mike Bostock (https://bl.ocks.org/mbostock/4063582) cependant nous avons utilisé un modèle plus simple http://bl.ocks.org/ganeshv/6a8e9ada3ab7f2d88022 . Nous nous sommes ainsi basé sur ce modèle car la construction du fichier json qui est associé au Treemap est la plus facile à mettre en application.

Figure 14: Ancien diagramme de la répartition de la population de 15ans ou plus selon la catégorie socioprofessionnelle en 2012



	2012	%
Agriculteurs exploitants	421	0 %
Artisans, commerçants, chefs d'ent.	17250	5 %
Cadres et prof. intel. supérieures	97419	26 %
Professions intermédiaires	102959	28 %
Employés	94149	25 %
Ouvriers	53710	14 %

Pour améliorer le graphique de la population de 15 ans ou plus selon la catégorie socioprofessionnelle (2012), nous avons décidé d'aller plus loin qu’une opposition entre chômeurs et actifs occupés.

Nous avons trouvé intéressant de pouvoir ajouter la répartition par catégorie socioprofessionnelle au sein du même graphique.

Au vu de ces éléments l’approche du treemap, nous a paru approprié, nous permettant d’avoir une information à deux échelles.

Dans un premier temps, nous avons donc mis en forme les données dans l'index sous forme de fichier json.

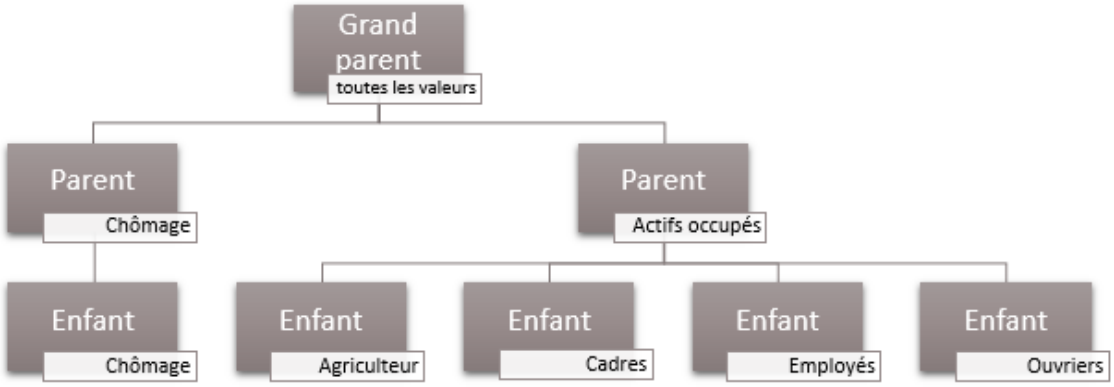
```
$a='[';
        $a .='{"key": "Chômage", "region": "Chômage", "value":';
        $a .= round($rows[0]["chom1564"] * 100 / $rows[0]["popact1564"]);
        $a .='},';
        $a .='{"key": "Agriculteurs", "region": "Actifs occupées", "value":';
        $a .= round($rows[0]['act1564cs1'] * 100 / $rows[0]['popact1564']);
        $a .='},';
        $a .='{"key": "Artisans", "region": "Actifs occupées", "value":';
        $a .= round($rows[0]['act1564cs2'] * 100 / $rows[0]['popact1564']);
        $a .='},';
        $a .='{"key": "Cadres", "region": "Actifs occupées", "value":';
        $a .= round($rows[0]['act1564cs3'] * 100 / $rows[0]['popact1564']);
        $a .='},';
        $a .='{"key": "Professions intermédiaires", "region": "Actifs occupées",
"value":';
        $a .= round($rows[0]['act1564cs4'] * 100 / $rows[0]['popact1564']);
        $a .='},';
        $a .='{"key": "Employés", "region": "Actifs occupées", "value":';
        $a .= round($rows[0]['act1564cs5'] * 100 / $rows[0]['popact1564']);
        $a .='},';
        $a .='{"key": "Ouvriers", "region": "Actifs occupées", "value":';
        $a .= round($rows[0]['act1564cs6'] * 100 / $rows[0]['popact1564']);
        $a .='}';
        $a .=']';
        $params['metier'] = $a;
```

