

# Manuel d'utilisation du plugin Module d'Export.

## Table des matières

Rappel succinct du cahier des charges.....	4
Technologies utilisées.....	5
Possibilités et limites.....	8
Déployer le module d'export.....	9
Coté client.....	9
Installer le plugin.....	9
Coté serveur.....	10
A/ Préparer la base de donnée.....	10
B/ Implémenter les fichiers .bat.....	11
Exemple d'utilisation du module d'export (modèle INPN).....	13
A/ Préparatifs.....	13
1/ Importer sa couche z_export.....	13
2/ Sélectionner sa surface de travail.....	13
L'utilisateur peut alors créer l'entité de son choix, surface sur laquelle portera l'export. Il est bien sûr possible et encouragé de faire un copier-coller d'une entité d'une autre couche sur z_export. Chaque entité dispose d'un ID, ainsi qu'un libellé qui apparaîtra dans la fiche d'export. Il peut donc être intéressant de reprendre le nom d'un site ou d'un lieu dit dans l'entité de z_export.....	13
Il faut ensuite sélectionner la ou les entités sur lesquelles porteront l'export. L'utilisateur peut ensuite lancer le <i>plugin</i> . Si aucune entité n'est sélectionnée ou si une autre couche que z_export a le focus, une structure de contrôle renverra un message d'erreur.....	13
B/ Rentrer ses paramètres d'export.....	14
Une fois le plugin lancé, l'utilisateur reçoit cette fenêtre de dialogue avec plusieurs critères à renseigner.....	14
a) Choix de destinataire.....	14
b) Choisir la date sur laquelle porte l'export.....	14
c) Choisir les paramètres de moissonnage.....	16
d) Choisir les espèces à prendre en compte.....	16
c) Choix du type d'export.....	20
d) Les fonctions relatives à l'export.....	21
C/ Identifier le résultat de son export.....	21
a) Le fichier de métadonnée.....	22
b) Des fichiers CSV.....	22
c) Des fichiers shapefile.....	23
D/ Contrôle supplémentaires.....	23
Schéma de fonctionnement.....	25
Simplifié.....	25
Tables utilisées.....	26
Comprendre le rôle des tables des schéma bdcenpicardie et bd_site_cen.....	28
Structure des fonctions sur schéma d'export.....	30
1/_1_Recupdesid.....	30
a) Récupérer les entités référencées géographiquement.....	30

b) Récupérer les entités qui ne sont pas directement référencées géographiquement.....	31
2/ 2_filtrage.....	34
a) Créer une table de suivi du filtrage des éléments.....	35
b) Faire les tris sur les espèces.....	36
3/ Export INPN.....	41
a) Remodeler la donnée pour générer St_Principal.....	43
b) Générer et exporter la table EN.....	64
c) Générer une table d'attributs supplémentaires.....	65
Kill_table().....	71
Structure du code python du module d'export.....	79
Libraires utilisées.....	79
Organisation du code du plugin.....	80
Initialiser le plugin.....	82
Dynamisme de la fenêtre principale.....	85
Ajouter des espèces.....	87
Fonction de recherche :.....	91
Sélectionner des espèces.....	92
Supprimer des espèces de la sélection.....	94
Ajouter les espèces dans la base de donnée.....	95
Assurer une fermeture propre de la fenêtre.....	96
Nettoyer les tables et fichiers.....	96
Exporter.....	97
Contrôle de cohérence du paramètre de dates.....	97
Informé l'utilisateur de l'évolution de l'export.....	97
première fonction : Enclencher le filtrage géographique.....	98
Deuxième fonction : enclencher le filtrage attributaire.....	98
Troisième fonction : Sélection du type d'export et Génération des tables temporaires et CSV.....	100
Quatrième fonction : Exécuter les fichiers .bat pour générer des shapefiles.....	101
Cinquième fonction : génération de la fiche de métadonnée.....	102
Code Python du module d'export.....	103
Classe principale.....	103
Classe de Dialogue de ModulExport :.....	124
Classe de dialogue de la Faune.....	126



## Rappel succinct du cahier des charges.

Lors de la migration d'ArcGis vers Qgis, et de MySQL vers PostgreSQL, le CenPic ambitionne de se diriger vers un outil plus performant tout en conservant les acquis de l'ancien. Un module d'export avait en effet déjà été développé sur une interface Access, mais dans ce contexte de migration, un portage sur Qgis en Qt du module permettrait une meilleure intégration de l'outil dans le nouvel environnement. Contrairement au connecteur où il s'agissait d'un simple portage, pour le module d'export, un travail de réflexion a lieu pour structurer les exports.

### Le module a pour principaux objectifs :

- **Pouvoir exporter au format INPN. Cela inclus la génération :**
  - *De fichiers CSV.*
  - *De fichiers ShapeFile.*
  - *D'une fiche de Métadonnée.*
- **Pouvoir passer une série de critères dans une interface conviviale :**
  - *Informations relatives au destinataire*
  - *Informations temporelles : saisie et Observation.*
  - *Validité, présence de la donnée. Est-elle aussi interne ?*
- **Pouvoir sélectionner des espèces.**
  - *Selon des critères différents : taxon, classe, ordre...*
  - *Pouvoir sélectionner toute la flore ou toute la faune.*
- **Choisir un type d'export.**
  - Un format d'export brut de test.*
  - *Un format d'export INPN.*

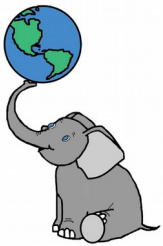
Le module ambitionne également d'être évolutif, sans avoir à retoucher au code Python, pour faire des exports de faune et flore portant sur des modèles différents. Donc si l'outil doit être intuitif, il ne faut pas oublier qu'il est à destination de géomaticiens, qui sont à même de générer des apports avec du code : l'outil doit donc combiner facilité d'utilisation et souplesse.

## Technologies utilisées.

Le choix des technologies utilisées a bien sûr été fait pour tirer parti au maximum de la migration. Celle-ci implique *De Facto* davantage d'outils libres.



### PostgreSQL / PostGis.



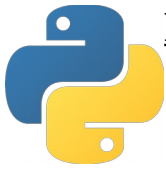
L'écrasante majorité des calculs est effectuée par le logiciel de SGBD PostgreSQL couplé avec son extension géographique Postgis. Le CenPic bénéficie ainsi d'un outil libre en constant développement, qui est puissant, très stable et à la fois souple avec un système d'extensions et de conversions de types à la volée. PostgreSQL bénéficie également d'un langage interne, le pgpsql qui permet de faire des boucles et des fonctions poussées.

Ce SGBD est mis au centre du fonctionnement du plugin. Pour aller plus loin, une grande partie des exports peuvent être possibles simplement en appelant des fonctions.



### QGIS 2.x

Le choix de faire un plugin pour le module d'export a été retenu pour plusieurs raisons. A la fois pour avoir un outil intégré dans Qgis et donc cohérent dans l'écosystème du CenPic. Mais aussi pour pouvoir sélectionner directement des entités dans le canevas de Qgis, sur lesquelles porteront l'export. Ce système permet de faire des exports précis réalisés de façon intuitive. On se passe aussi complètement d'Access.



## Python 2.7

Les plugins sous Qgis peuvent être développés en C++ ou en Python, un langage sous licence GPL. Ce dernier a été retenu pour sa facilité d'utilisation, de lecture et de déploiement. Puisque le plugin est déployé en environnement Windows, l'interpréteur est toujours celui de Qgis. Un effort est donc fait pour utiliser exclusivement les libraires présentes dans le Path de l'interpréteur Qgis. La version utilisée est ainsi la 2.7, qui sera d'actualité jusqu'à la sortie de Qgis 3.0.



## QT 4

QT est une librairie graphique puissante et multi-plateforme, sous licence GNU/GPL, d'où sont adoption par le projet Qgis. On peut notamment l'utiliser avec C++ et python. Cette librairie dépasse largement le simple affichage de donnée avec des fonctionnalités qui permettent de directement se connecter à une base de donnée. Un outil graphique permet de dessiner de façon intuitive ses interfaces puis de les connecter à son code : QT Designer. Une version spécifique à Qgis a été utilisé pour développer le *plugin*.



Le format de fichier d'échange standard, qui se présente comme un langage à balises. Dans le plugin, le fichier est automatiquement généré avec Python et sa librairie Dom. L'intérêt d'utiliser un fichier plutôt que de faire des tables est sa facilité d'échange, d'éviter de charger la base inutilement, tout en ayant une structure capable d'être interrogé grâce un langage de requêtes.



## **Batch Windows.**

Dans la mesure où un modèle évolutif, indépendant du script Python a été décidé, la génération de ShapeFiles devait se faire ailleurs que dans le corps du script. Un système de scripts lancés dans le langage de script Windows sous formes de fichiers .bat permet cette émancipation. Dans les faits, le Batch n'est utilisé que pour appeler OGR2OGR.



## **OGR2OGR**

OGR2OGR est une fonction de GDAL que l'on appelle sur Windows avec les scripts .bat. L'intérêt de l'outil est de convertir des fichiers géographiques d'un format à un autre, notamment pour des bases de données. Pour des conversions spécifiques vers du Shape, la documentation recommande d'utiliser pgsq2shape. Nous resterons pourtant sur OGR2OGR car c'est un outil qui est connu au conservatoire, et qui permet d'exporter vers d'autres format que du fichier de formes exclusivement.

## Possibilités et limites

Cette assemblage de technologies laisse entrevoir la possibilité d'évolution du programme avec la possibilité d'ajout de fonctions sans toucher à Python. En altérant le code Python, on peut aussi ajouter des critères, mais un vrai travail développement sera nécessaire. Le format base de donnée permet également d'insérer le choix des espèces à traiter par une requête SQL plutôt que passer par le plugin.

En revanche, le plugin sans d'importantes modifications, ne peut s'appliquer qu'aux besoins du CenPic du faite qu'il s'adapte à la structure de la base de donnée de Qgis, et que les fichiers .bat sont spécifiques à l'écosystème de Microsoft.



# Déployer le module d'export.

Dans la mesure où le module d'export a été déployé pour les besoins précis du conservatoire, son déploiement tel quel ne peut se faire que dans cette structure.

## Coté client.

Pour pouvoir utiliser le plugin, l'utilisateur doit avoir une connexion en utilisateur PostgreSQL dans Qgis.

## Installer le plugin

### 1/ Copier

L'export peut se lancer du coté client comme du coté serveur, il suffit d'avoir une installation Qgis compatible avec le plugin (développé en 2.14) . il faut ensuite chercher le plugin sur le serveur dans le répertoire suivant : [S:\00\\_MODULE\\_EXPORT\\_BDCEN](#) , puis le copier dans son répertoire local de plugins Qgis. Selon le type d'installation choisi, le répertoire peut-être soit :

- dans `C:\Users\NOM_UTILISATEUR\.qgis2\python\plugins`
- où dans le dossier programmes.

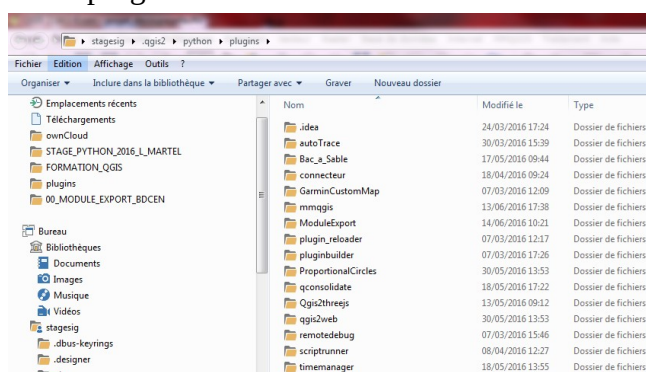


Illustration 1: Exemple de dossier de plugin

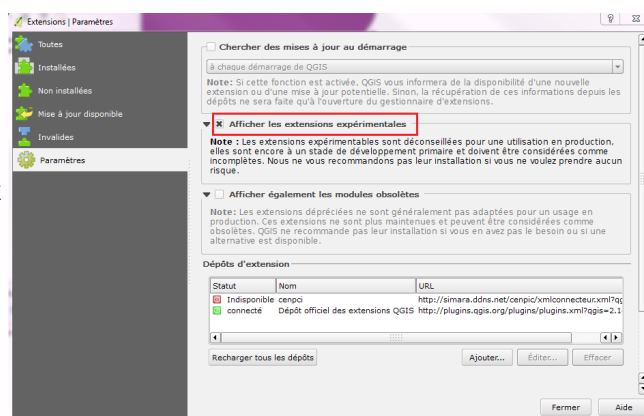
Dans l'illustration ci-dessus, on aperçoit bien le *plugin* de Module d'Export, sous la forme d'un répertoire, comme les autres *plugins*.

### 2/Activer

L'utilisateur lance ensuite Qgis, et doit activer manuellement l'extension en cliquant sur :

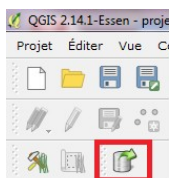
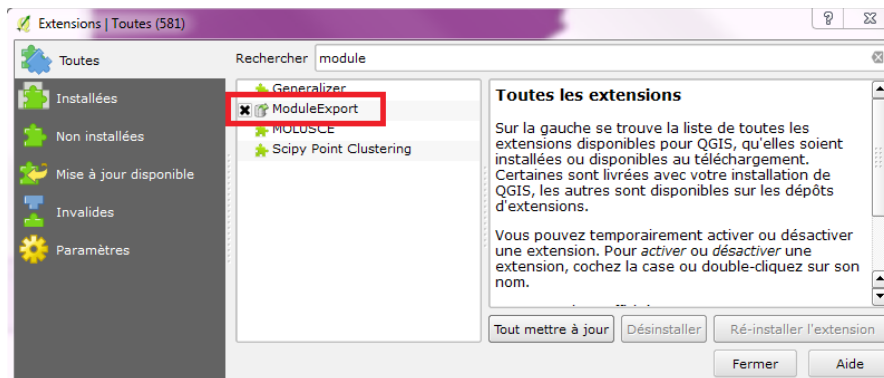
extension > installer/gérer les extensions.

Puis dans le menu paramètre, il faut cocher l'affichage des extensions expérimentales. Cette opération est une précaution pour bien



pouvoir afficher le plugin.

On active ensuite l'extension dans le menu « toutes », en tapant par exemple module pour retrouver l'extension.



Une icône apparaît dans la barre de lancement. Le plugin peut être lancé à partir de celle-ci, ou en passant par le menu extension > ModulExport > Module d'export.

L'installation s'arrête ici pour le client. Pour que le plugin fonctionne, des éléments doivent être présents aussi coté serveur.

## Coté serveur

La configuration coté serveur est plus complexe et doit être réalisée par le géomaticien.

### A/ Préparer la base de donnée.

#### 1/ S'assurer de l'intégrité de sa base.

La base de donnée doit être celle du conservatoire,, sur serveur PostgreSQL, et disposer de son schéma bdfauneflore.

#### 2/ Créer les tables permanentes.

Le schéma « export » doit ensuite être crée dans la même base que bdfauneflore : bdcenpicardie.

L'administrateur doit ensuite insérer 4 tables qui respectent une nomenclature et une structure précise. Ces tables seront permanentes et sont indispensables au bon fonctionnement du plugin.

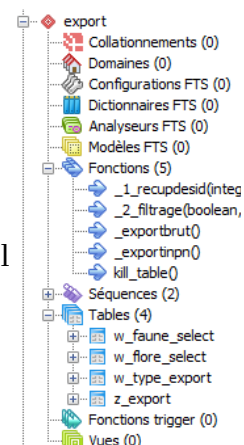
- **w\_faune\_select / w\_flore\_select** : Pour sélectionner des espèces.
- **w\_type\_export** : répertoire les types d'exports, leurs noms, et les fichiers .bat qui y sont associés.
- **z\_export** : la couche géographique qui sert à sélectionner la surface d'export.

Le détail de ces tables est disponible en annexes.

## 2. Créer les fonctions.

Toujours dans le schéma d'export, il faut créer diverses fonctions qui seront appelées par le *plugin*. Ces fonctions sont également disponibles en annexe.

- **\_1\_recupdesid** : Fonction qui effectue un tri géographique sur la sélection de l'utilisateur.
- **\_2\_filtrage** : Fonction de moissonnage qui récupère les critères attributaires de l'utilisateur et les applique de façon soustractive.
- **\_exportXXX** : Chaque modèle d'export dispose de sa propre fonction. Il y aura donc une fonction par modèle.
- **kill\_tables** : permet un nettoyage (par suppression) des tables temporaires et vues du schéma export.



## B/ Implémenter les fichiers .bat.

Un dossier spécifique au module d'export doit être mis en place :

[S:\00\\_MODULE\\_EXPORT\\_BDCEN](#). Le chemin à son importance car il est appelé dans les fonctions.

Ce dossier contient 2 répertoires :

- **batfiles** : Le répertoire qui contient les fichiers .bat. Chaque fichier est mis en lien avec un type d'export ; on a donc 1 .bat par type d'export.
- **Exports** : Le répertoire qui reçoit le résultat des exports : Les fichiers .csv, .shp, .xml... Dans l'architecture retenue, **les exports ne peuvent se faire que sur un répertoire sur le serveur.**

Les fichiers .bat permettent de faire appel à OGR2OGR sans coder une fonction en dur en Python et son garant de la pérennité de l'évolutivité du module d'export. **Les mots de passe postgres sont écrits en dur dans ce fichier, le répertoire doit donc être très sécurisé.**

Les fichiers .bat doivent être adaptés selon l'utilisateur qui va utiliser le plugin, avec la définition du chemin du dossier d'OGR2OGR en début de fichier, avec une syntaxe similaire à celle-ci :

```
cd /d C:\OSGeo4W64\share\gdal\
```

Selon l'installation, la définition du chemin d'OGR2OGR dans le path peut permettre de se passer de cette première ligne. Voici la procédure dans Windows 7 :

- Clic droit sur « Ordinateur » > Propriété
- Dans la grande fenêtre qui s'ouvre, cliquez sur le lien "Paramètres système avancés" dans la

liste de gauche.

- Dans la nouvelle fenêtre ainsi ouverte, cliquez sur le bouton “Variables d’environnement...”
- Dans cette fenêtre, dans la partie “Variables systèmes” (la liste du bas), il faut cliquer sur la ligne qui contient ‘Path’ .

On ajoute alors le chemin d'OGR2OGR. Dans tout les cas, la gestion des path sous windows n'est pas ergonomique et il est conseillé de copier le contenu de la ligne dans un bloc note pour une utilisation plus simple

Le *plugin* est prêt à l'emploi.

# Exemple d'utilisation du module d'export (modèle INPN)

## A/ Préparatifs.

### 1/ Importer sa couche z\_export

Une fois le module installé, il faut se connecter à la base de donnée bdcenpicardie dans Qgis, et chercher dans le schéma export la couche z\_export. Celle-ci est ensuite chargée sur le canevas de Qgis.