Pour être formaté sous la forme suivante:

```
[metier] => [{"key": "Chômage", "region": "Chômage", "value":15},
{"key": "Agriculteurs", "region": "Actifs occupées", "value":0},
{"key": "Artisans", "region": "Actifs occupées", "value":5},
{"key": "Cadres", "region": "Actifs occupées", "value":27},
{"key": "Professions intermédiaires", "region": "Actifs occupées", "value":28},
{"key": "Employés", "region": "Actifs occupées", "value":25},
{"key": "Ouvriers", "region": "Actifs occupées", "value":14}]
```

Ainsi avec les valeurs sous cette forme, stockées dans la variable "*metier*" du tableau $params, nous allons pouvoir y faire appel dans la maquette. Le code pour la réalisation du treemap est long et complexe, nous avons décidé de ne pas le détailler mais nous allons néanmoins expliquer la logique de création du treemap. Le treemap se développe comme un arbre généalogique dans notre cas, il est composé de grand parent, de parent et d'enfant qui correspondent à un niveau hiérarchique des variables.

Figure 15: Organisation du treemap et des valeurs



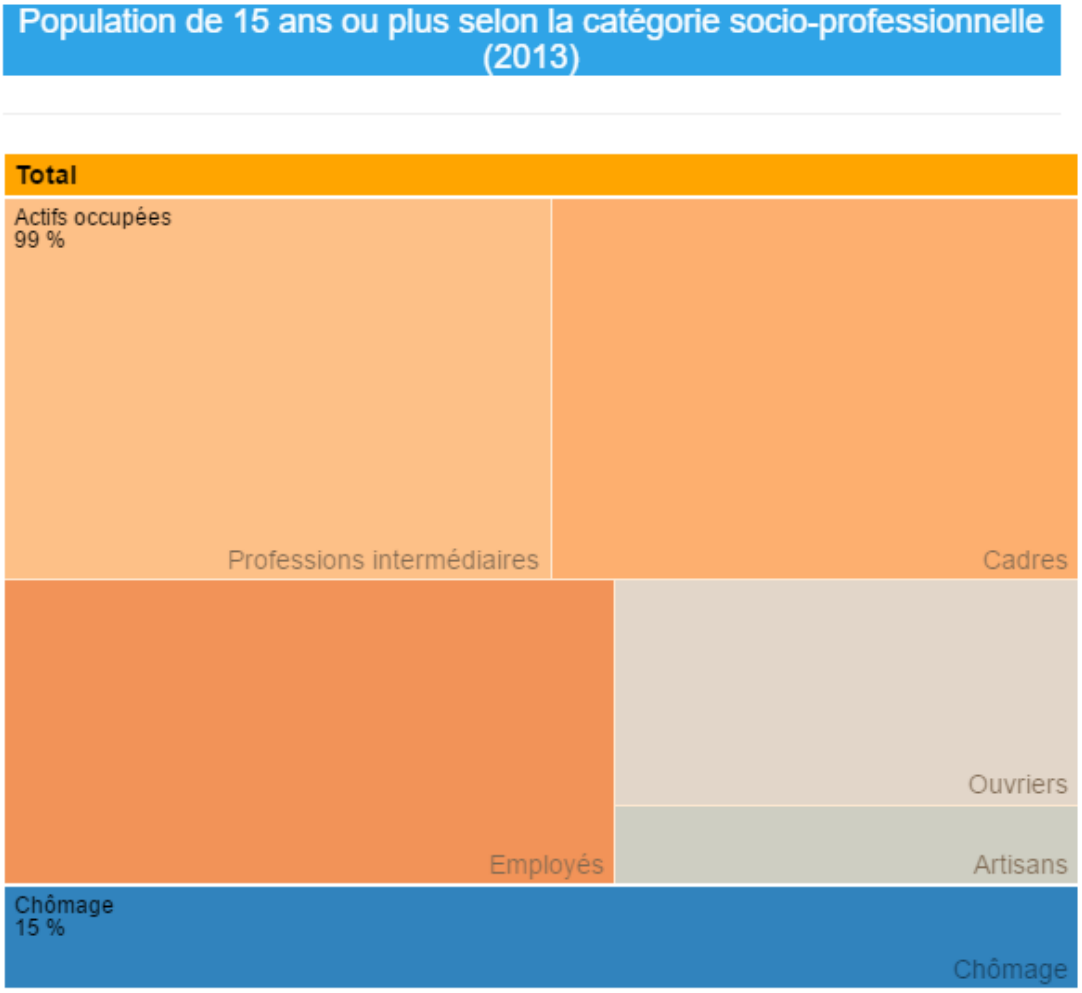
Cela permet d'avoir une organisation rationnelle des données et un traitement proportionnel des variables selon sa situation.

Pour importer dans la maquette les variables stockées dans la variable [metier] du $params, les données sont importées en brut et nous attribuons la variable [metier] à la variable data qui alimente tout le treemap.

```
var source = {{metier|raw}};
      var data =d3.nest().key(function(d) { return d.region; })
                  .entries(source);
      main({title: ""}, {key: "Total", values: data});
```

Voici le résultat du Treemap où si l'on clique sur "*actifs occupés*", on a le détail en pourcentage par catégorie socioprofessionnelle.

Figure 16: Treemap des catégories socioprofessionnelles



Les diagrammes donuts

Nous allons décrire ci-dessous les principales étapes réalisés pour concevoir les graphiques visibles sur le prototype web.

Les lignes de codes JavaScript ci-dessous permettent dans un premier temps de configurer l'emprise de la fenêtre graphique, ainsi que les couleurs :

```
var width=400;
var height=500;
var radius=160;
var color =
d3.scale.ordinal().range(["#ffd699","#ffad33","#ff9900","#cc7a00","#995c00","
#4d2e00"]);
 var svg = d3.select("#graphe_vrpes")
.append("svg")
.attr("width", width )
.attr("height", height)
.append("g")
.attr("transform", "translate(" + (width / 2) + ',' + (height / 2) + ')');
```

La forme de type donuts est ensuite configurée par les lignes de code suivantes, permettant de définir le rayon interne vide (innerRadius) et externe (outerRadius).

```
var donutWidth =50;
var arc = d3.svg.arc()
       .innerRadius(radius - donutWidth)
       .outerRadius(radius);
```

Une requête SQL dans le fichier index est préalablement stockée sous forme de tableau. Ces données sont ensuite récupérées dans le fichier maquette où sont dessinés les graphiques, par la procédure ci-dessous.

```
var data = [ {% for vrpe in vrpes %}
{"type":"{{vrpe.nom_com}}", "valeur":"{{vrpe.pnb_equ}}"},
{% endfor %}   ]
gdata = data;
```

Tableau de données dans le $params

```
[vrpes] => Array
        (
            [0] => Array
                (
                    [nom_com] => Toulouse
                    [pnb_equ] => 52.5
                )

            [1] => Array
                (
                    [nom_com] => Blagnac
                    [pnb_equ] => 4.5
                )

            [2] => Array
                (
                    [nom_com] => Colomiers
                    [pnb_equ] => 4.3
                )

            [3] => Array
                (
                    [nom_com] => Autres
                    [pnb_equ] => 38.7
                )

        )
```

Les couleurs du graphique sont ensuite incorporés par la procédure suivante

```
var g = svg.selectAll(".arc")
 .data(pie(data))
.enter()
.append("g")
  .attr("class", "arc");
g.append("path")
       .attr("d", arc)
       .style("fill", function(d) {
       return color(d.data.type); })
```