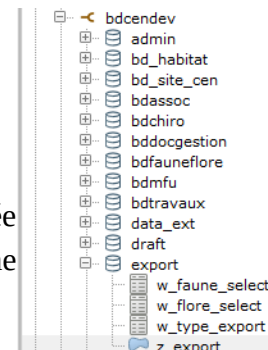
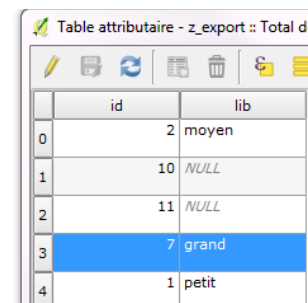


Illustration 2:  
z\_export dans

### 2/ Sélectionner sa surface de travail.

L'utilisateur peut alors créer l'entité de son choix, surface sur laquelle portera l'export. Il est bien sûr possible et encouragé de faire un copier-coller d'une entité d'une autre couche sur z\_export. Chaque entité dispose d'un ID, ainsi qu'un libellé qui apparaîtra dans la fiche d'export. Il peut donc être intéressant de reprendre le nom d'un site ou d'un lieu dit dans l'entité de z\_export.



	id	lib
0	2	moyen
1	10	NULL
2	11	NULL
3	7	grand
4	1	petit

Illustration 3: constitution de  
z\_export

Il faut ensuite sélectionner la ou les entités sur lesquelles porteront l'export. L'utilisateur peut ensuite lancer le *plugin*. Si aucune entité n'est sélectionnée ou si une autre couche que z\_export a le focus, une structure de contrôle renverra un message d'erreur.

## B/ Rentrer ses paramètres d'export.

module d'export

**Choix du destinataire**

Intitulé de l'export

Structure du destinataire

Destinataire

**Choix du type de sélection temporelle**

☒ Date de Saisie ☐ Année d'observation

Date minimale de prise en compte

Date maximale de prise en compte

**Paramètres de moissonnage**

☐ Inclure les données externes ☒ Exclure les données externes

☐ Uniquement les objets Valides ☐ Uniquement les objets présent

**Sélection des espèces prises en compte**

☐ Toutes les espèces ☒ Sélectionner des espèces

☐ Toute la flore ☐ Toute la faune

**Choix du type d'export**

Choix de la méthode d'export

☐ Nettoyage automatique

Une fois le plugin lancé, l'utilisateur reçoit cette fenêtre de dialogue avec plusieurs critères à renseigner.

On remarque plusieurs grandes sections :

- **Choix de destinataire.**
- **Choix du type de sélection temporelle.**
- **Les paramètres de moissonnage,** relatifs aux attributs de la donnée.
- **Le choix d'espèces.**
- **Le type d'export.**
- **Les fonctions relative à l'export.**

Illustration 4: L'interface QT du module d'export

### a) Choix de destinataire.

Les informations rentrées dans cette section sont uniquement reprises dans la fiche de métadonnée, ce qui permet une traçabilité des exports. **L'utilisateur veillera à ne pas utiliser d'accents ou caractères spéciaux.**

### b) Choisir la date sur laquelle porte l'export.

L'utilisateur peut vouloir chercher de la donnée selon 2 critères temporels : La date de saisie ou l'année d'observation. A l'utilisation, trouver une date de saisie précise est utile, pour retrouver par exemple des erreurs ou une session de saisie particulière. Pour l'observation, on se contente en revanche d'une recherche à l'année. L'interface Qt permet une adaptation dynamique de la boîte de dialogue de saisie.

☒ Date de Saisie ☐ Année d'observation

Date minimale de prise en compte

Date maximale de prise en compte

☐ Date de Saisie ☒ Année d'observation

Date minimale de prise en compte

Date maximale de prise en compte



### c) Choisir les paramètres de moissonnage.

Après les critères temporels, il faut définir sur quels attributs va porter l'export.

- L'export portera donc :
  - Sur les données internes **ET** externes.
    - **OU BIEN**
  - Sur les données internes **uniquement**.

**ET**

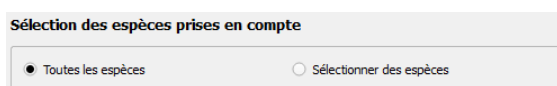
- Sur les données jugées valides **ET** invalides.
  - **OU BIEN**
- Sur les données jugées valides **uniquement**.

**ET**

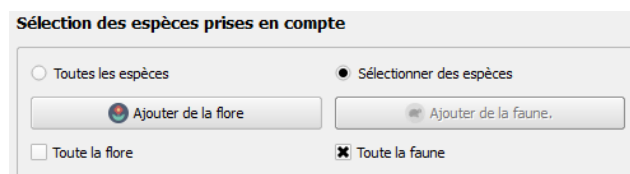
- Sur les données présentes **ET** absentes.
  - **OU BIEN**
- Sur les données présentes **uniquement**.

### d) Choisir les espèces à prendre en compte.

L'utilisateur a un contrôle fin sur les espèces qu'il désire inclure dans son export. Par défaut, toutes les espèces, faunes et flores sont sélectionnées.



*Illustration 6: Sélection de toutes les espèces*



*Illustration 5: Sélection de toute la faune et d'espèces flores spécifiques*

Mais en cliquant sur Sélectionner des espèces, l'interface QT affiche dynamiquement de nouveaux boutons.

En cliquant sur ajouter de la faune ou ajouter de la flore, l'utilisateur accède à une nouvelle interface :



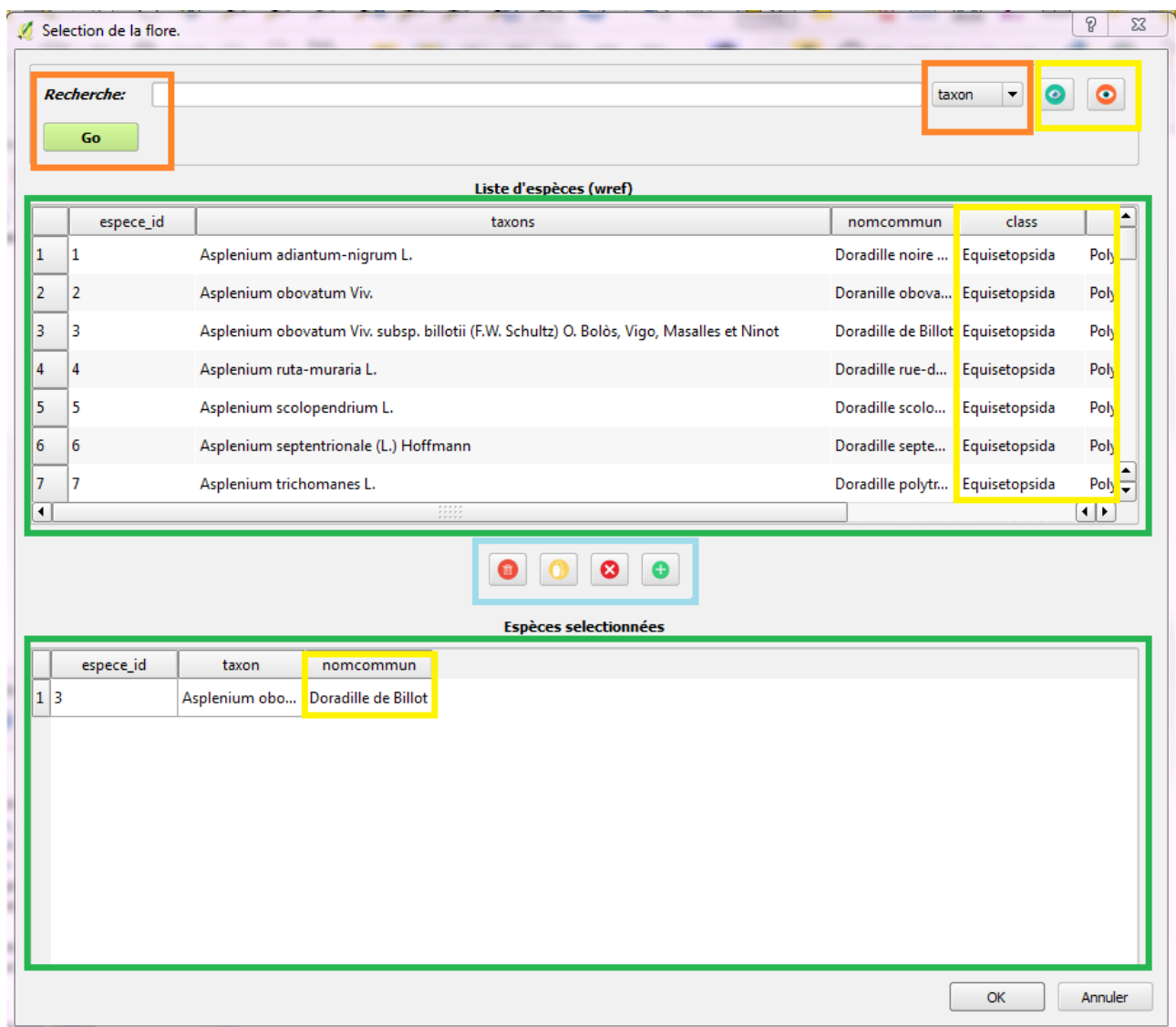


Illustration 7: fonctionnement de la fenêtre de sélection d'espèces

La fenêtre se compose de plusieurs sections : en haut les fonctions de recherches et d'affichage, un tableau supérieur de recherche d'espèces central, et un tableau inférieur qui affiche la sélection d'espèces.

Il convient de décrire ces fonctionnalités.

#### Les fonctions de recherches en Orange :

L'utilisateur peut entrer un ou des mots dans la barre de recherche, puis sélectionner un champ sur lequel la recherche portera. Il appuie ensuite sur le bouton vert pour qu'apparaissent dans le tableau central les résultats correspondant à sa recherche.

#### Les tableaux en Vert :

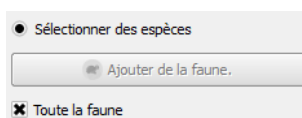
Les tableaux reprennent explicitement des champs de table w\_ref\_espece de la base de donnée, c'est-à-dire la table qui contient les données relatives aux espèces. Par défaut, la majorité des champs sont cachés pour des raisons d'ergonomie. L'utilisateur peut ensuite sélectionner une ou plusieurs lignes qu'il transfère sur le tableau d'en bas, qui contient les espèces à choisir pour l'export.

### Les fonctions d'affichage en Jaune :

En cliquant sur l'œil vert, des champs cachés apparaissent. Il y en a davantage sur le tableau du haut que sur celui du bas ; l'utilisateur peut avoir besoin de davantage d'informations pour alimenter sa sélection. Pour revenir à un mode d'affichage minimaliste, il suffit de cliquer sur l'œil orange.

### Les fonctions de manipulation de données en Bleu :

Cette section permet de mettre en relation les 2 tableaux. L'icône de poubelle orange permet de vider tout le contenu du tableau inférieur et donc de générer un tri total pour l'export. La souris jaune permet de désélectionner tout les entités sélectionnées dans le tableau du haut ; il s'agit d'une fonction de confort d'utilisation. La croix rouge permet de supprimer les entités sélectionnées dans le tableau du bas. Et le plus vert permet d'ajouter une sélection du tableau du haut vers le tableau du bas.



En revenant sur la fenêtre principale du module, on constate des petites boîtes à cocher sous les boutons d'ajout. L'utilisateur peut alors décider d'inclure toute la faune ou toute la flore avec une case à cocher. S'il sélectionne toute la faune, le bouton d'ajout de faune sera alors grisé. Par contre, la liste de flore spécifique ajoutée sera aussi prise en compte. A condition de faire attention à ce point, l'utilisateur pourra facilement décider de sélectionner toute la flore ainsi qu'une espèce faunistique ou plus.

### c) Choix du type d'export.

Le module d'export a pour vocation d'être souple, et permet de faire plusieurs types d'exports. L'utilisateur peut alors retrouver sa liste d'exports dans un menu déroulant. Lors d'ajout de types d'exports dans la base de donnée, le plugin les intègre automatiquement dans sa liste déroulante.

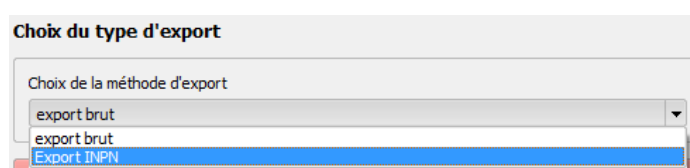
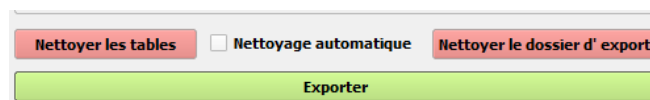


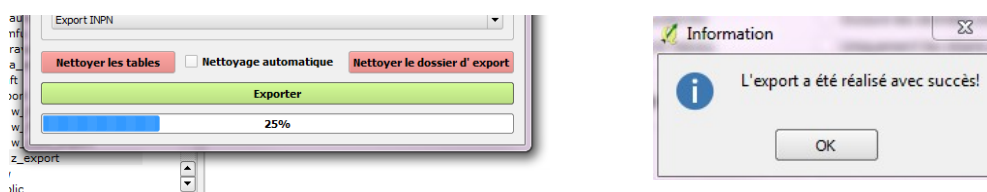
Illustration 8: La liste déroulante des types d'exports.

## d) Les fonctions relatives à l'export.



Les boutons colorés en rouge ont une vocation de nettoyage : L'un pour supprimer les tables temporaires du schéma export, et l'autre pour vider le dossier d'export. Les 2 sont facultatifs pour réaliser des exports, mais peuvent faciliter l'administration ou la clarté des exports. Une boîte à cocher offre la possibilité de supprimer automatiquement les tables temporaires à la fin de l'export. (Ce n'est évidemment pas le cas pour le dossier d'export...)

Enfin, le bouton Exporter, en vert, lance l'export en prenant en compte les paramètres passés dans la boîte de dialogue.



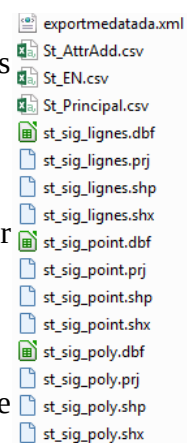
Une fois l'export lancé, une barre de progression indique à l'utilisateur la progression du programme. Et en cas de succès, un message informe l'utilisateur. En cas d'échec, Qgis renverra un message d'erreur en général suffisamment verbeux pour trouver le dysfonctionnement.

## C/ Identifier le résultat de son export.

L'intégralité de l'export peut-être retrouvé dans le dossier export sur le serveur. Dans le cas d'un export sur le modèle de l'INPN voici à droite les éléments présents.

On trouve 3 familles de fichiers :

- **Une fiche de métadonnée** au format XML. Sa structure est la même pour tout type d'export.
- Des **fichiers .CSV** qui correspondent au standard exigé par l'INPN.
- Des **fichiers Shapefiles** qui rassemblent les géométries par types. Dans le cas de l'export INPN, flore et faune sont mélangés.



*Illustration 9:  
résultat d'un  
export au  
format INPN*

## a) Le fichier de métadonnée.

Le fichier de métadonnées est au format XML, qui est ouvert par défaut sur Windows sur un navigateur internet. Cet affichage permet de déplier et replier des branches du fichier.

```
<?xml version="1.0"?>
<root>
  <parametres_d_export>
    <ObjetsSIG>
      <id libelle="grand" ID="7"/>
    </ObjetsSIG>
    <paramobjets Validite_de_la_donnee="Invalide" Donnée_externe="Exclusion" Certitude_de_presence="Absence">
      <param_date date_portant_sur="La date d'observation" date_minimale_de_prise_en_compte="01-01-2000" date_maximale_de_prise_en_compte="01-01-2015"/>
    </paramobjets>
    <Resume_export Type_export="export brut">
      <Destinataire structure="structureduDestinataire" libele_export="monexport" interlocuteur="ledestinataire"/>
      <stats_especes Nombre_total_de_taxons="461" Nombre_total_d_entite="3033">
        + <faune_exportees nombre_d_especes="244" nombre_d_entites="2324">
        + <faune_exportees nombre_d_especes="217" nombre_d_entites="709">
      </stats_especes>
    </Resume_export>
  </parametres_d_export>
  <Description_des_donnees>
    <formats date_export="16/06/2016">
      <csv_format separateur=";" encodage="UTF-8"/>
      <sig_format extension="Shapefile" SCR="Lambert 93"/>
    </formats>
  </Description_des_donnees>
</root>
```

Illustration 10: Affichage des métadonnées dans un Web Browser

On retrouve bien les informations tapées dans Qgis avec quelques statistiques en prime. L'affichage peut-être plus convivial en passant par le freeware XML Viewer. Ou le site codebeautify.org.

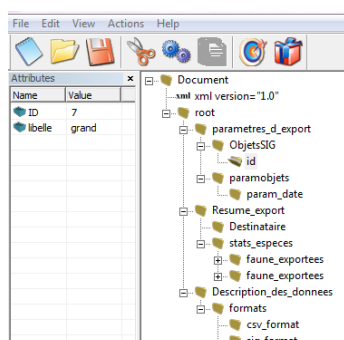


Illustration 11: Affichage du XML dans XMLViewer

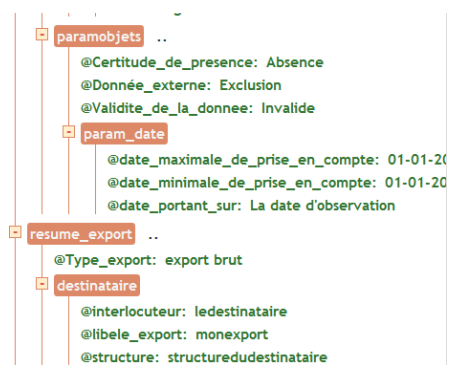


Illustration 12: Affichage du XML sur le site codebeautify

## b) Des fichiers CSV.

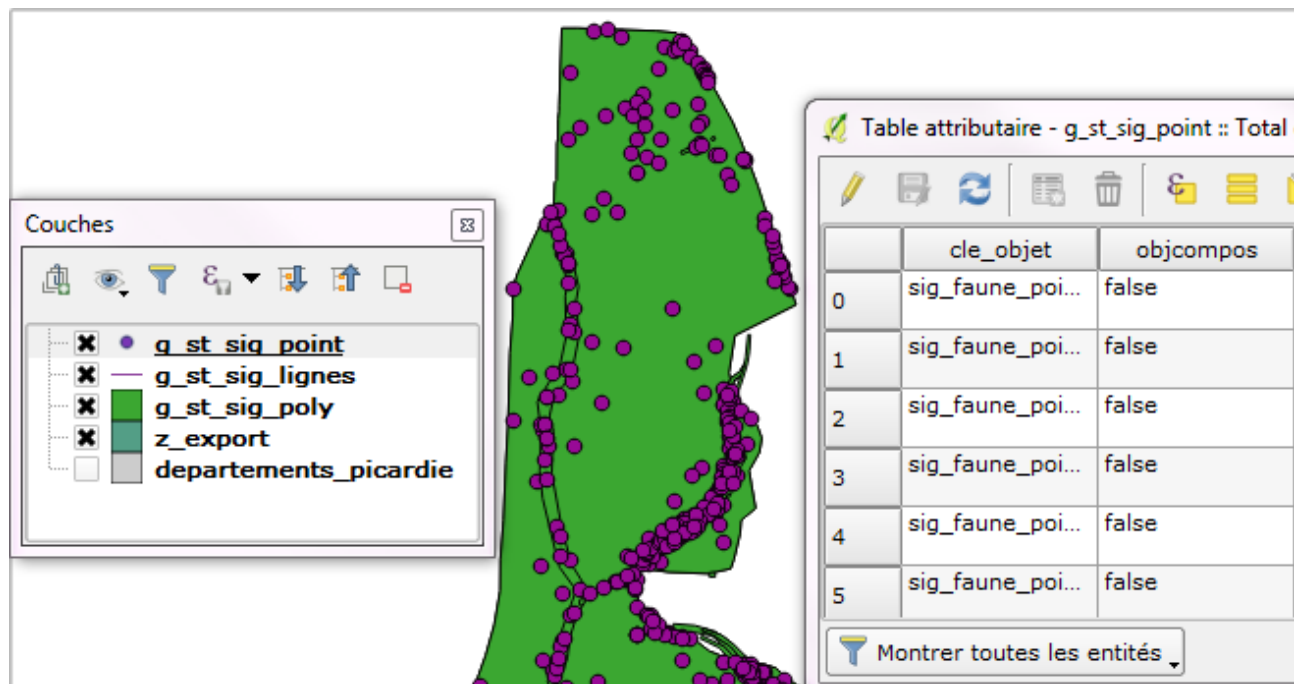
Les exports peuvent aussi fournir des fichiers CSV, qui ont un même avantage que le XML : l'interopérabilité. Ces fichiers peuvent être ouverts dans un Sig ou dans un tableur.