Afin de rendre les graphiques dynamiques nous avons rajouté les fonctions mouseover et mouseout utilisées pour l'affichage des pourcentages. Une bordure a également été ajoutée sur la partie du diagramme où l'utilisateur passe la flèche de navigation la console.log est uniquement utilisée pour faciliter la programmation du graphique.

```
.on("mouseover", function(d) {
console.log(d);
d3.select(this).style("stroke", "white");

eqs.append("text")
.attr("y",90)
.attr("x",-30)
.style("font-size", 28)
.style("opacity", 0.6)
.attr("class","label")
.style("fill", function(d){return "#2980B9";})
.text(function() {var a = d.data.valeur;if (a == 0) {a = 0.01;}return pr(a) + " %";
})}
.on("mouseout",              function()              {eqs.select(".label").remove();
d3.select(this).style("stroke", "none");});
```

Les lignes de code ci-dessous permettent de créer, configurer la couleur, la taille et la position de la légende.

```
var legendRectSize = 18;
var legendSpacing = 4;

var legend = eqs.selectAll(".legend")
       .data(color.domain())
       .enter()
       .append("g")
       .attr("class", "legend")
       .attr("transform", function(d, i) {
var height = legendRectSize + legendSpacing;
var offset =  height * color.domain().length / 2;
var horz = -5 * legendRectSize;
var vert = i * height - offset;
       return 'translate(' + horz + ',' + vert + ')';});
```

```
legend.append("rect")
       .attr("width", legendRectSize)
       .attr("height", legendRectSize)
       .style("fill", color)
       .style("stroke", color);

legend.append("text")
.attr("x", legendRectSize + legendSpacing)
.attr("y", legendRectSize - legendSpacing)
.text(function(d) { return d; });
```

Équipements sportifs (2015)

Ces graphiques montrent la répartition des équipements sportifs sur les trois communes principales des EPCI, cela permet d'avoir une estimation du poids en équipement de ces communes principales.

Figure 17: Ancien diagramme des équipements sportifs en 2015

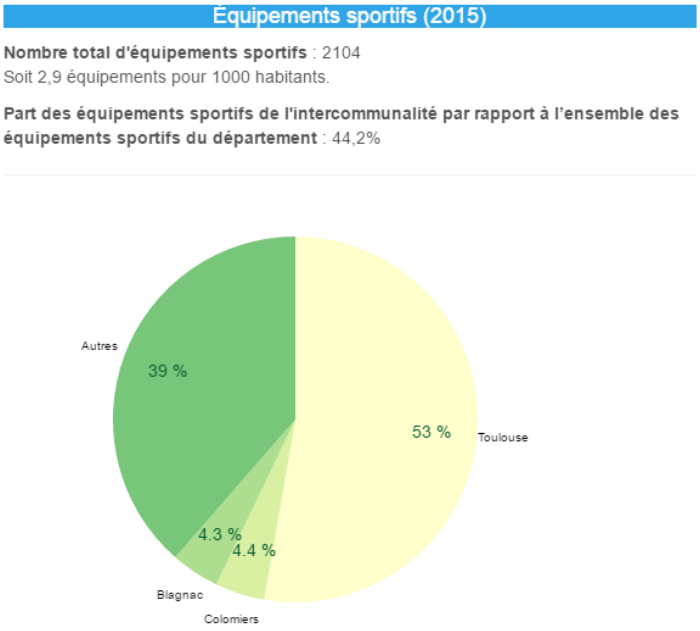
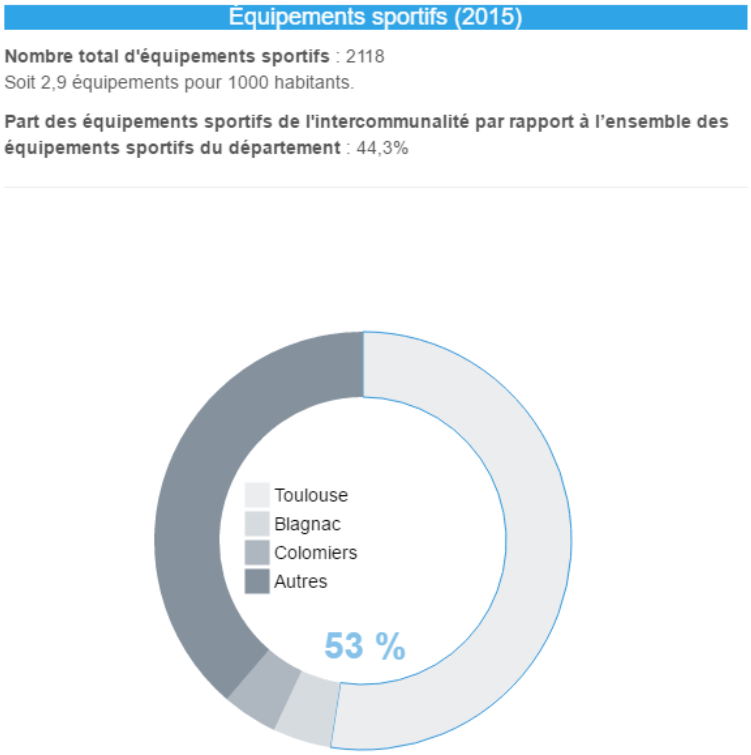


Figure 18: Diagramme donuts des équipements sportifs en 2015



Mise en service des équipements sportifs (2015)

Ce graphique montre la mise en service des équipements sportifs sur différentes périodes, pour avoir une estimation de l'âge des équipements par EPCI et cela donne une idée des maintenances ou travaux à venir.

Figure 19: Ancien diagramme de la mise en service des équipements sportifs en 2015

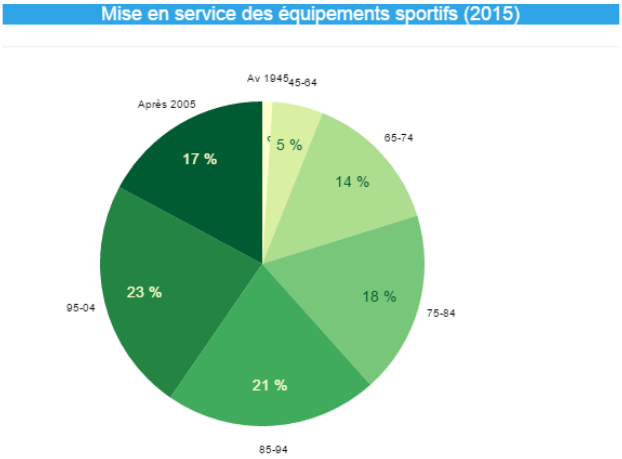
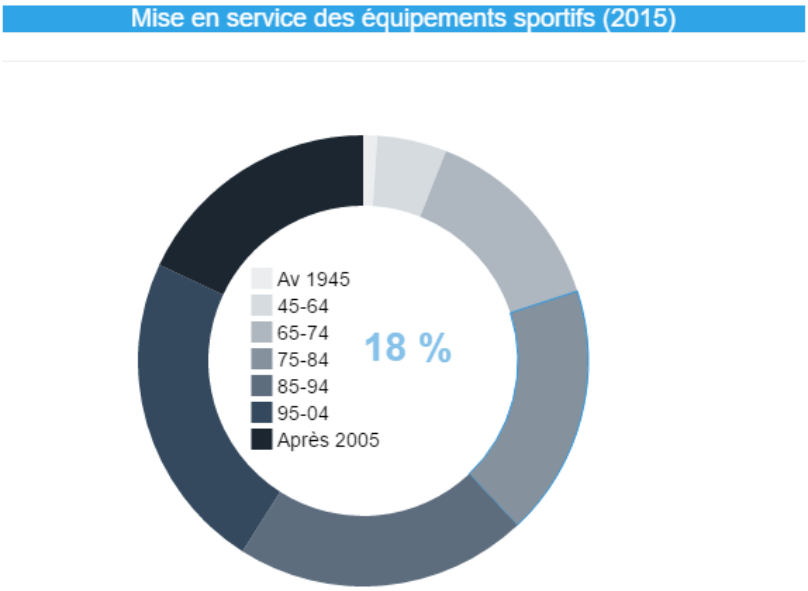