	A	B	C	D	E	F	G	H	I
1	cleobs	statsource	orggestdat	dspublic	statobs	cdref	cdnom	vtaxref	nomcite
2	sig_faune_p	Te	COMCOM Sa	Pr	Pr	61057	61057	v9.0	Capreolus ca
3	sig_faune_p	Te	COMCOM Sa	Pr	Pr	61057	61057	v9.0	Capreolus ca
4	sig_faune_p	Te	COMCOM Sa	Pr	Pr	259	259	v9.0	Bufo bufo
5	sig_faune_p	Te	COMCOM Sa	Pr	Pr	259	259	v9.0	Bufo bufo
6	sig_faune_p	Te	COMCOM Sa	Pr	Pr	444443	342	v9.0	Rana ridibun

Illustration 13: ouverture d'un CSV avec Excel

### c) Des fichiers shapefile

L'utilisateur peut vérifier la véracité des informations géographiques en insérant dans son SIG les fichiers de forme obtenus. Voici le type de résultat attendu :



Les entités sont bien intersectées dans l'entité de départ avec seulement 3 champs dans le cas d'un export INPN : cle\_objet, objcompos et la geom (invisible dans Qgis).

### D/ Contrôle supplémentaires.

L'utilisateur peut contrôler le succès de ses opérations en consultant la table r\_id\_all, qui rescence les entités qui ont été exclues sur les critères de dates et d'attributs. Cette table nous indique donc le différentiel entre la pure sélection géographique et celle attributaire avec une valeur booléenne négative pour les exclus. La table r\_retock contient les raisons de ces exclusions. Un jointure entre ces 2 tables est possible sur le champs id\_perm.

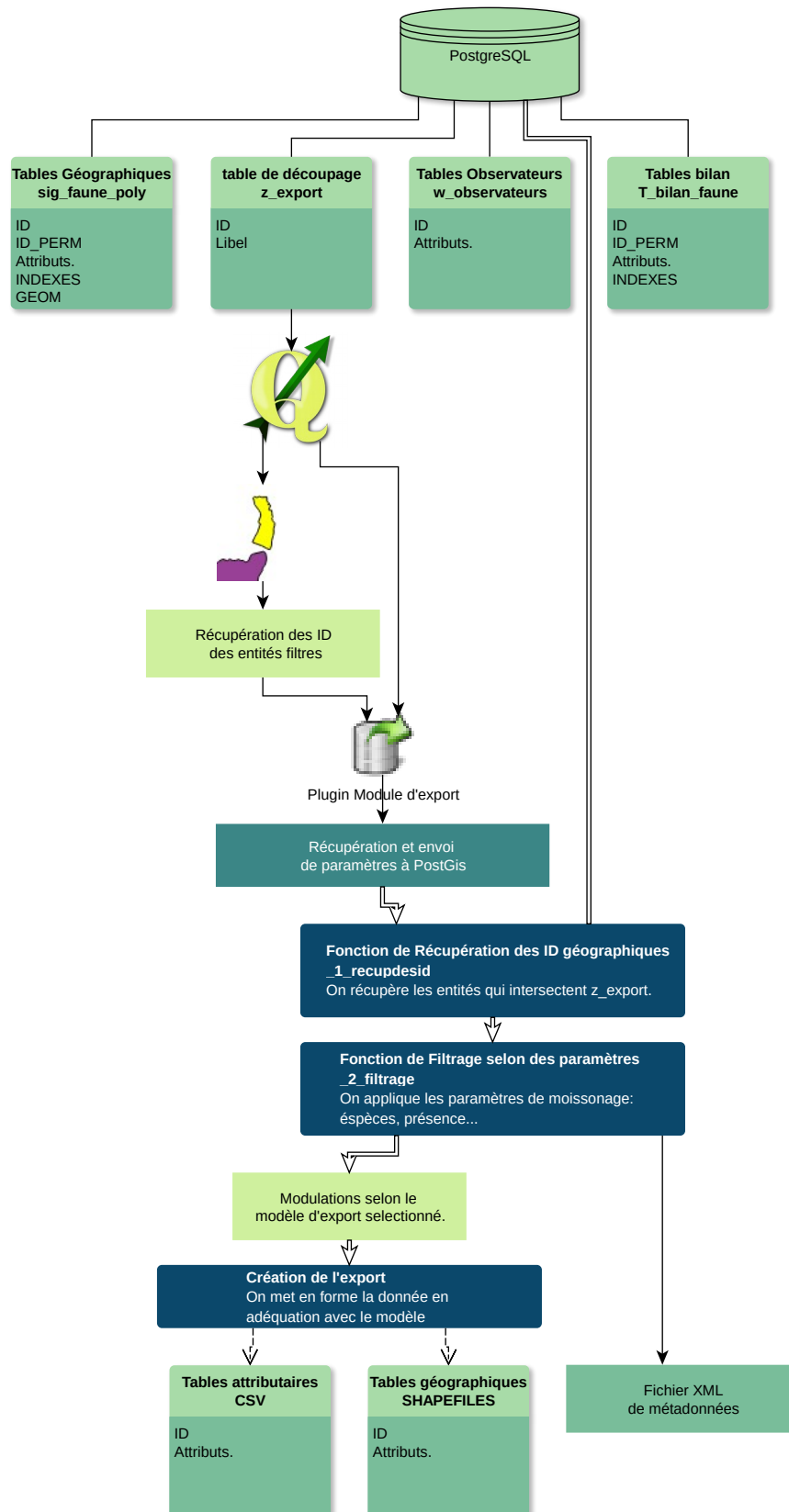
Table attributaire - r_id_all :: Total des entités			Table attributaire - r_retock :: Total des entités: 388		
	id_perm	presence		id_perm	raison
3029	t_observation...	t	0	t_observation...	date de saisie indesirable
3030	t_observation...	t	1	sig_flore_poin...	date de saisie indesirable
3031	t_observation...	t	2	t_observation...	date de saisie indesirable
3032	t_observation...	t	3	t_observation...	date de saisie indesirable
3033	sig_faune_poi...	f	4	sig_flore_poin...	date de saisie indesirable

Illustration 14: Aperçu des tables de contrôle.



# Schéma de fonctionnement.

## Simplifié



- PostgreSQL dispose de l'intégralité de la donnée. La couche de découpage y compris.
- Présence de tables non géographiques.
- L'utilisateur sélectionne une entité sur la couche de filtrage puis lance le plugin et détermine des paramètres d'export.
- Les informations sont transmises à une succession de fonctions Postgres.
- Pendant ce temps, Qgis est passif.
- Création d'une fiche de métadonnée à la fin du filtrage.
- L'export est réalisé avec la génération de fichiers CSV et Shape.

Ce schéma très simplifié permet une première approche du module d'export, et met en lumière l'importance de PostgreSQL par rapport à Qgis. Mais la simplification entraîne par nature l'imprécision, qui peut mener à l'erreur. Il convient donc de faire un développement sur le fonctionnement du module en le présentant sous la forme de « sous-sections ».

## Tables utilisées.

Le sommet du dessin ci-dessus nous montre une base Postgis avec quelques tables. La réalité est plus complexe avec différents schémas et beaucoup d'attributs par tables.

Le module n'utilise que 3 schémas :

- **bdfauneflore** pour récupérer et interroger la donnée faunistique et floristique.
- **Export** pour les données intrinsèques aux plugin.
- **bd\_site\_cen** pour les données géographiques des sites.

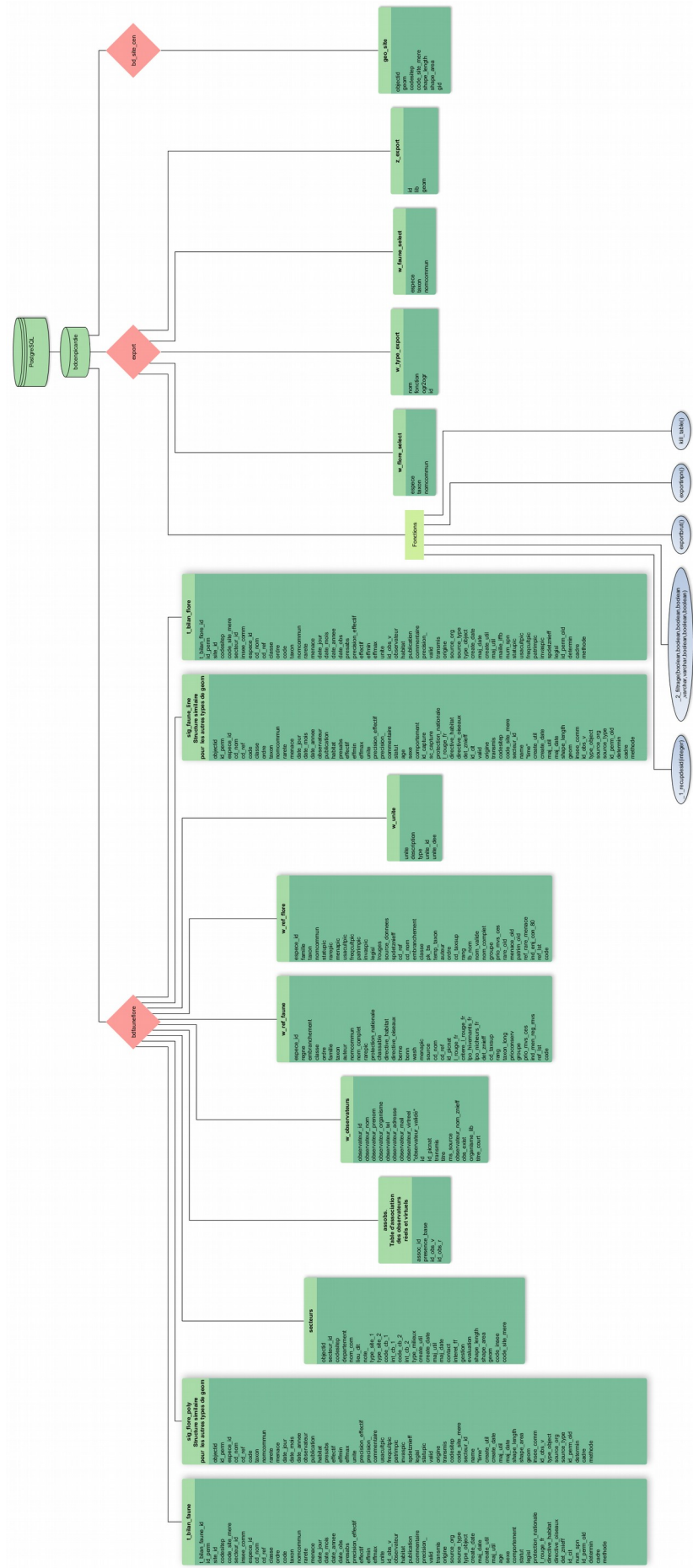
Les tables de bdfauneflore sont nombreuses et riches en champs et lignes. En opposition avec celles dans le schéma d'export, dont le nombre de lignes variera selon la surface d'export, mais avec un faible nombre de colonnes constant. La légèreté de ces tables a pour but de tirer un maximum parti du Système de Gestion de Bases de Données Relationnelles (SGBDR), avec des jointures permanentes aux tables de bdfauneflore, qui sont pourvues d'index spatiaux et classiques.

Dans la structure de la base, la faune et la flore sont séparées, avec beaucoup d'attributs en commun, mais pas tous. Ces divergences se retrouvent dans les tables de bilan et de sig, ainsi que dans les référentiels d'espèces.

La redondance de donnée dans le schéma bdfauneflore est due à l'organisation du CenPic découpé en antennes : les tables sont faites autant que possible en dur avec un système de régénérations pour maximiser la vitesse de consultation.

Voici la constitution des tables qui vont concerner le module d'export, sans entrer dans le détail de leurs relations, de leurs types, règles ou index.





## Comprendre le rôle des tables des schéma bdcenpicardie et bd\_site\_cen

### **t\_bilan\_faune / t\_bilan\_flore :**

Ces tables contiennent les informations relatives aux observations. Les informations entre t\_bilan\_faune et flore peuvent varier, ce qui empêche un regroupement simple des tables. Aussi, ces tables entrent en relation avec d'autres tables, comme le référentiel d'espèces, où encore les observateurs. Un lien peut également se faire avec les tables sig qui bénéficient des entités géographiques et de beaucoup d'attributs, ainsi que T\_Observation qui contient les observations qui n'ont pas de colonnes géométriques.

*Note : Dans les faits, les tables t\_bilan sont régénérées régulièrement avec les informations d'autres tables, telles que celles de sig.*

Il s'agit des tables les plus lourdes, qui disposent respectivement d'environ 120 000 et 150 000 enregistrements, avec plus d'une cinquantaine de colonnes chacune. Le nombre de colonnes importantes a motivé le choix de ne pas générer de tables qui dupliquent les enregistrements pour éviter de traiter trop de donnée en même temps.

### **Sig\_ \* :**

Au nombre de 6, ce sont les tables qui contiennent les données géographiques des entités faune / flore recensées. Elles contiennent des attributs qui sont reversés dans la table t\_bilan. Pour notre *plugin*, la colonne de géométrie sera donc celle qui retient notre attention.

### **t\_observations\_\* :**

Les tables qui contiennent les observations non référencées géographiquement. Pour pourvoir les transmettre géographiquement, ces observations sont raccrochées à un secteur et si possible à un site.

### **Secteur :**

La couche de secteurs permet de localiser des observations non référencées géographiquement à une échelle peu précise.

### **bd\_site\_cen.geo\_site :**

La couche des sites permet de localiser des observations non référencées géographiquement à une échelle plus précise que celle des secteurs.

**w\_ref\_faune/flore :**

La table qui contient les informations sur les espèces est notamment invoquée lors de la phase de sélection d'espèces dans le plugin.

**w\_observateurs :**

La table qui regroupe les observateurs réels et virtuels. La table est en relation avec les observations, qui pointent généralement sur des observateurs virtuels (groupes d'observateurs). Dans le cas de l'export INPN, on désire des observateurs réels uniquement.

**Asobs :**

Table qui permet de faire le lien entre les observateurs réels et virtuels.

**w\_unite :**

Rescence les valeurs qu'il est possible d'utiliser pour quantifier une observation. Il y a un attribut qualitatif et quantitatif. Dans le cas d'un export pour l'INPN des adaptations sont nécessaires.

Les tables permanentes du schéma export ont déjà été abordées dans la section d'installation du plugin coté serveur à la page X . Les tables temporaires de ce schéma seront abordées ultérieurement dans le manuel.

# Structure des fonctions sur schéma d'export.

La relation du plugin avec PostgreSQL consiste en un appel successif de fonctions qui vont faire le travail de traitement de la donnée. D'abord un filtrage géographique, puis attributaire, et enfin un export relatif au modèle choisi. La dernière fonction, kill\_table, qui supprime les tables temporaires, est indépendante des autres.

## 1/\_1\_Recupdesid

**Paramètre :**

**param\_listeid** > integer[] : ID d'objets z\_export.

### a) Récupérer les entités référencées géographiquement.

La fonction de filtrage géographique prend en compte les ID des objets de z\_export sélectionnés dans Qgis et transmis par le plugin. Le paramètre est donc une liste d'entiers (integer[]). C'est le seul paramètre de cette fonction.

```
-- 1 Récupération de la liste des ID
DROP table IF EXISTS ids;
CREATE TEMP TABLE ids AS(
SELECT *
FROM export.z_export e
WHERE e.id IN (select(unnest(PARAM_listeid))))
```

La première étape consiste en une simple sélection des entités de z\_export présentes dans la liste du paramètre, et les conserver dans une table temporaire ids.

La table r\_id\_all est ensuite créée. Elle est primordiale et sera constamment appelée par la suite.

r_id_all
id_perm character varying(100)
presence boolean default TRUE

```
-- 2 Création de la table d'accueil des ID_PERM
DROP TABLE IF EXISTS export.r_id_all CASCADE;
CREATE TABLE export.r_id_all(
id_perm character varying(100),
presence boolean DEFAULT 1::boolean)
```

Elle contient un champ d'ID pour récupérer les entités des entités retenus dans le filtrage, et un champ booléen par défaut à vrai qui sera utilisé dans les fonctions ultérieures.

```
-- 3 Peuplement de la table avec les ID geo
INSERT INTO export.r_id_all(id_perm)
SELECT DISTINCT fpo.id_perm
FROM ids decoup,
bdfauneflore.sig_faune_poly fpo
WHERE st_intersects(fpo.geom, decoup.geom)
UNION
SELECT DISTINCT fal.id_perm
```

Cette table peut ensuite être remplie dans un premier temps avec les éléments des tables sig, qui contiennent une géographie. La fonction `st_intersects` de Postgis est utilisée, et les résultats sont unis. Seuls les `ID_perm` sont conservés.

## b) Récupérer les entités qui ne sont pas directement référencées géographiquement.

Toutes les tables n'ont pas de colonne géométrie, il faut donc essayer de les localiser le plus précisément possible : au site si possible, sinon au secteur.

```
--- 4 Recuperation les des sites (codesitep)
DROP TABLE if EXISTS export.r_codes_des_sites;
CREATE TABLE export.r_codes_des_sites AS
WITH site AS (
    SELECT geo_site.codesitep, code_site_mere,
    st_union(geo_site.geom) AS geom
    FROM bd_site_cen.geo_site
    GROUP BY geo_site.codesitep, code_site_mere
)
SELECT codesitep,code_site_mere, site.geom
FROM site, ids decoup
WHERE st_intersects(site.geom, decoup.geom)
```

### r\_codes\_des\_sites

**codesitep** character varying(50)  
**code\_site\_mere** character varying(50)  
**geom** geometry

Dans la section du code en rouge, la couche de sites est reconstituée

Une table permanente d'Ids des secteurs est aussi créée, en intersectant avec la sélection faite dans z\_export.

```
--5idem avec les secteur => recupere secteur_id + geom
drop table if exists export.r_id_des_secteurs;
create table export.r_id_des_secteurs as
select secteur_id, secteur.geom
from bdfauneflore.secteur,ids decoup
where st_intersects(secteur.geom, decoup.geom)
```

Les entités dont la géographie n'était pas directement renseignée peuvent désormais être associés à un site ou à un secteur. Une jointure est faite entre t\_bilan et les couches de site ou secteur, pour pouvoir récupérer l'ID perm de ces entités et les insérer dans r\_id\_all.

```
-- Tout les Id / meme non geo.
--6recuperer liste des idperm where : id_perm in (ma liste d id_perm geo) or (secteur_id in( ma liste)
INSERT INTO export.r_id_all(id_perm)
select DISTINCT bdf.id_perm
from bdfauneflore.t_bilan_flore bdf, export.r_id_des_secteurs sec
where
(bdf.secteur_id in (sec.secteur_id))
AND bdf.type_object not like ('sig%')

UNION
select DISTINCT bdf.id_perm
from bdfauneflore.t_bilan_flore bdf, export.r_codes_des_sites co
where
(bdf.codesitep in (co.codesitep) or bdf.codesitep in (co.code_site_mere))
AND bdf.type_object not like ('sig%')

UNION
select DISTINCT bdf.id_perm
from bdfauneflore.t_bilan_faune bdf, export.r_id_des_secteurs sec
where
(bdf.secteur_id in (sec.secteur_id))
AND bdf.type_object not like ('sig%')
UNION
select DISTINCT bdf.id_perm
from bdfauneflore.t_bilan_faune bdf, export.r_codes_des_sites co
where
(bdf.codesitep in (co.codesitep) or bdf.codesitep in (co.code_site_mere))
AND bdf.type_object not like ('sig%')
```

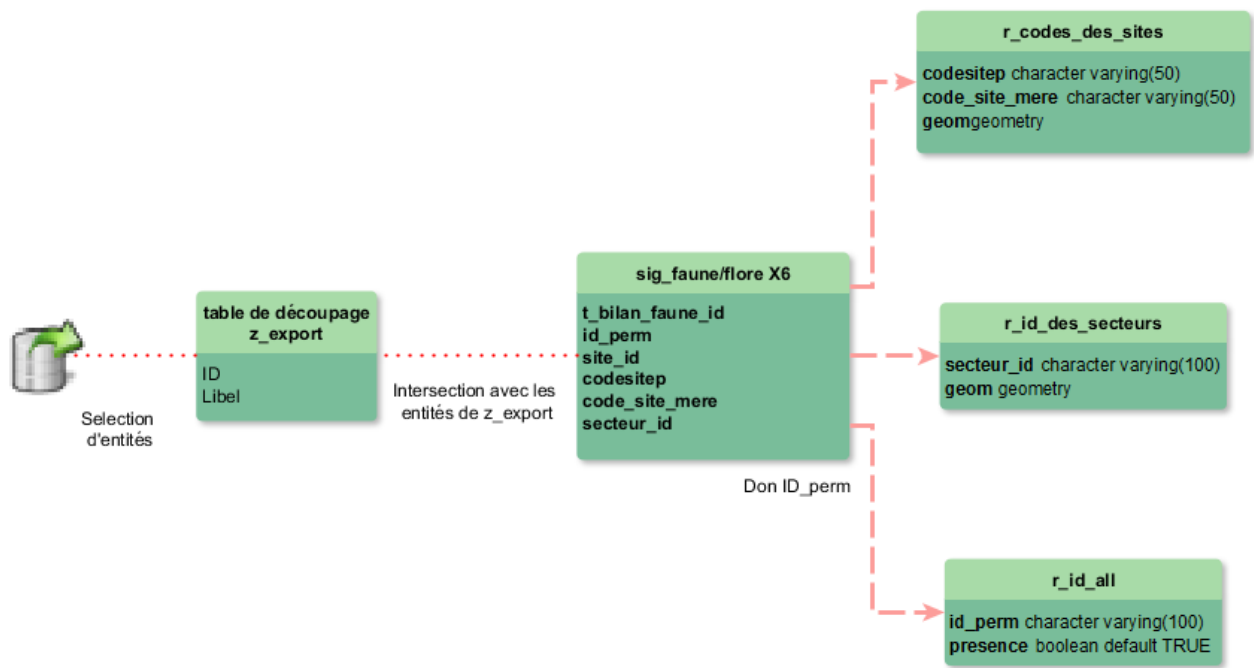
**r\_id\_des\_secteurs**

secteur\_id character varying(100)  
geom geometry

Enfin, un index *btree* est généré sur r\_id\_all.id\_perm pour accélérer les traitements à venir sur cette table. En revanche, aucun index n'est créé sur la colonne de booléens, car cette colonne sera amenée à subir un nombre important de modifications.

```
--Creation d un index sur id_perm
CREATE INDEX r_id_all_id_perm_idx
ON export.r_id_all
USING btree
(id_perm)
```

La fonction \_1\_recupdesid génère donc plusieurs tables, ré exploitables pour d'autres fonctions. La plus importante est néanmoins r\_id\_all, qui contient le résultat du tri géographique.



## 2/ 2\_filtrage

Paramètres :

**param\_espece** > boolean :

*TRUE = Sélection sur un pool d'espèces.*

*FALSE = Sélection de toutes les espèces.*

**param\_faune** > boolean :

*TRUE = Pas de prise en compte de la table w\_faune\_select.*

*FALSE = Si param\_espece = TRUE :Sélection du pool de faune de la table w\_faune\_select.*

**param\_flore** boolean :

*TRUE = Pas de prise en compte de la table w\_flore\_select.*

*FALSE = Si param\_espece = TRUE :Sélection du pool de faune de la table w\_flore\_select.*

**param\_typedate** boolean :

*TRUE = Tri sur la date de saisie.*

*FALSE = Trie sur la date d'observation.*

**param\_date\_debut** character varying :

*Passage de date de début en string : 'dd-MM-yyyy'*

**param\_date\_fin** character varying :

*Passage de date de fin en string : 'dd-MM-yyyy'*

**param\_valide** boolean :

*TRUE = Prise en compte des objets valides Uniquement.*

*FALSE = Pas de prise en compte du critère de validité.*

**param\_present** boolean :

*TRUE = Prise en compte des objets présents Uniquement.*

*FALSE = Pas de prise en compte du critère de présence.*

**param\_externe** boolean

*TRUE = Prise en compte des objets internes et externes.*

*FALSE = Prise en compte des données internes uniquement.*



## a) Créer une table de suivi du filtrage des éléments.

```
DROP TABLE IF EXISTS export.r_retock CASCADE;  
CREATE TABLE export.r_retock(  
  id_perm character varying(100),  
  raison character varying(200)  
);
```

r_retock
id_perm character varying(100)
raison varchar (150)

La table r\_retock, vide lors de sa création, se remplira de tout les id\_perm des entités qui sont exclues lors du triage. La colonne raison nous renseigne de la raison. Bien sûr un élément peut ne pas répondre à plusieurs critères. Dans notre architecture, nous choisissons de ne retenir que la première exclusion du tri. La table ne peut donc pas contenir de doublons. Une règle pour empêcher l'insertion de doublons est générée, pour garantir la cohérence de la table.

```
--Regle pour empecher l'insertion de doublons.  
CREATE OR REPLACE RULE no_noublon_retock AS  
  ON INSERT TO export.r_retock  
  WHERE (EXISTS ( SELECT 1  
                  FROM export.r_retock k  
                  WHERE k.id_perm = NEW.id_perm)) DO INSTEAD NOTHING
```

Les fonctions de tri sont systématiquement effectuées en deux temps. D'abord une insertion dans la table r\_retock de l'id\_perm de l'entité écartée, ainsi qu'une raison dans le champ raison.

Le tri s'effectue sur les attributs des entités, des jointures constantes sont ainsi réalisées entre id\_perm et les tables contenant les attributs à trier.

## b) Faire les tris sur les espèces.

Les critères faunistiques et floristiques portent sur les tables flore et w\_faune\_select, qui contiennent les listes des sélections faites dans les tableaux des plugin. Ces tables contiennent des id d'espèces, dont la liste est retrouvée en procédant à une jointure entre r\_id\_all et les tables t\_bilan. Voici le code effectué si l'export porte sur une sélection précise d'entités faunistique.

```
--FAUNE
-- si selection sur pool de faune SELECTIONNE.
IF param_espece IS TRUE AND param_faune IS FALSE THEN

  -- r_retock
  INSERT INTO export.r_retock(id_perm, raison)
  SELECT i.id_perm, 'espece indesirable' AS raison
  FROM bdfauneflore.t_bilan_faune t
  JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
  WHERE t.espece_id
  NOT IN (SELECT espece_id
          FROM export.w_faune_select)
  ;

  -- update
  UPDATE export.r_id_all
  SET presence = 0::boolean
  where r_id_all.id_perm
  IN (
    SELECT i.id_perm
    FROM bdfauneflore.t_bilan_faune t
    JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
    WHERE t.espece_id
    NOT IN (SELECT espece_id
            FROM export.w_faune_select)
  )
  ;
END IF
```

On remarque que les 2 requêtes sont très similaires. En rouge on trouve la jointure, en jaune la sélection que toutes les entités que l'utilisateur ne désire pas pour les passer en booléen 0 (en vert) dans la table r\_id\_all. La procédure est strictement identique pour la flore.

### c) Faire le tri sur les dates.

Réaliser un tri sur les dates demandées par l'utilisateur nécessite une mise en forme des champs de dates. Dans t\_bilan, les dates sont stockées de façon différente en fonction de si l'on est sûr de l'observation ou de la date de saisie (et mise-à-jour).

Dans le cas de la date de saisie, la requête doit prendre en compte la date la plus récente entre la date de création et celle

```
--retock
INSERT INTO export.r_retack(id_perm, raison)
SELECT i.id_perm, 'date de saisie indésirable' AS raison
FROM (
  SELECT
    i.id_perm,
    CASE
      WHEN f.maj_date IS NULL THEN f.create_date
      WHEN f.create_date >= f.maj_date THEN f.create_date
      WHEN f.create_date <= f.maj_date THEN f.maj_date
      ELSE f.create_date
    END AS create_date
  FROM bdfauneflore.t_bilan_faune f
  JOIN export.r_id_all j ON j.id_perm = f.id_perm
  WHERE create_date
    NOT BETWEEN PARAM_date_debut::timestamp AND PARAM_date_fin::timestamp
  UNION
  SELECT
    i.id_perm,
    CASE
      WHEN f.maj_date IS NULL THEN f.create_date
      WHEN f.create_date >= f.maj_date THEN f.create_date
      WHEN f.create_date <= f.maj_date THEN f.maj_date
      ELSE f.create_date
    END AS create_date
  FROM bdfauneflore.t_bilan_flore f
  JOIN export.r_id_all j ON j.id_perm = f.id_perm
  WHERE create_date
    NOT BETWEEN PARAM_date_debut::timestamp AND PARAM_date_fin::timestamp) as i
```

de saisie. Ceci est matérialisé sous la forme de CASE . (en bleu). Les champs de types varchar sont ensuite convertis à la volée en type date (en vert) grâce à la souplesse de PostgreSQL. Les requêtes étant similaires entre les table r\_rerock et r\_id\_all, nous nous contenterons de n'en montrer qu'une des deux pour la suite de la documentation. La requête se dédouble entre faune et flore, c'est inévitable et inhérent à la structure de la base de donnée. Les dates sont stockées dans le type timestamp.

Dans le cas d'une date d'observation, l'opération est simplifiée :

```
--retock
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'date d observation indésirable' AS raison
FROM bdfauneflore.t_bilan_faune bf
JOIN export.r_id_all i ON i.id_perm = bf.id_perm
WHERE bf.date_annee
NOT BETWEEN PARAM_date_debut AND PARAM_date_fin

UNION

SELECT i.id_perm, 'date d observation indésirable' AS raison
FROM bdfauneflore.t_bilan_flore bf
JOIN export.r_id_all i ON i.id_perm = bf.id_perm
WHERE bf.date_annee
NOT BETWEEN PARAM_date_debut AND PARAM_date_fin
```

En effet, la recherche ne porte cette fois que un champs, et aucune conversion de types à la volée n'est appelée.

### c) Faire le tri sur les critères de validité externalité et présence.

Le tri se fait grâce à un tri avec les tables t\_bilan pour chercher le champ de validité. De même pour la présence avec le champ presabs, et pour l'inclusion ou non des données externes, le champ transmis est interrogé.

### d) Création d'une liste des observateurs.

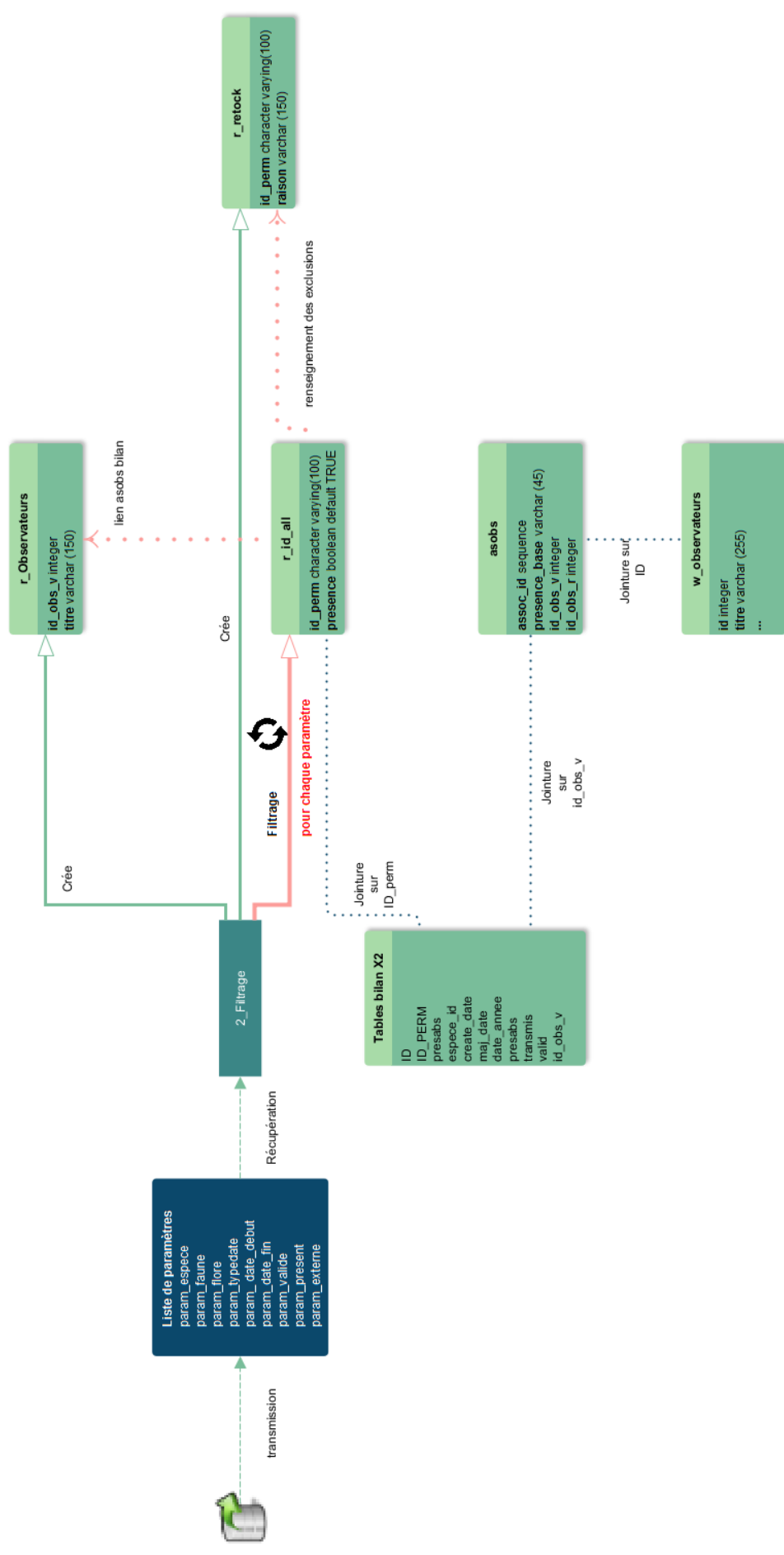
A la fin de ce filtrage, une nouvelle table contenant la liste des observateurs réels est également créée.

```
-- Liste des observateurs virtuels présents dans pool de id faune flore

drop table if exists export.r_observateur;
create table export.r_observateur as

SELECT V.id_obs_v, w.titre
FROM bdfauneflore.asobs a
] JOIN (SELECT i.id_perm, f.id_obs_v
FROM bdfauneflore.t_bilan_faune f
RIGHT JOIN export.r_id_all i ON i.id_perm = f.id_perm
UNION
SELECT i.id_perm, f.id_obs_v
FROM bdfauneflore.t_bilan_flore f
RIGHT JOIN export.r_id_all i ON i.id_perm = f.id_perm) as V
ON (a.id_obs_v = V.id_obs_v)
JOIN bdfauneflore.w_observateur w on (a.id_obs_r = w.id)
GROUP BY w.id, w.titre, V.id_obs_v
ORDER BY w.titre
```

Les ID d'observateurs virtuels sont récupérés avec une jointure entre r\_id\_all et les tables t\_bilan, puis une autre jointure est faite avec la table asobs pour pouvoir récupérer la liste des observateurs réels présents dans le filtrage.

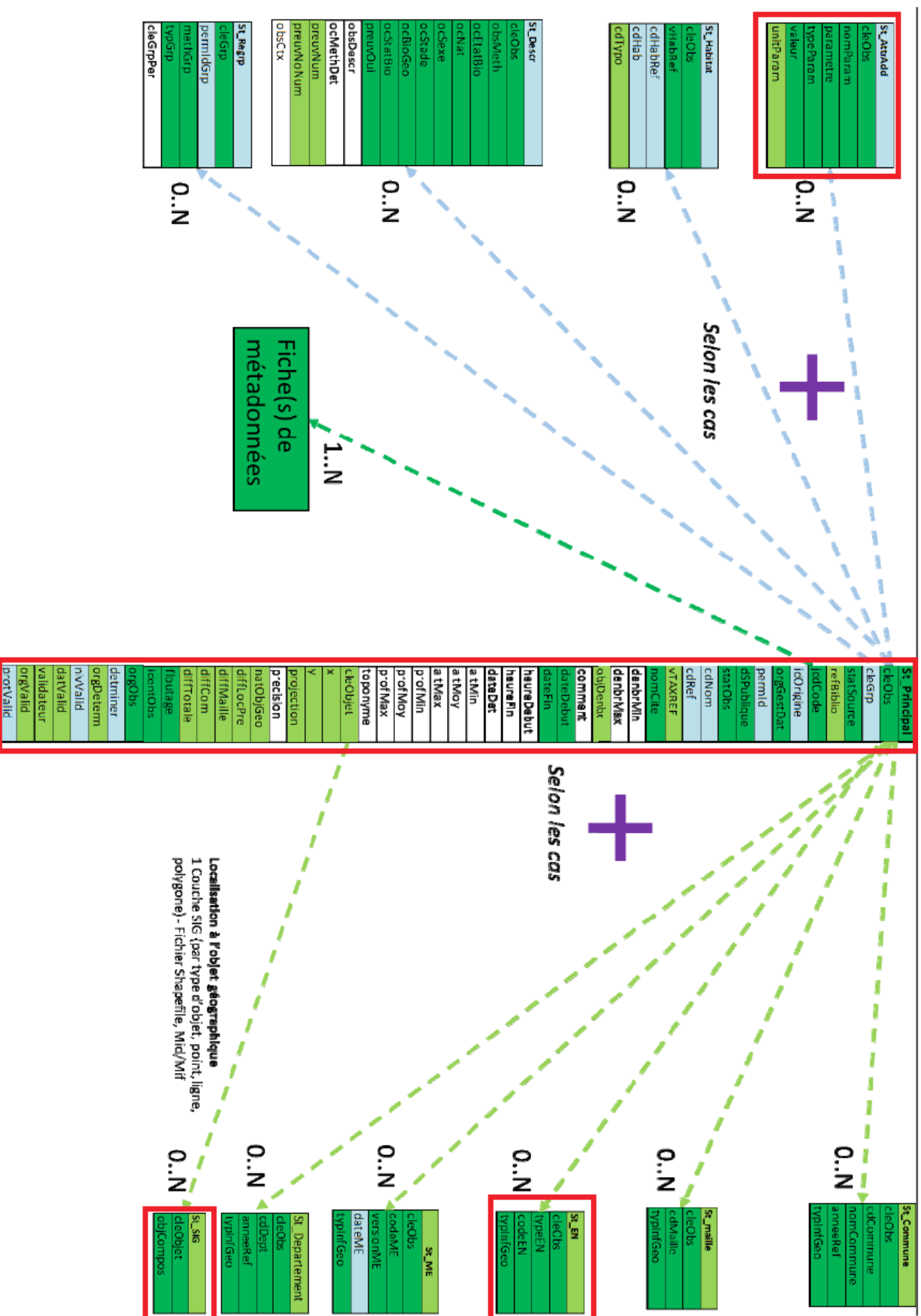


### **3/ Export INPN.**

Cette fonction à la charge d'exporter au format de l'INPN, et donc de sortir une donnée remodelée et divisée en plusieurs formats. Il y a donc plusieurs séquences dans cette fonction :

- Création du fichier ST\_Principal en adaptant la donnée du CenPic.
- Création des fichiers d'attributs supplémentaires : EN et ATTRADD.
- Création des tables SIG, qui contiennent donc les géométries.