Figure 20: Diagramme donuts de la mise en service des équipements sportifs en 2015



Part des équipements dans lesquels ont été réalisés des travaux depuis 1995: 21,4%

Cette nouvelle fonctionnalité permet une lecture de l'information plus claire. Ci-dessous nous pouvons voir qu'il était impossible de lire les petites valeurs comme le 1% des équipements construit avant 1945. Le nouveau graphique dynamique permet une lecture claire de l'information avec une légende rajoutée et reprenant le code couleur du site.

Figure 21: Zoom sur la répartition des données

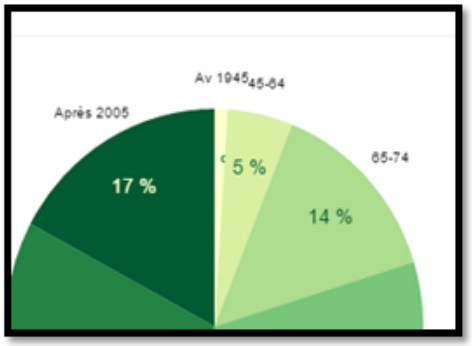
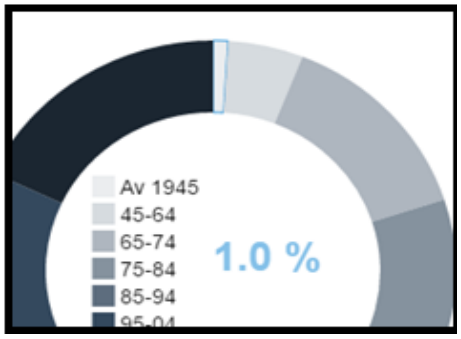


Figure 22: Zoom sur la répartition des données du diagramme donuts



4. Discussions et limites

Pour la cartographie, nos objectifs fixés ont donc été remplis, en effet nous souhaitions d'une part un affichage des communes de l'EPCI, mais aussi des pop-up par communes, apportant un degré d'information supplémentaire que nous n'avions pas auparavant. De plus ces informations ne sont pas stockées en dur, elles proviennent d'autres tables et peuvent donc être actualisées en cas de changement de table. Il en est de même avec l'affichage des communes par EPCI, en cas de changement d'EPCI d'une commune cette dernière y sera instantanément rattachée en cas de changement de la table regroupant la composition communale des EPCI. En revanche en cas de fusion de communes où de modification du contour communal, le chargement d'une nouvelle table des géométries est indispensable.

5. Conclusion

Pour conclure, ce projet nous a permis d'approfondir et de mettre en pratique des connaissances acquises en cours de programmation web et de gestion de bases données. Concernant l'aspect visuel du site, nous retrouvons sur la carte un degré d'informations supplémentaire au niveau communal avec les fenêtres pop-up affichant la population et le nombre d'équipements à cette échelle-là.

Nous avons également modifié les graphiques afin de les rendre dynamiques et interactifs. Le premier était dans l'ancienne version du site un diagramme en camembert représentant les catégories d'âges de l'EPCI, la nouvelle pyramide des âges permet d'avoir cette même répartition par sexe. Le graphique de type treemaps plus compliqué à programmer permet d'avoir dans un premier niveau la répartition entre actifs occupés et chômeurs, puis dans un deuxième niveau la répartition des PCS (Professions et Catégories Socioprofessionnelles). Enfin les deux graphiques de types donuts représentant l'âges des équipements et leurs répartitions par commune apportent une lisibilité supérieure des petites valeurs ainsi qu'un affichage dynamique.

Table des matières