Le modèle de l'INPN est disponible à la page suivante. Les éléments à générer sont encadrés en rouge.





## a) Remodeler la donnée pour générer St\_Principal.

La table St\_principal est constitué d'une série de sélections conditionnées sous la forme de requête avec WITH, rassemblées dans une seule et même table au final.

```
DROP TABLE IF EXISTS export.e_St_Principal;  
]Create TABLE export.e_St_Principal as(
```

### CléObs :

Le champs cléObs est constitué des id\_perm de r\_id\_all. Il y a qu'a récupérer ceux qui sont titulaires d'un booléen positif Dans toutes les requêtes à venir sur ST\_principal, seuls les lignes dont champ présence de r\_id\_all est positif seront traitées.

```
--CLEOBJ  
--Constitution d une liste de tout nos IDPerm.  
]WITH T_cleObs as (  
  SELECT i.id_perm  
  FROM export.r_id_all i  
  WHERE i.presence = 1::boolean
```

### Others :

Les autres champs qu'il n'est pas nécessaire de retravailler sont réunis dans une seule grande requête WITH:

```
-- CHAMPS BRUTS  
-- Une table qui contient les champs "tels quels" de tbilan  
T_inchanged as(  
  SELECT i.id_perm, t.cd_ref as cdRef, t.cd_nom as cdNom, t.taxon as nomCite  
  FROM export.r_id_all i  
  JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm  
  WHERE i.presence = 1::boolean  
  UNION  
  SELECT i.id_perm, t.cd_ref as cdRef, t.cd_nom as cdNom, t.taxon as nomCite  
  FROM export.r_id_all i  
  JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm  
  WHERE i.presence = 1::boolean)
```

### statsource :

Le champs d'origine est conformé au modèle INPN à l'aide CASE selon le modèle suivant.  
(jointure préalable avec t\_bilan)

```
CASE  
  WHEN t.origine = 'Donnée terrain' THEN 'Te'  
  WHEN t.origine = 'Donnée Biblio' THEN 'Li'  
  WHEN t.origine = 'Com.pers.' THEN 'NSP'  
  ELSE 'NSP'  
END as statSource
```

La forme de l'organisme gestionnaire peut rester conforme à la base de donnée du CenPic, sauf dans le cas de prestations où le sigle CENP est repris. Le recours au CASE est de nouveau de mise (jointure préalable avec t\_bilan) :

```

CASE
  WHEN t.source_type <> 'prestation' THEN t.source_org
  WHEN t.source_type = 'prestation' THEN 'CENP' --CenPicardie

```

dspublic :

Pour déterminer le type de prestation (dspublic), le combo CASE – JOIN sur T\_bilan est à nouveau utilisé :

```

CASE
  WHEN t.source_org = 'CENP' AND t.source_type = 'bénévole' THEN 'Pr'
  WHEN t.source_type = 'prestation' THEN 'Ac'
  WHEN t.source_org = 'CENP' AND t.source_type = 'interne' THEN 'Re'
  WHEN t.source_type <> 'CENP' AND t.source_type <> 'prestation' THEN 'Pr'
  ELSE t.source_type
END as DsPublic

```

statobs :

De même pour Statobs pour déterminer la certitude de présence des objets :

```

CASE
  WHEN t.presabs = '0' THEN 'No'
  WHEN t.presabs = '1' THEN 'Pr'
END as statObs

```

vtaxref :

En revanche, le référentiel pour la détermination des taxons est implémenté en dur :

```

T_vTAXREF as(
  SELECT i.id_perm, 'v9.0' as vTAXREF
  FROM export.r_id_all i
  JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
  WHERE i.presence = 1::boolean
  UNION
  SELECT i.id_perm, 'v9.0' as vTAXREF

```

denbrMin/Max :

La question du dénombrement a été déterminé de la façon suivante :

- Si une valeur de dénombrement minimale existe, alors celle-ci est gardée.
- Si cette valeur n'existe pas, alors la valeur d'effectif générique est prise.

Le schéma est le même pour la valeur maximale et minimale.

```

CASE
  WHEN t.effmin = '' OR t.effmin IS NULL THEN t.effectif
  ELSE t.effmin
END as denbrMin

```

### Objdenbr:

L'objet du dénombrement (objdenbr) est déterminé en deux temps. Une double jointure est faite avec les habituelle table t\_bilan mais aussi la table w\_unite, qui contient les unités. Le contenu du champs unite\_dee est récupéré, et s'il est vide alors la valeur 'AUTR' est retenue.

```
T_objdenbr as(
WITH brut AS(
SELECT i.id_perm, u.unite_dee
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON i.id_perm = t.id_perm
left JOIN bdfauneflore.w_unite u ON u.unite = t.unite
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm, u.unite_dee
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON i.id_perm = t.id_perm
left JOIN bdfauneflore.w_unite u ON u.unite = t.unite
WHERE i.presence = 1::boolean
)
SELECT b.id_perm,
       CASE
           WHEN b.unite_dee = '' OR b.unite_dee IS NULL THEN 'AUTR'
           ELSE b.unite_dee
       END as objDenbr
FROM brut b
)
```

### DateDebut :

Dans la table t\_bilan, la date d'observation est éclatée en 3 champs, en varchar. Un champ date doit donc être reconstitué en concaténant ces champs au format DD/MM/YYYY pour être conforme au format INPN.

Mais des champs jour et mois peuvent être vides. Si tel est le cas, un CASE prend en considération la date minimale : 01, pour le premier jour du mois, ou le premier mois (Janvier). A chaque fois, une jointure est refaite avec les tables t\_bilan.

```
T_dateDebut as(
    WITH jourplein as(
        select i.id_perm,
            CASE
                WHEN t.date_jour = '' Or t.date_jour is null THEN '01'
            ELSE t.date_jour
            END jp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        select i.id_perm,
            CASE
                WHEN t.date_jour = '' Or t.date_jour is null THEN '01'
            ELSE t.date_jour
            END jp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean),
    moisplein as(
        select i.id_perm,
            CASE
                WHEN t.date_mois = '' Or t.date_mois is null THEN '01'
            ELSE t.date_mois
            END mp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        select i.id_perm,
            CASE
                WHEN t.date_mois = '' Or t.date_mois is null THEN '01'
            ELSE t.date_mois
            END mp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean),
    annee as(
        SELECT i.id_perm, t.date_annee
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        SELECT i.id_perm, t.date_annee
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean)
    select i.id_perm, to_char(((j.jp || '/' || m.mp || '/' || a.date_annee)::date), 'DD/MM/YYYY') as dateDebut
    from export.r_id_all i
    Join jourplein j on (i.id_perm = j.id_perm)
    Join moisplein m on (i.id_perm = m.id_perm)
    join annee a on (i.id_perm = a.id_perm)
)
```

### DateFin:

Pour la date de fin le principe est le même, à l'exception qu'il faut prendre en compte l'irrégularité du nombre de jour dans les mois, ainsi que la particularité du mois de Février. Une structure de contrôle est intégrée avec un CASE pour gérer les jours en 30 ou 31 jours. Pour le cas de Février, un deuxième CASE est intégré, avec la vérification de si l'année sur laquelle porte la date est bissextile ou non.

```
CASE
  WHEN t.date_jour = '' Or t.date_jour is null
  THEN
    CASE
      WHEN moisplein.mp IN('01','03','05','07','08','10','12')
      THEN '31'
      WHEN moisplein.mp IN ('04','06','09','11')
      THEN '30'
      WHEN moisplein.mp = '02' THEN
        CASE
          WHEN
            t.date_annee::integer % 4 = 0
            AND
            t.date_annee::integer % 100 <> 0
            OR
            t.date_annee::integer % 400 = 0
          THEN '29'
          ELSE
            t.date_jour
        END
      ELSE t.date_jour
    END
  END jp
```

*Illustration 15: La double structure de contrôle mois/année bissextile.*

### Natobjgeo:

Pour ce cas de figure, une simple jointure avec t\_bilan et un Case de vérification du type d'objet dans type\_object est suffisant.

```
CASE
  WHEN t.type_object LIKE 'sig_%' THEN 'St'
  WHEN t.type_object LIKE 't_obs%' THEN 'In'
  ELSE t.type_object
END as natObjGeo
```

### Difs:

Pour les champs diflocpre, diffMaille, diffCom, et le champs de floutage, des valeurs abitraires ont été définies en concertation avec l'équipe SIG.

```
SELECT i.id_perm, 1 AS diflocpre, 1 AS diffMaille, 1 AS diffCom, 1 AS diffTotale, 0 AS floutage
```

### Observateurs :

L'objectif de ce champs est de réunir les noms des observateurs concaténés. En suivant le schéma de la base du CENPIC.

Un WITH crée une première table temporaire qui contient les groupes d'observateurs (appelés observateurs virtuels dans le CENPIC) ainsi que les individus qui les composent. En cas de nom de groupe vide, la mention « INCONNU » est appliquée.

```
-- identObs
|tmp AS (
    SELECT t1.id_obs_v,
           string_agg(t2.titre_court::text, ', '::text) AS identObs,
           string_agg(COALESCE(t2.observateur_organisme::text, 'INCONNU'), ', '::text) as orgObs
    FROM bdfauneflore.asobs t1
         JOIN bdfauneflore.w_observateur t2 ON t1.id_obs_r = t2.id
    GROUP BY t1.id_obs_v
)
```

Cette table temporaire peut ensuite être réutilisée grâce à un jointure entre l'id\_obs\_V et le résultat de la jointure entre t\_bilan et r\_id\_all.

```

T_Observateurs AS(
select
t.id_perm,
tmp.id_obs_v,
tmp.orgObs,
tmp.identObs
FROM export.r_id_all i
  JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
  LEFT JOIN tmp ON (t.id_obs_v = tmp.id_obs_v)
WHERE i.presence =1::boolean
UNION
select
t.id_perm,
tmp.id_obs_v,
tmp.orgObs,
tmp.identObs
FROM export.r_id_all i
  JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
  LEFT JOIN tmp ON (t.id_obs_v = tmp.id_obs_v)
WHERE i.presence =1::boolean)
```

### NivValide :

Le dernier élément à traiter avant de réunir ces tables temporaires est la catégorisation de la validité des données. Une jointure avec t\_bilan et un CASE sont utilisés pour répondre au standard INPN :

```
-----
CASE
    WHEN t.valid = 'Valide' THEN '1'
    WHEN t.valid = 'A confirmer' THEN '2'
    WHEN t.valid = 'Invalide' THEN '4'
    WHEN t.valid = '' OR t.valid IS NULL THEN '6' --En cas de vide 'non évalué'
    ELSE t.valid
END nivValid
```

Les clauses WITH sont ensuite réunies pour créer la table ST\_Principal.

```
SELECT
    c.id_perm as cleObs,
    T_statSource.statsource,
    T_orgagest.orggestdat,
    T_dsPublic.dspublic,
    T_StatObs.statobs,
    T_inchanged.cdref,
    T_inchanged.cdnom,
    T_vTAXREF.vtaxref,
    T_inchanged.nomcite,
    T_Denbrmin.denbrmin,
    T_Denbrmax.denbrmax,
    T_objdenbr.objdenbr,
    T_dateDebut.datedebut,
    T_dateFin.datefin,
    c.id_perm as cleObjet,
    T_natObjGeo.natobjgeo,
    T_diff.difflopre,
    T_diff.diffmaille,
    T_diff.diffcom,
    T_diff.difftotale,
    T_diff.floutage,
    T_Observateurs.identobs,
    T_Observateurs.orgObs,
    T_nivValide.nivvalid

FROM T_cleObs as c
JOIN T_statSource ON (T_statSource.id_perm = c.id_perm)
JOIN T_orgagest ON (T_orgagest.id_perm = c.id_perm)
JOIN T_dsPublic ON (T_dsPublic.id_perm = c.id_perm)
JOIN T_StatObs ON (T_StatObs.id_perm = c.id_perm)
JOIN T_vTAXREF ON (T_vTAXREF.id_perm = c.id_perm)
JOIN T_Denbrmin ON (T_Denbrmin.id_perm = c.id_perm)
JOIN T_Denbrmax ON (T_Denbrmax.id_perm = c.id_perm)
JOIN T_objdenbr ON (T_objdenbr.id_perm = c.id_perm)
JOIN T_dateDebut ON (T_dateDebut.id_perm = c.id_perm)
JOIN T_dateFin ON (T_dateFin.id_perm = c.id_perm)
JOIN T_natObjGeo ON (T_natObjGeo.id_perm = c.id_perm)
JOIN T_diff ON (T_diff.id_perm = c.id_perm)
JOIN T_inchanged ON (T_inchanged.id_perm = c.id_perm)
LEFT JOIN T_Observateurs ON (T_Observateurs.id_perm = c.id_perm)
JOIN T_nivValide ON (T_nivValide.id_perm = c.id_perm)
```

## b) Générer et exporter la table EN.

Ensuite, avec la fonction COPY de postgres, il est possible de faire un export directement en CSV, avec la définition des entités à sélectionner, du chemin où enregistrer l'export, et enfin du séparateur CSV.

Dans la mesure où l'export concerne toute la table EN, l'intégralité de celle-ci est sélectionnée. Un point important à noter est que l'export doit être fait dans un répertoire où Postgres a les droits, et que les disques virtuels sont inopérants. D'où l'export dans le disque D . Enfin, le délimiteur CSV est un point-virgule, en adéquation avec le standard demandé.

```
Copy (Select * From export.e_St_EN) to 'D:\sig\00_MODULE_EXPORT_BDCEN\exports\St_EN.csv'
With DELIMITER ';' CSV HEADER
```

### c) Générer une table d'attributs supplémentaires.

La table d'attributs supplémentaires a pour vocation d'être souple, mais dispose quand même d'une structure précise :

```
DROP TABLE IF EXISTS export.e_St_AttrAdd;
CREATE TABLE export.e_St_AttrAdd--creation de table d accueil
(
  CleObs character varying(100),
  NomParam character varying(45),
  Parametre character varying(45),
  typeParam character varying(45),
  Valeur character varying(45),
  unitParam character varying(45)
)
```

e_St_AttrAdd
CleObs varchar(100)
NomParam varchar (45)
Parametre varchar(45)
typeParam varchar(45)
Valeur varchar(45)
unitParam varchar(45)

« cleObs » contient des id\_perm pour faire le lien avec la table St\_Principal.

« NomParam » contient le nom de l'attribut supplémentaire.

« Parametre » en faite la description.

« TypeParam » indique de s'il s'agit d'un paramètre quantitatif ou qualitatif.

« Valeur » contient les valeurs dans les paramètres.

« UnitParam » est remplie en cas de critères quantitatif pour décrire le système d'unité utilisé.

La table est remplie avec des INSERT, spécifiques à chaque types de paramètres. Le système de jointure r\_id\_all et t\_bilan est de nouveau utilisé, à l'exception notable que tout les paramètres passés sauf Unite ne concernent que la faune.

Le sexe, l'âge et le comportement sont passés selon le modèle suivant :

```
--SEXE
INSERT INTO export.e_St_AttrAdd(CleObs,NomParam,Parametre,TypeParam,Valeur)
-- seule la faune est sexuée
(SELECT i.id_perm, 'sexe' AS nomParam, 'Sexe de 1 individu' AS Parametre, 'QUAL' AS TypeParam, t.sexe
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
AND t.sexe IS NOT NULL
AND t.sexe != '')
```

Pour le comportement, le champ comportement de t\_bilan est utilisé, pour l'âge age. Les champs nuls ou vides sont exclus pour ne pas surcharger inutilement le fichier d'attributs supplémentaires. Aussi pour unite, le procédé est le même mais une union est faite entre les données de la faune et de la flore.

Puis un COPY est réalisé de la même manière que pour St\_EN.csv.



## D) Générer des tables référencées géographiquement.

Le modèle INPN requiert de générer des tables au format shp. La fonction export INPN créer le modèle de ces fichiers sous la forme de tables dans PostgreSQL. Puisque les logiciels SIG les plus utilisés ne peuvent ouvrir qu'un seul type de géométrie à la fois, il doit y avoir un fichier par type de géométrie.

### Créer la table des polygones.

Cette table nécessite de réunir plusieurs sources : celles des polygones issus des couches sig, ainsi que la géométrie des sites et des secteurs moissonnés. Plusieurs clauses WITH sont utilisées pour parvenir à cet effet.

Tout d'abord une reconstitution des sitep (un niveau de précision) est généré

```
--  
DROP TABLE IF EXISTS export.e_st_sig_poly;  
CREATE TABLE export.e_st_sig_poly AS(  
WITH geo_site AS (--reconstitution des sitep avec union  
    SELECT geo_site.codesitep,  
           st_union(geo_site.geom) AS geom  
    FROM bd_site_cen.geo_site  
    GROUP BY geo_site.codesitep)
```

*Illustration 16: Reconstitution du niveau de précision codesitep*

Ensuite, un set de données est crée en prenant les id\_perm, secteurs\_id, codesitep, géométries et types d'objets des entités grâce à une double jointure r\_id\_all / t\_bilan / sig\_fxxxx\_poly.

```
r_all_poly AS(  
-- Set réduit & compact r_  
  
SELECT i.id_perm, t.secteur_id, t.codesitep, p.geom, t.type_object  
FROM export.r_id_all i  
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm  
JOIN bdfauneflore.sig_faune_poly p ON p.id_perm = i.id_perm  
WHERE i.presence = 1::boolean  
UNION  
-----doubler  
SELECT i.id_perm, t.secteur_id, t.codesitep, p.geom, t.type_object  
FROM export.r_id_all i  
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm  
JOIN bdfauneflore.sig_flore_poly p ON p.id_perm = i.id_perm  
WHERE i.presence = 1::boolean)
```

Une dernière clause WITH , « temp\_st\_sig », permet de faire l'union de toute les entités polygonales : celles issues des couches sig, celles des secteurs et celles au niveau d'échelle codesitep.

Il n'y a  
plus  
qu'à  
remplir  
la table

```
temp_st_sig_poly AS(
select
  id_perm as cleObjet,
  geom as geom
  from r_all_poly
  where r_all_poly.type_object like 'sig_%poly%'
union
select s.secteur_id as cleObjet,
s.geom as geom
from r_all_poly r join bdfauneflore.secteur s on r.secteur_id = s.secteur_id
group by s.secteur_id, s.geom

union

select
s.codesitep as cleObjet,
s.geom
from r_all_poly r join geo_site s on r.codesitep = s.codesitep
where r.secteur_id is null or r.secteur_id = ''
group by s.codesitep, s.geom)
```

*Illustration 17: Rassemblement de toutes les entités polygonales*

e\_st\_sig\_poly :

```
SELECT te.cleobjet, 'false' as objCompos, te.geom FROM temp_st_sig_poly te)
```

```
e_St_sig_poly
cleobjet varchar(100)
objcompos unknown
geom Geometry
```

L'opération est simplifiée pour les lignes et points qui ne prennent pas en compte les lignes et secteurs, mais uniquement leur propre type. La jointure de type r\_id\_all / t\_bilan / sig\_flore\_line remplit son office.

```
--g_st_sig_point
DROP TABLE IF EXISTS export.e_st_sig_point;
CREATE TABLE export.e_st_sig_point AS(
SELECT i.id_perm as cle_objet, 'false' AS objCompos, p.geom
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_faune_point p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm as cle_objet, 'false' AS objCompos, p.geom
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_flore_point p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean)
```

La fonction `plpgsql` s'achève ici, mais les fichiers *shapefile* non pas encore été générés. Le plugin Python est en charge de déléguer la création de ces *shapefiles* à Ogr2Ogr en se basant sur les tables `e_st_sig`.

## E) Générer St\_DESCR

```
-- CSV St_Descr
DROP TABLE IF EXISTS descr_fa;
CREATE TEMP TABLE descr_fa AS(
SELECT
i.id_perm as cleObs,

--
CASE
WHEN t.methode IS NULL OR t.methode = ''
THEN 'Inconnu'::character varying(45)
ELSE t.methode::character varying(45)
END AS obsMeth,

--

CASE
WHEN t.age IN('MORT','Mort') or t.comportement IN('Mort','mort')
THEN 'Trouvé mort'::character varying(45)
WHEN (t.age NOT IN('MORT','Mort') or t.comportement NOT IN ('Mort','mort'))
AND (t.comportement IS NOT NULL OR t.comportement <> '')
THEN 'Observé vivant'::character varying(45)
ELSE 'NSP'::character varying(45)
END AS ocEtatBio,

--

'Inconnu'::character varying(45) AS ocNat,

--

CASE
WHEN t.sexe = 'M' or t.sexe = 'm'
THEN 'Mâle'::character varying(45)
WHEN t.sexe = 'F' or t.sexe = 'f'
THEN 'Femelle'::character varying(45)
WHEN t.sexe = 'MF'
THEN 'mixte'::character varying(45)
ELSE 'Inconnu'::character varying(45)
END AS ocSexe,

--

--ocStade (age)

--reste a régler des sygles.

CASE
WHEN t.age = 'CHENILLE' OR t.age = 'CHE'
THEN 'Chenille'::character varying(45)
WHEN t.age = 'JUV' or t.age = 'JUV'
THEN 'Juvénile'::character varying(45)
WHEN t.age IN ('AD',' AD', 'Adulte')
THEN 'Adulte'::character varying(45)
```

```

'NSP'::character varying (45) AS preuvOui-- se baser sur methode? determination?
,

--
-- obsCtx
--

CASE WHEN t.cadre IS NOT NULL AND t.cadre <> '' THEN t.cadre::character varying (45)
ELSE 'NSP'::character varying (45)
END AS obsCtx

FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean)
:
    WHEN t.presabs = '-1' THEN 'Introduit'::character varying(45)
    ELSE 'Non renseigné'::character varying(45)
END AS ocBioGeo,

--ocStatBio

CASE

    WHEN t.comportement in ('vol migratoire','vol migratoir', 'vol ; cri', 'vol local', 'vol + repos', 'vol ; i
    THEN 'Passage en vol'::character varying(45)
    WHEN t.comportement in ('vol nuptial','Parade nuptiale','accouplement et ponte',
    'tandem ponte', 'accouplement', 'parade nuptiale', 'tandem + vol', 'parade', 'reproduction',
    'tandem + vol + repos', 'tandem + ponte', 'parade ; vol', 'ponte', 'tandem ; ponte','tandem', 'Gestante')
    THEN 'Reproduction'::character varying(45) -- accouplement, parades et pontes
    WHEN t.comportement in ('alimentation', 'transport de nourriture', 'butinage', 'chasse',
    'nourrissage')
    THEN 'Chasse / alimentation'::character varying(45) -- accouplement, parades et pontes
    ELSE 'Inconnu'::character varying(45)
END AS ocStatBio,

--
--preuvOui
--

```

La table ST\_Descr comprend des attributs spécifiques aux informations relatives aux individus observés. Il s'agit de la section où les choix les plus arbitraires ont été fait. Et à l'instar de la table St\_Principal, il a parfois fallu se baser sur plusieurs tables du CENPic pour se conformer à un champ, c'est par exemple le cas pour déterminer la mort d'un individu qui peut-être renseigné au CENPic dans son état, son comportement ou même son âge. Il a fallu tâtonner et explorer la base pour avoir le script le plus juste possible.

C'est la table ocStatBio qui comporte les choix qui pourraient être le plus facilement remis en question. Les comportements d'espèces observées peuvent être variés, et après avoir balayé tout les types de comportement de la base du CENPic, les adaptations ont été faites. Parfois plusieurs comportements ont été observés, aors le premier inscrit est gardé pour faire la conversion : c'est le cas par exemple de « parade;vol » qui a été classé dans la catégorie reproduction plutôt que passage en vol.

## Kill\_table()

### Pas d'arguments.

Cette fonction supprime toutes les tables qui ne sont pas permanentes. Le système de nomenclature de tables mis en place permet de réaliser ce traitement de façon stable et facilement.

Le plpgsql offre la possibilité de faire des boucles, comme dans un vrai langage de programmation. Cette boucle indique qu'il faut supprimer dans le schéma d'export toutes les tables du schéma qui ne commencent pas par w, z, ou p. Les suppressions doivent se faire en cascade pour assurer la suppression des dépendances.

```
declare row record;

BEGIN

for row in
    select *
    from pg_tables p
    where p.schemaname = 'export'
    and tablename not ilike 'w_%'
    and tablename not ilike 'z_%'
    and tablename not ilike 'p_%'
loop

    begin
        raise notice ' requete effectué sur(%)', row.tablename;
        execute 'DROP TABLE export.' || row.tablename || ' cascade';

    end;
```

*Illustration 18: Une boucle PGPSQL: Suppression des tables régénérées*

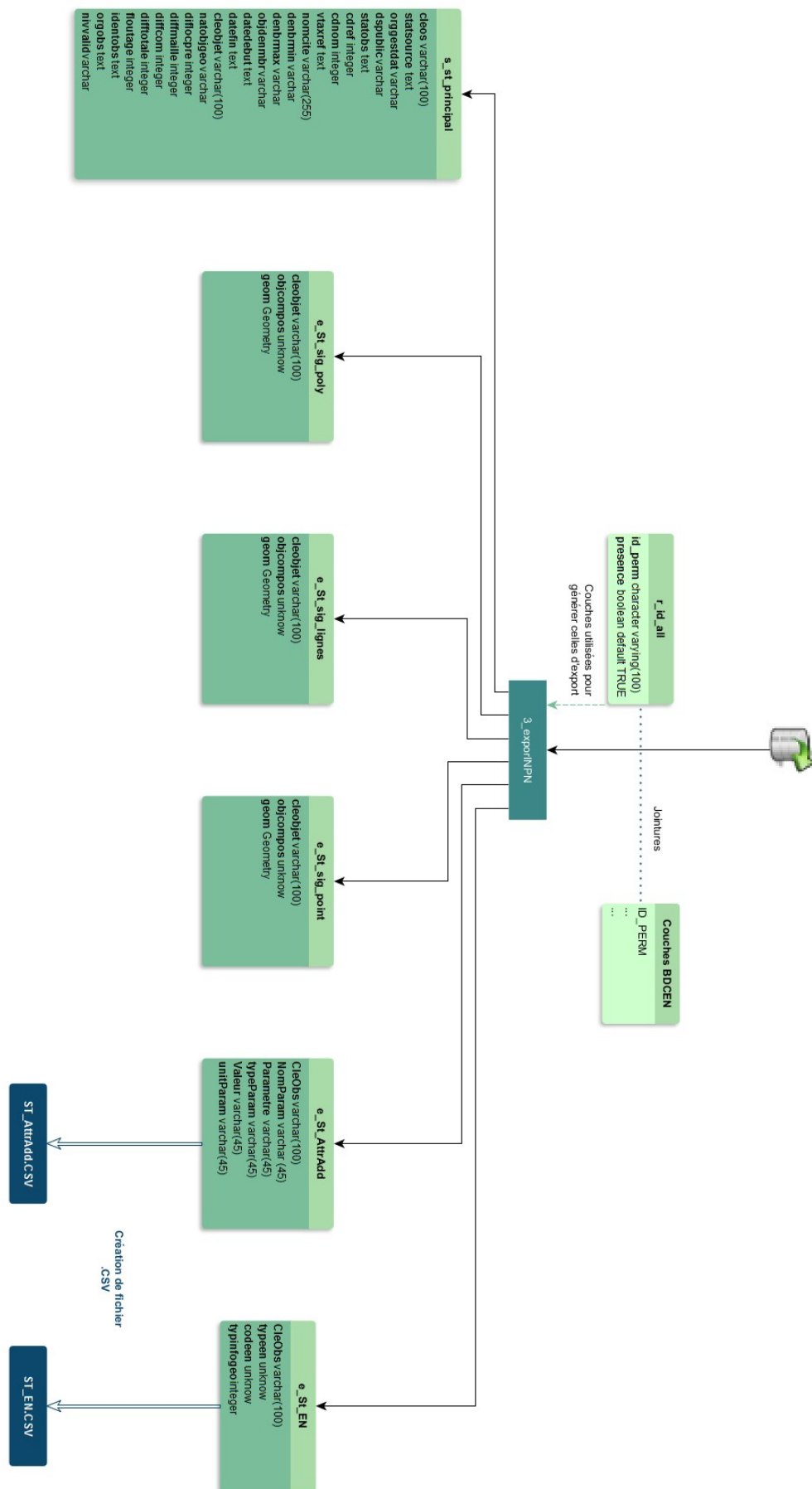


Illustration 19: Fonctionnement de la fonction d'export INPN

## Structure du code python du module d'export.

Le plugin en tant que tel a pour objectif d'être l'interface entre l'utilisateur, Qgis et la base de donnée. Il doit donc contenir des fonctions d'affichage, mais aussi de communications avec les autres technologies.

Le code est également organisé de façon basique, avec une classe par fichier. Chacune de ces classes contient un certain nombre de méthodes, qui ont dans la mesure du possible été séparées ostensiblement avec des blocs de commentaire. Aussi, les fonctions elles même sont généralement commentées. La documentation se focalise sur l'effet des fonctions plus que sur leur construction. En cas d'éléments plus complexes des pauses explicatives seront effectuées. COEUR

## Libraires utilisées

Dans un soucis de facilité de déploiement, toutes les libraires utilisées sont incluses dans la version de Python installée avec Qgis.

```
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
from PyQt4.QtGui import QAction, QIcon, QMessageBox, QTableWidgetItem
# Initialize Qt resources from file resources.py
import resources
# Import the code for the dialog
from ModuleExport_dialog import ModuleExportDialog
from DialogSelectFaune import DialogSelectFaune
from DialogSelectFlore import DialogSelectFlore
import os.path
import time
import psycpg2
from PyQt4.Qt import QPushButton, QSqlDatabase, QSqlTableModel
from xml.dom import minidom #Pour pouvoir générer notre fichier de metadata.
from subprocess import call
```

*Illustration 20: Les libraires Python utilisées dans le module d'export*

permet de se connecter à PostgreSQL, d'envoyer des requêtes et de recevoir des résultats. Les identifiants de connexion sont directement récupérés grâce à des fonctions de PyQgis.

- **PyQgis** : utilisé pour récupérer des informations sur les couches présents dans le Canevas. Surtout utilisé pour faire des structures de contrôle. Son utilisation est réduite au minimum au profit de PostgreSQL.
- **PyQt** : utilisé pour communiquer avec la librairie Qt, écrite en C. Qt est la librairie utilisée par Qgis.
- **Minidom** : Module inclus à l'installation de python utilisé pour générer une feuille XML de métadonnée.
- **OS** : Utilisé dans le plugin pour naviguer dans les arborescences de répertoire. Permet notamment de normer des chemins et de vérifier leur existence.
- **Time** : librairie permettant d'utiliser des fonctions de temps. Elle est utilisée uniquement dans la génération du fichier de métadonnée.

• P  
s  
y  
c  
o  
p  
g  
2  
:  
:



- **Subprocess** : Cette librairie est utilisée avec call dans ce plugin pour pouvoir envoyer des lignes de commandes au système d'exploitation.

## Organisation du code du plugin.

Un plugin Qgis en python dispose de plusieurs fichiers, et le plugin Builder permet de les générer sans plus avoir à se soucier d'eux par la suite. C'est par exemple le cas de `__init__.py`. Les fichiers réalisés par le CENPic sont les suivants :

- **ModuleExport.py** : le coeur du plugin, avec les fonctions d'initialisation, contrôle, lancement, interaction avec Qgis et la base de donnée.
- **ModuleExport\_dialog.py** : Contient les éléments graphiques de la fenêtre principale du plugin. La majeure partie est importée depuis le fichier `.ui`. Il y a également du code pour relier les boutons à des fonctions.
- **DialogSelectFlore.py** : Contient les éléments graphiques de la fenêtre de sélection d'espèces, importés depuis un fichier `ui`. Mais la classe dans ce fichier contient également des éléments inhérents au fonctionnement du plugin, avec par exemple les méthodes d'ajout et de retrait d'espèces dans la liste `flore_select`.
- **DialogSelectFaune.py** : Réplique de `DialogSelectFlore` pour la faune.

```
class ModuleExport

def
__init__(self, iface)
tr(self, message)
add_action(self, icon_path, text, callback,...)
initGui(self)
unload(self)
peuplerComboExport(self)
get_table_name(self)
get_db_name(self)
get_host_ip(self)
get_dbuser_name(self)
get_pswrd(self)
recup_id(self)
fenetre_select_faune(self)
fenetre_select_flore(self)
vider_la_table(self, typesuppr)
ajout_ls_espece_pg(self, typeadd)
va_chercher(self, typecherche)
_1_requeteGenerale(self)
_2_moissonneuse_bateuse(self)
_3_Fichier_CSV(self, typeExport)
_4_Export_SIG(self, typeExport)
_5_Metadonne(self)
nettoyage_des_tables(self)
nettoyage_dossier_export(self)
lancement_des_operations(self)
run(self)
```

```
class ModuleExportDialog

def
__init__(self, iface)
griser_bouton(self, monbouton)

class DialogSelectFlore

def
__init__(self, iface)
visibilite(self, bool = False)
transvasajout_espece_flore(self)
transvasretrait_espece_flore(self)
clear(self)
peupler_table_from_base(self, tableurcible, iter)
peupler_table_from_base_selected(self, tableurcible, iter)

class DialogSelectFaune

def
__init__(self, iface)
visibilite(self, bool = False)
transvasajout_espece_faune(self)
transvasretrait_espece_faune(self)
clear(self)
peupler_table_from_base(self, tableurcible, iter)
peupler_table_from_base_selected(self, tableurcible, iter)
```

### Choix du destinataire

labelibele

Intitulé de l'export

libele

labelstruc

Structure du destinataire

sstructure

labeldest

Destinataire

destinataire

### Choix du type de sélection temporelle

☒ radiosaisie

☐ Date de Saisie

☐ radioObs

☐ Année d'observation

Date minimale de prise en compte

01-01-2000

date\_min\_pec

Data maximale de prise en compte

01-01-2000

date\_max\_pe

### Paramètres de moissonnage

☒ radioExt

☐ Inclure les données externes

☐ radioInt

☐ Exclure les données externes

☐ Uniquement les objets Valides

☐ Uniquement les objets présent

obj\_valide

obj\_present

### Sélection des espèces prises en compte

☐ rad\_all\_espece

☐ Toutes les espèces

☒ rad\_slc\_espece

☒ Sélectionner des espèces



Ajouter de la flore



Ajouter de la faune.

ButtonFlore

ButtonFaune

☐ Toute la flore

checkflore

☐ Toute la faune

checkfaune

### Choix du type d'export

Choix de la méthode d'export

export brut

comboExport

killButton

Nettoyer les tables

☐ checkClean

Nettoyage automatique

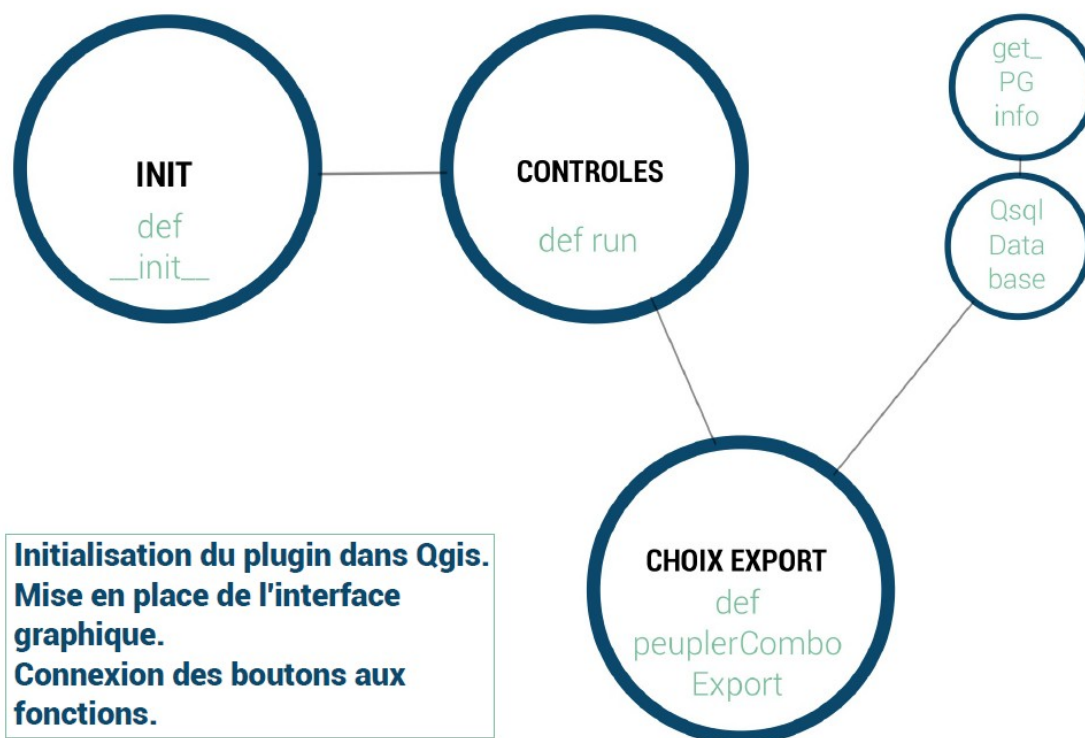
ButtonCleanExport

Nettoyer le dossier d' export

Exporter

ButtonLaunch

## Initialiser le plugin



*Illustration 21: Schéma de lancement du plugin*

L'initialisation charge l'interface graphique depuis le gui, et des images sont rajoutées en lignes de code. Dans `def __init__` Le lien entre les boutons et les fonctions associées a été établi avec l'utilisation de lambda, qui permet de passer une fonction qui ne pointe pas vers un « SLOT » Qt.<sup>1</sup>

La `self.dlg_Flo.btn_visible.clicked.connect(lambda : self.dlg_Flo.visibilite(True))` fonction clicked indique que l'action connectée sera effectuée en cas de clic de l'utilisateur.

Les fonctions esthétiques font appel à la méthode `setIcon`, et récupèrent le chemin des images.

```
self.dlg.ButtonFaune.setIcon(QIcon(self.plugin_dir+"/turtle-128.png"))
```

Qt est une librairie souple qui permet aussi l'utilisation de feuilles de styles en cascade (CSS). La méthode `setStyleSheet` est appelée :

```
self.buttonLaunch.setStyleSheet('QPushButton {background-color: #CCF390;font-weight: bold;}')
```

Ces opérations sont réalisées lors du lancement de Qgis, et l'utilisateur dispose de la possibilité de vraiment lancer le plugin en cliquant sur l'icône du plugin, qui lance la fonction `run`. La série de

<sup>1</sup> Les objets Qt peuvent émettre avec SIGNAL et ont des fonctions de réception de signaux pré codées : les SLOT. Pour plus d'informations : <http://doc.qt.io/qt-4.8/signalsandslots.html>

contrôle qui s'assure que l'utilisateur soit sur la bonne couche s'enclenche à ce moment, en faisant appel à *PyQgis* avec l'appel d'*Iface*. *ActiveLayer* permet de récupérer la couche sélectionnée, et d'ainsi pouvoir l'interroger. En cas de non satisfaction au contrôle, une *QMessageBox* Qt est renvoyée à l'utilisateur et un *return* casse le déroulement des opérations.

```
def run(self):
    #CONTROLE : verifier si une couche est bien sélectionnée
    if self.iface.activeLayer() == None :
        QMessageBox.information(self.iface.mainWindow(), 'Information', u'Veuillez sélectionner une couche à connecter')
        return
    else :
        #CONTROLE : : Si couche sélectionné est un raster => exit
        #MODIF LOIC
        if self.iface.activeLayer().type() == 2 or self.iface.activeLayer().type() == 1:
            QMessageBox.information(self.iface.mainWindow(), 'Information', u'Veuillez sélectionner une couche vecteur')
            return

    #CONTROLE : verifier si des objets sont sélectionnés
    if self.iface.activeLayer().selectedFeatureCount() <= 0 :
        QMessageBox.information(self.iface.mainWindow(), 'Information', u'Vous devez sélectionner au moins un objet')
        return

    #CONTROLE : Si ma couche ne s'appelle pas z_export, alors erreur.
    if self.iface.activeLayer().name() != u"z_export" :
        QMessageBox.information(self.iface.mainWindow(), 'Information', u'Vous devez sélectionner la couche z_export')
        return
```

Illustration 22: Les structures de contrôles au lancement du plugin.

Avant d'afficher la fenêtre de dialogue principale, une autre action est effectuée : La récupération des types d'export possibles, et leur insertion dans *comboExport*.

Les types d'export sont stockés dans une table dans PostgreSQL, il est donc nécessaire d'établir une connexion vers PostgreSQL. Le plugin d'export reprend le même fonctionnement que le plugin connecteur avec les fonctions « get »

```
def get_db_name(self):
    #Fonction pour obtenir le nom de la base de données source
    madb = str(self.iface.activeLayer().source())
    madb = madb.replace(madb[0:madb.find('dbname=')+7], "").replace("'", '').replace('"', '').split()
    return madb[0]
```

Illustration 23: Les fonctions de récupérations de connexion PG

Les informations de connexions sont ensuite passées dans la fonction *peuplerComboExport* qui s'appuie sur la librairie *QsqlDatabase* de Qt, qui permet de réaliser en très peu de lignes un objet graphique connecter à la base grâce au système de modèle/vue spécifique à Qt. Ce système crée un intermédiaire entre la donnée et les interactions de l'utilisateur.

#### w\_type\_export

**nom** varchar(255)  
**fonction** varchar(255)  
**ogr2ogr** varchar  
**id** sequence

La

```
def peuplerComboExport(self):  
    #creation connexion PGQT pour peupler la liste des type_export.  
    self.QConn = QSqlDatabase("QPSQL")  
    self.QConn.setHostName(self.get_host_ip())  
    self.QConn.setDatabaseName(self.get_db_name())  
    self.QConn.setUserName(self.get_dbuser_name())  
    self.QConn.setPassword(self.get_pswrd())  
    self.QConn.open()  
  
    #Creation du modele de donnée pour peupler laQcombobox.  
    self.Model_export = QSqlTableModel(db = self.QConn)  
    self.Model_export.setTable("export.w_type_export")  
    self.Model_export.select()  
    #Peupler la Qcombobox  
    self.dlg.comboExport.setModel(self.Model_export)  
    self.dlg.comboExport.setModelColumn(self.Model_export.fieldIndex("nom"))
```

*Illustration 24: Connecter une comboBox à une table PGSQL.*

dernière ligne de cette capture indique que les éléments affichés sont ceux de la colonne « nom ».

## Dynamisme de la fenêtre principale.

Le plugin dispose de fonctions qui permettent d'adapter l'affichage des données en fonctions des éléments cochés dans la fenêtre. C'est le cas pour la saisie des dates ainsi que pour le choix des espèces. Dans le cas des espèces, les boutons de choix d'espèces sont par défaut en mode Hide, c'est à dire qu'ils sont cachés dans l'interface, mais néanmoins chargés en mémoire. Pour les afficher, il faut donc envoyer un signal inverse au hide, c'est à dire show (montrer). Les boutons radio rad\_slc\_espece et rad\_all\_espece sont donc connectés aux boutons d'ajout de faune flore pour afficher ou cacher les boutons.

```
self.rad_all_espece.toggled.connect(lambda: self.checkFlore.hide())
self.rad_all_espece.toggled.connect(lambda: self.checkFaune.hide())
self.rad_slc_espece.toggled.connect(lambda: self.ButtonFaune.show())
self.rad_slc_espece.toggled.connect(lambda: self.ButtonFlore.show())
```

*Illustration 25: Afficher et cacher des éléments graphiques dans Qt: Le choix des espèce.*

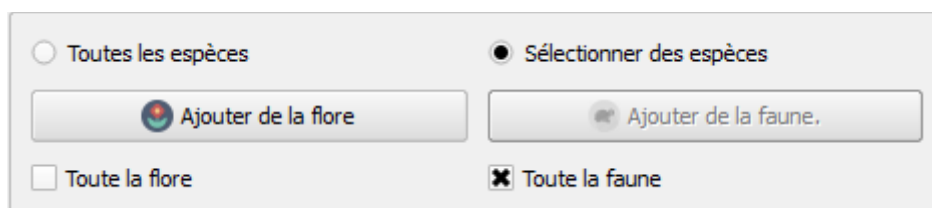
Une logique similaire est utilisée pour griser et donc empêcher la sélection d'espèces particulières si l'on veut toute les espèces d'une famille.

```
self.checkFlore.clicked.connect(lambda: self.griser_bouton(self.ButtonFlore))
self.checkFaune.clicked.connect(lambda: self.griser_bouton(self.ButtonFaune))
```

*Illustration 26: Appel des fonctions pour griser un bouton*

```
def griser_bouton(self, monbouton):
    if monbouton.isEnabled() == True:
        return monbouton.setEnabled(0)
    else:
        return monbouton.setEnabled(1)
```

*Illustration 27: Fonction locale avec*



*Illustration 28: Exemple de bouton grisé*

Pour ce qui est des fonctions de dates, si l'export porte sur les dates de Saisie, alors on a une date au format jj-MM-yyyy, et sur une date d'observation yyyy. Ceci est réalisé grâce à l'appel de la fonction `setDisplayFormat`, qui nécessite en paramètre le format souhaité.

```
self.date_min_pec.setDisplayFormat('yyyy')
self.date_max_pec.setDisplayFormat('yyyy')
self.radioSaisie.toggled.connect(lambda: self.date_min_pec.setDisplayFormat('dd-MM-yyyy'))
self.radioSaisie.toggled.connect(lambda: self.date_max_pec.setDisplayFormat('dd-MM-yyyy'))
self.radioObs.toggled.connect(lambda: self.date_min_pec.setDisplayFormat('yyyy'))
self.radioObs.toggled.connect(lambda: self.date_max_pec.setDisplayFormat('yyyy'))
```

Les boutons radios des dates sont connectées au signal qui change le format, et les dates sont par défaut au format yyyy.

## Ajouter des espèces.

Le fonctionnement étant symétrique entre la faune et la flore, l'exemple porte uniquement sur la flore.

Si l'utilisateur désire ajouter toutes les espèces de la faune ou de la flore, il lui faut cocher la checkbox checkfaune ou checkflore. Sinon, pour obtenir des espèces précises, c'est sur ButtonFlore qu'il faut cliquer.

The interface is titled "Recherche:" and includes a search bar labeled "searchBar" with a "Go" button labeled "searchButton". To the right of the search bar is a dropdown menu labeled "taxon" and two buttons: "comboSearch" and "btn\_invisible". Below the search bar is a button labeled "btn\_visible".

The main section is titled "Liste d'espèces (wref)" and contains a table with the following data:

	espece_id	taxons	nomcommun
1	55814	Puccinia rubi DC.	None
2	55797	Phragmidium rosarum f. rosae-pimpinellifoliae Rabhenhorst	None
3	57551	Salix alba L. × S. viminalis L.	Saule (hybride)
4	57571	Brachythecium velutinum (Hedw.) Schimp.	None
5	57572	Campylium chrysophyllum (Brid.) Lange	None
6	55799	Phragmidium rubi-saxatilis Liro	None
7	55795	Phragmidium rosae-alpinae (DC.) G. Winter	None

Below the table is a panel titled "Espèces sélectionnées" with four buttons: 1 (red circle with 'x'), 2 (yellow circle with 'i'), 3 (red circle with 'x'), and 4 (green circle with '+').

The bottom section is titled "tab\_flore\_select" and contains a table with the following data:

	espece_id	taxon
1	3	Asplenium obo...

To the right of the table is a list of actions: 1. vider\_flore, 2. unselect, 3. btn\_rmfllore, 4. btn\_ajoutflore.

At the bottom right is a button labeled "buttonBox" and two buttons: "OK" and "Annuler".

L'organisation des tableaux est la partie qui est à la fois la plus verbeuse et la plus complexe du plugin. En effet, contrairement au cas de la comboExport, ces tableaux ne reposent pas sur le système Modèle / Vue mais sur le modèle classique d'éléments additionnés.

Les fonctions liées à la sélection de la flore sont divisées entre les classes ModuleExport et DialogSelectFlore. Dans cette deuxième classe se trouvent les fonctions d'affichage mais aussi de passage d'éléments d'un tableau à un autre, le dés emplissage d'un tableau ou la désélection d'éléments. Ces fonctions ont pour point commun ne de pas interroger la base de donnée. Les fonctions qui le font sont implantées dans la classe ModuleExport : Fonctions de recherches, de



suppressions d'éléments dans tab\_flore\_select.

Le schéma montre dans les grandes lignes le fonctionnement de la sélection d'espèces.

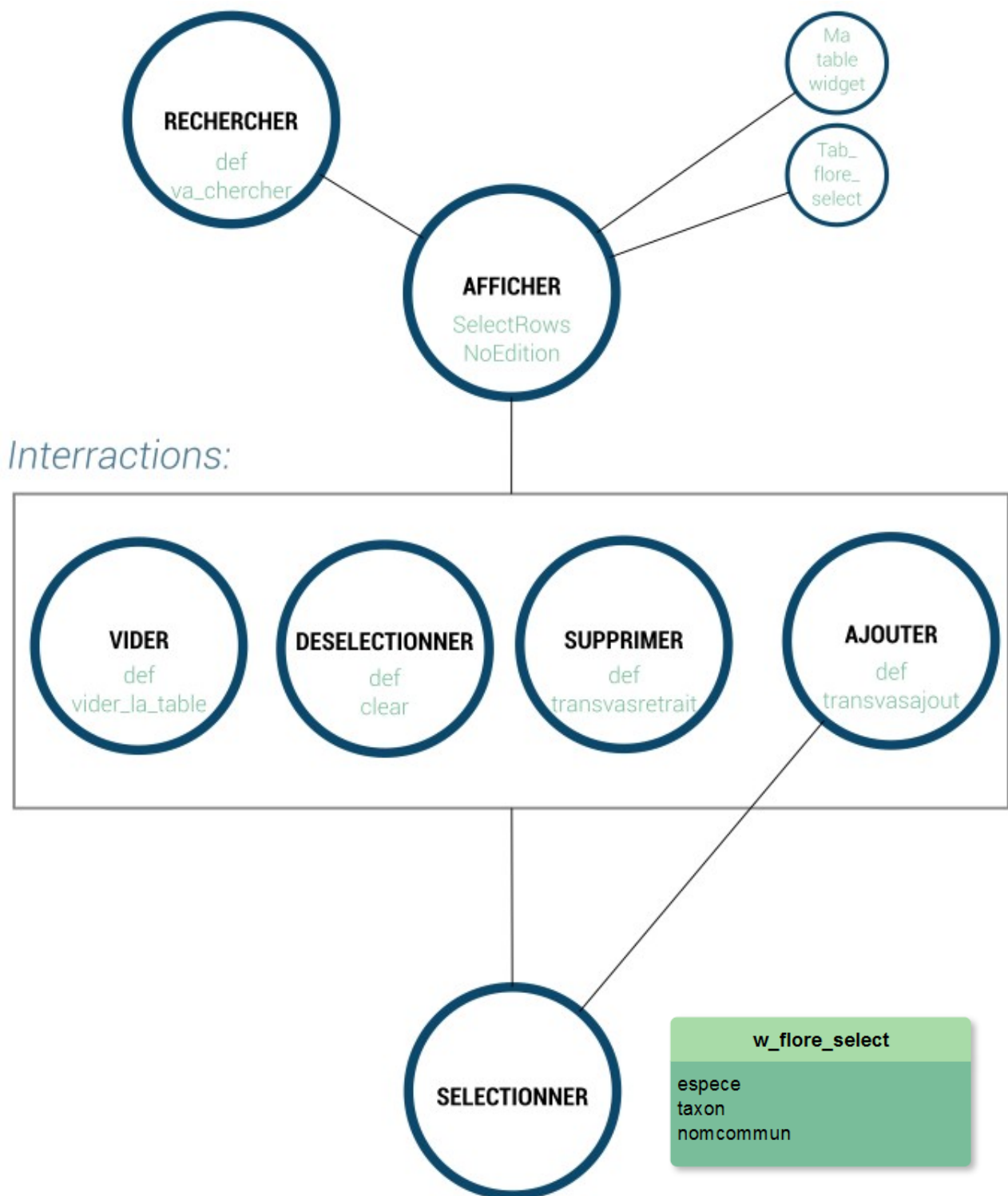


Illustration 29: Schéma simplifié de gestion d'espèces.

Le fait de ne pas passer par un système Modèle/View oblige à paramétrer certains détails manuellement, comme par exemple le nom des colonnes, qui reprennent pourtant la nomenclature de la table. Une liste d'en-tête est donc définie dans la classe de Dialogue, puis appliqué au *Widget* de table.

```
#définir les titres de mes colonnes
header_labels = ["espece_id", "taxons", "nomcommun", 'class', 'ordre', "cd_nom", "cd_ref", "rarete", "menace" ]
self.matablewidget.setHorizontalHeaderLabels(header_labels)
```

Les tables en présence dans la fenêtre ont pour vocation d'uniquement afficher les données et de permettre une sélection par ligne. Ces 2 lignes permettent de le faire.

```
#La selection sur la vue wref ne pourra se faire que par lignes
self.matablewidget.setSelectionBehavior(QAbstractItemView.SelectRows)
#Notre table ne doit pas être éditable
self.matablewidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
```

Les données dans le tableau supérieur sont directement prises dans la base de donnée, avec un échantillon d'une centaine d'individus par défaut. Pour remplir la table la fonction `peupler_table_from_base` est appelée dans une boucle où chaque itération dans la boucle crée une ligne dans le tableau. C'est notamment le cas pour fournir l'échantillon de données de base, généré dans la fonction `ModuleExport.fenetre_select_faune`. Les fonctions de récupération d'identifiant postgres sont à nouveau utilisées pour faire le lien avec la base. Puis `Pyscopg2` est utilisé pour passer une requête SQL et obtenir les éléments désirés :

```
idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(), self.get_host_ip(), se
conn = pyscopg2.connect(idconnexion)
cur = conn.cursor()
#fourniture d un echantillon basique et léger.
marequete = """
select w.espece_id, w.taxon, w.nomcommun, w.classe, w.ordre, w.cd_nom, w.cd_ref ,w.rarepic ,w.menapic
from bdfauneflore.w_ref_faune w
limit 100
"""
cur.execute(marequete)
```

Illustration 30: Requete SQL enrobée pour sélectionner un échantillon de 100 espèces

Une itération est ensuite effectuée sur la requête et appelle `DialogSelectFlore.peupler_table_from_base_selected` à chaque itération.

```
for i in cur:
    self.dlg_F.peupler_table_from_base_selected(self.dlg_F.tab_faune_selected,i)
```

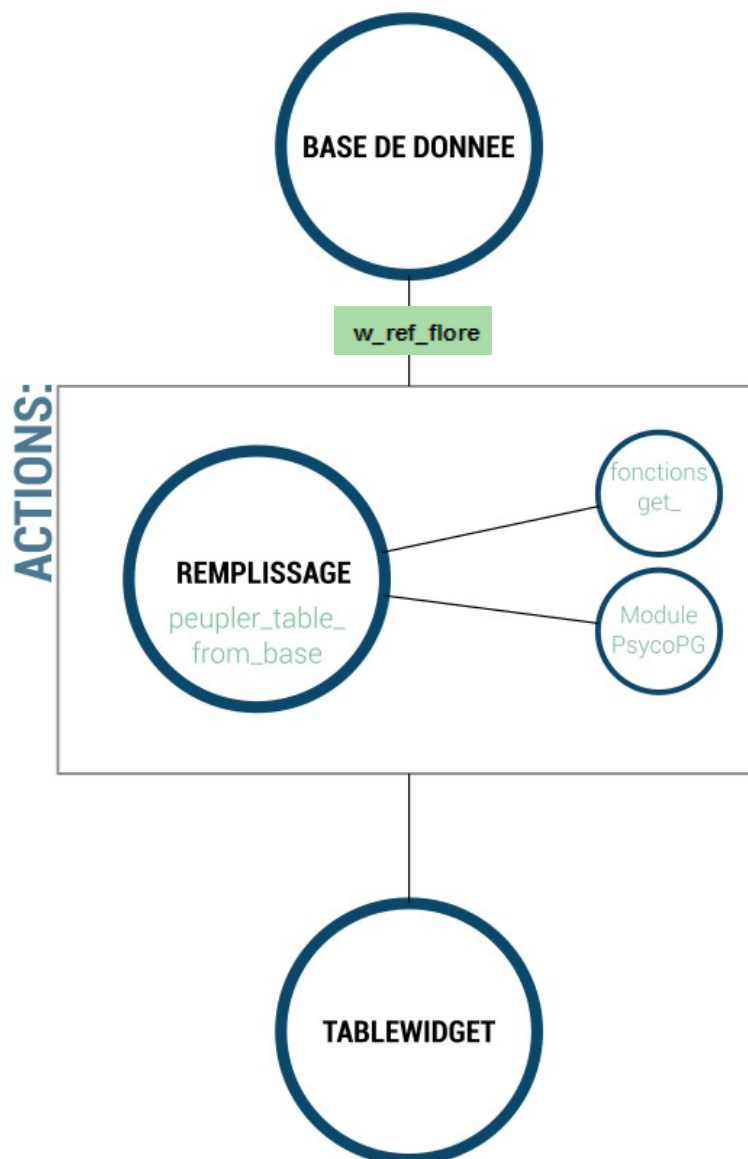
Illustration 31: Appel de la fonction d'ajout d'espèces dans le tableau de choix de sélection

La fonction appelée, insert une ligne (insertRow), puis effectue elle même une itération sur le nombre de champs, défini en amont : donc 9 champs.

```
#Param: Type de tableau cible, itération.
def peupler_table_from_base(self,tableurcible, iter):

    tableurcible.insertRow(tableurcible.rowCount())
    for i in range(0,9):
        tableurcible.setItem(tableurcible.rowCount()-1,i ,QTableWidgetItem(unicode(iter[i])))
```

La méthode setItem se charge ensuite de remplir les lignes cellules par cellules, en prenant soin de convertir le contenu en chaîne unicode pour assurer la compatibilité avec Qt.



*Illustration 32: Ajouter des éléments dans le tableau de sélection d'espèces supérieur.*

## Fonction de recherche :

Le fonctionnement est ainsi utilisé pour l'échantillon par défaut, mais aussi dans la fonction de recherche : *ModuleExport.va\_chercher*. Pour des raisons d'optimisation, la recherche ne se produit que s'il y a 3 ou plus caractères dans la barre de recherche. L'utilisateur peut également choisir dans quel champs il va faire sa recherche grâce à la *combobox comboSearch*. La fonction de recherche est rudimentaire avec l'utilisation d'ilike et d'un double joker.

```
marequete = """
select w.espece_id, w.taxon, w.nomcommun, w.classe, w.ordre, w.cd_nom, w.cd_ref ,w.rarepic ,w.menapic
from %s w
where %s%s%s iLIKE '%s'

""" % (ma_wref, 'CAST(', moncritere, ' AS TEXT)', marecherche)
# Le ILIKE permet une recherche insensible à la casse, mais augmente considérablement le temps de requetage.

cur.execute(marequete)

if typecherche == 'f1':
    for i in cur:
        self.dlg_Flo.peupler_table_from_base(self.dlg_Flo.matablewidget,i)
    #Trier les résultats par taxon.
    self.dlg_Flo.matablewidget.sortItems(1)
```

*Illustration 33: Fonctionnalité de recherche.*

Il est important de noter que l'appel du contenu des champs est converti à la volée en texte avec la fonction CAST.

## Sélectionner des espèces.

La sélection d'espèces s'opère par une opération de copiage d'éléments dans le tableaux du haut vers celui du bas. Mais puisque tout les éléments dans le tableau du bas sont également présent dans la table `w_flore_select`, tout ajout dans l'interface graphique doit également être fait dans la base de donnée si l'usager appuie sur le bouton OK. Des fonctions de contrôle de doublon et d'écriture dans la base de donnée sont donc implémentées.

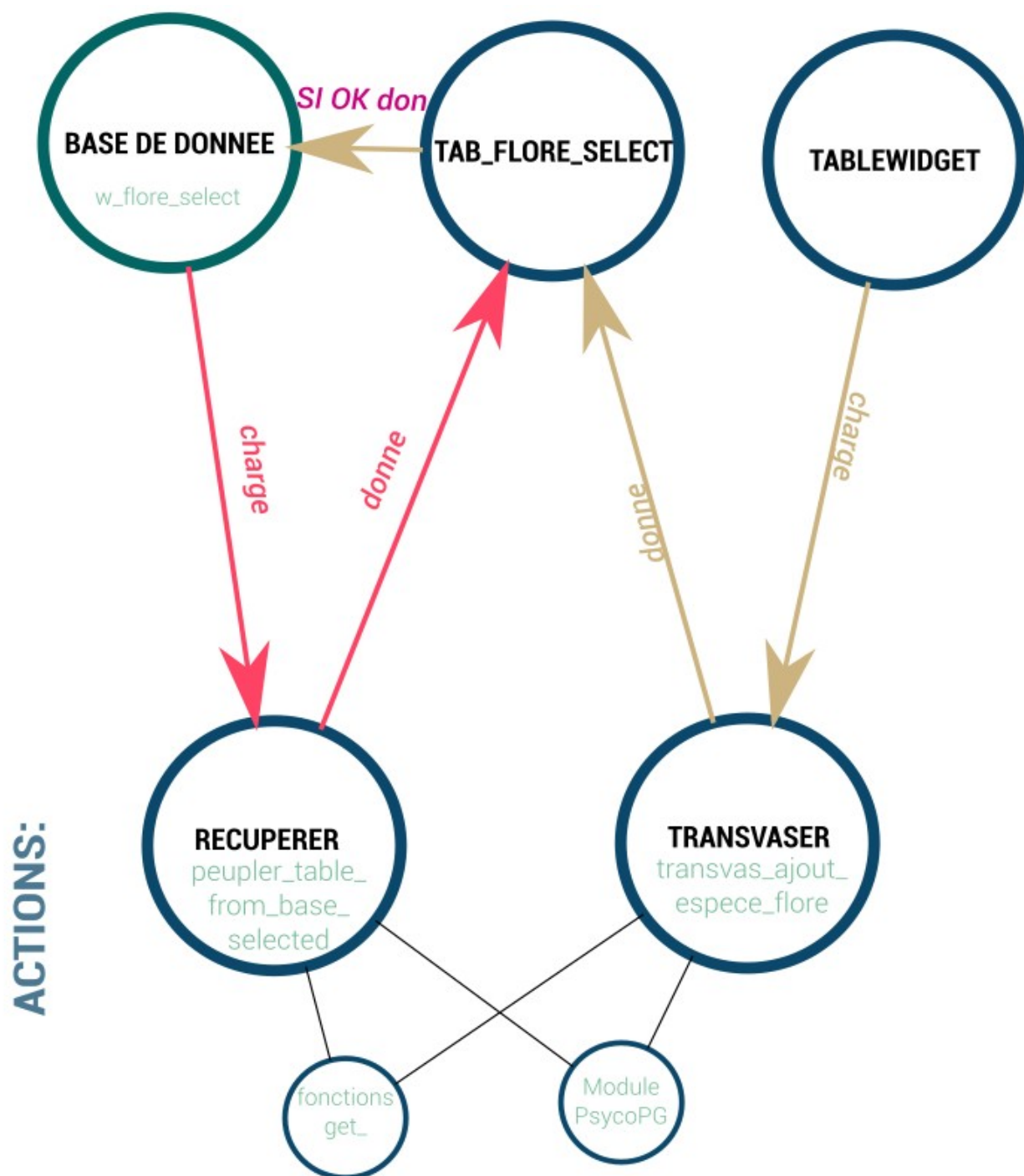


Illustration 34: Afficher et transférer la sélection d'espèces.

Lors d'une première instance (en **rose** dans le schéma ci-dessus), les éléments présents dans `w_flore_select` sont chargés et retransmis dans le widget `tab_flore_select`, en passant par la fonction *DialogSelectFlore.peupler\_from\_base\_selected*. On retrouve une fonction similaire à celle de l'autre tableau mais le premier paramètre, c'est à dire le tableau cible, est changé. Le concept est donc similaire.

Mais dans le cas d'une copie (en **marron** dans le schéma ci-dessus), il faut prendre en compte les cas de redondance. Une liste Python qui contient la première colonne (`espece_id`) de tout les éléments déjà présents dans `tab_flore_select` est mise en place.

```
compterow = self.tab_flore_selected.rowCount()
old_maliste0=[]
for i in xrange(0,compterow):
    old_maliste0.append(self.tab_flore_selected.item(i,0).text())
```

Illustration 35: Création d'une liste "anti-doublon"

Puis un contrôle a lieu pour s'assurer que des lignes ont bien été sélectionnées dans le tableau supérieur.

```
#MON TRANSVASAGE
#Contrôle
if self.matablewidget.selectionModel().hasSelection() != True:
    QMessageBox.information(self,'Information','Veuillez selectionner des entites a transferer.')
else:
```

Illustration 36: structure de contrôle: une ligne doit être sélectionnée

Une fois le contrôle passé, des variables récupèrent le contenu des lignes, puis elles sont itérées et les éléments présents dans `old_maliste0` sont exclus. Les éléments peuvent ensuite être ajoutés dans le tableau inférieur avec la méthode `set item`.

```
else:

    #Variables qui contiennent les elements des cellules dans les rows sélectionnés
    mesrows0 = self.matablewidget.selectionModel().selectedRows(0)
    mesrows1 = self.matablewidget.selectionModel().selectedRows(1)
    mesrows2 = self.matablewidget.selectionModel().selectedRows(2)

    #Threeliste
    maliste0 = []
    for i in mesrows0:
        if i.data() not in old_maliste0:
            maliste0.append(i.data())

    maliste1 = []
    for i in mesrows1:
        maliste1.append(i.data())

    maliste2 = []
    for i in mesrows2:
        maliste2.append(i.data())

    #On répartit les listes dans des QTableWidgetItem!
    for i in xrange(0, len(maliste0)):
        self.tab_flore_selected.insertRow(self.tab_flore_selected.rowCount())
        self.tab_flore_selected.setItem(self.tab_flore_selected.rowCount()-1,0 ,QTableWidgetItem(self.tr(maliste0[i])))
        self.tab_flore_selected.setItem(self.tab_flore_selected.rowCount()-1,1 ,QTableWidgetItem(self.tr(maliste1[i])))
        self.tab_flore_selected.setItem(self.tab_flore_selected.rowCount()-1,2 ,QTableWidgetItem(self.tr(maliste2[i])))
```

Illustration 37: Insertion d'éléments dans `tab_flore_select`

## Supprimer des espèces de la sélection.

Si l'utilisateur désire vider la liste d'espèces sélectionnées, la méthode `ModuleExport.vider_la_table` est appelée, avec en paramètre la table flore ou faune. Le fonctionnement est simple avec la transmission d'une requête de type `DELETE` sur la table `w_flore_select`. Pour la partie QT, le contenu est effacé et le nombre de lignes remis à zéro manuellement :

```
if typesuppr == 'fl':  
    self.dlg_Flo.tab_flore_selected.clearContents()  
    self.dlg_Flo.tab_flore_selected.setRowCount(0)
```

*Illustration 38: Vider une table de sélection.*

La suppression est immédiatement prise en compte, sans avoir à cliquer sur OK.

Pour la suppression d'un échantillon, il faudra en revanche bien cliquer sur OK pour que l'action soit prise en compte. L'action ne porte donc que sur la partie Qt. Le principe est de supprimer chaque élément selon son index. Une liste d'index sélectionnés est donc générée puis supprimé du tableau. Mais une inversion de la liste est nécessaire, pour commencer la suppression par la fin et ainsi préserver les index inférieurs à supprimer :

```
#Méthode de retrait d'espèce dans la colonne de selection  
def transvasretrait_espece_flore(self):  
  
    rows = self.tab_flore_selected.selectionModel().selectedRows()  
    #liste inversée pour régler les pb d'index.  
    rows = sorted(rows, reverse = True)  
    for r in rows:  
        self.tab_flore_selected.removeRow(r.row())
```

*Illustration 39: Méthode de suppression d'éléments dans la sélection d'espèces.*

## Ajouter les espèces dans la base de donnée.

Les actions d'ajouts d'espèces décrites précédemment ont montré comment ajouter des entités dans l'interface QT, mais sans vraiment aborder le transfert vers la table w\_flore\_select. Le bouton OK lance l'instruction d'exécution, qui est elle même reliée à la fonction ModuleExport.ajout\_ls\_espece\_pg().

```
if self.dlg_Flo.exec_():  
    self.ajout_ls_espece_pg('fl')
```

*Illustration 40: Appel de la fonction d'ajout d'espèces lors de l'appui sur OK*

La fonction récupère les ID des éléments dans le tableau de sélection d'espèces, vide la table w\_flore\_select, puis insère les entités de la table w\_ref\_flore qui correspondent aux ID sélectionnés.

```
elif typeadd == 'fl':  
    matable_select = self.tr('export.w_flore_select')  
    ma_wref = self.tr('bdfauneflore.w_ref_flore')  
    #Création d'une liste convertie en string + guillemets + parentheses pour la passer en param  
  
    compterow = self.dlg_Flo.tab_flore_selected.rowCount()  
  
    malisteid=[]  
    for i in xrange(0,compterow):  
        malisteid.append(self.dlg_Flo.tab_flore_selected.item(i,0).text())  
  
    if malisteid: #Syntaxe pour vérifier que la liste ne soit pas vide.  
        malisteid = ", ".join(malisteid)  
        malisteid = self.tr("(" + malisteid + ")")  
        idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(),  
                                                                    self.get_host_ip(), self.get_dbuser_name(),  
                                                                    self.get_pswrd())  
  
        conn = psycopg2.connect(idconnexion)  
        cur = conn.cursor()  
  
        #On vide la table pour éviter d'insérer des doublons (optimisation)  
        #TODO: Ajouter un Vacuum pour éviter une surcharge del a table à long terme.  
        marequetel=""  
        delete from %s  
        """ %matable_select  
        cur.execute(marequetel1)  
        conn.commit()  
        cur.close()  
        cur=conn.cursor()  
  
        #Envoi de nos parametres dans la liste espece_select  
        especeid = '(espece_id'  
        marequete2 = ""  
        insert into %s %s, taxon, nomcommun%s  
        select espece_id, taxon, nomcommun  
        from %s bdf  
        where bdf.espece_id in %s  
        """ % (matable_select, self.tr('(espece_id)'), self.tr(')'), ma_wref, malisteid)  
        print marequete2  
        cur.execute(marequete2)  
        conn.commit()  
        conn.close
```

*Illustration 41: Envoyer dans la base de donnée les espèces retenues.*



## Assurer une fermeture propre de la fenêtre.

Lorsque la fenêtre de sélection d'espèce est fermé, l'action `close()` est appelée. Cette action remet dans un état vierge les tableaux Qt, en vidant le contenu. La barre de recherche est également remise à zéro. Le fait de vider le tableau Qt inférieur permet à l'utilisateur de ne pas prendre en compte ses derniers ajouts dans le cas où il changerait d'avis.

```
if self.dlg_Flo.close():
    self.dlg_Flo.tab_flore_selected.clearContents()
    self.dlg_Flo.matablewidget.clearContents()
    self.dlg_Flo.tab_flore_selected.setRowCount(0)
    self.dlg_Flo.matablewidget.setRowCount(0)
    self.dlg_Flo.combosearch.clear()
```

*Illustration 42: Fermer proprement la fenêtre de sélection d'espèces*

## Nettoyer les tables et fichiers.

Le plugin offre 2 fonctionnalités pour faire de l'ordre dans les tables et fichiers d'export, dans un souci de confort pour l'usager. La première fonctionnalité, le *KillButton*, appelle tout simplement la fonction `kill_table`, explicité dans la documentation sur les fonctions SQL. Le bouton `ButtonCleanExport` procède quant à lui entièrement en Python avec l'appel de la fonction `ModuleExport.nettoyage_dossier_export()`.

Le dossier d'export est défini et normé avec `os.path.normpath`, puis Python itère sur chaque élément du dossier avec `os.listdir`. Chaque élément est supprimé, et son nom est ajouté à une liste. Cette liste est retournée à l'usager dans Qgis sous la forme d'une boîte de dialogue informative.

```
def nettoyage_dossier_export(self):
    mondossier = os.path.normpath('S:/00_MODULE_EXPORT_BDCEN/exports')
    cpt = 0
    listdel = []
    for filename in os.listdir(mondossier) :
        cpt+=1
        listdel.append(os.path.split(mondossier + '/' + filename)[1])
        os.remove(mondossier + '/' + filename)

    elemsuppr = u"%s éléments ont été supprimés:" %cpt
    QMessageBox.information(self.dlg,'Information',u'Nettoyage du dossier d\'export effectué.' + '\n' + elemsuppr \
        + "\n" + ("\n".join(listdel)))
```

*Illustration 43: Fonction pour vider un dossier.*

Le bouton à cocher `checkClean` lance automatiquement en fin d'export la fonction `kill_table` uniquement.

## Exporter.

L'export se lance en cliquant sur le bouton d'export, *ButtonExport*, qui enclenche une succession d'actions avec la fonction d'export, *ModuleExport.lancement\_des\_operations()* :

- Contrôle sur la saisie des dates.
- Affichage d'une barre de progression.
- Exécution des fonctions PostgreSQL et Python d'export, avec récupération des paramètres.
- Suppression éventuelle des tables temporaires.

## Contrôle de cohérence du paramètre de dates.

Le contrôle sur la saisie des dates se contente de vérifier que la date minimum n'est pas supérieure à la maximale. Si tel est le cas, la fonction est avortée et un message d'avertissement est envoyé.

```
#Structure de controle date de saisie
if self.dlg.radioSaisie.isChecked() == True:
    if self.dlg.date_min_pec.date() > self.dlg.date_max_pec.date():
        QMessageBox.warning(self.dlg, 'Information', u'La date minimum de saisie doit être inférieure à la maximale')
    return
```

Illustration 44: Contrôle de la date

## Informé l'utilisateur de l'évolution de l'export.

Lors du passage des fonctions, une barre de progression apparaît et renseigne l'utilisateur sur la nature des opérations en cours de réalisation. Ceci est réalisé avec des changements de valeurs entre chaque fonction :

```
self.dlg.progressBar.show()
self.dlg.avancee.show()

self.dlg.avancee.setText(u'1.Procédure de filtrage géographique en cours.')
self._1_requeteGenerale()
self.dlg.progressBar.setValue(25)

self.dlg.avancee.setText(u'2.Procédure de filtrage attributaire en cours.')
self._2_moissonneuse_bateuse()
self.dlg.progressBar.setValue(40)

self.dlg.avancee.setText(u'3.Procédure de génération des tables et CSV en cours.')
self._3_Fichier_CSV(typeExport = self.dlg.comboExport.currentText())
self.dlg.progressBar.setValue(50)

self.dlg.avancee.setText(u'4.Procédure d'export des éléments géographiques en cours')
self._4_Export_SIG(typeExport = self.dlg.comboExport.currentText())
self.dlg.progressBar.setValue(85)

self.dlg.avancee.setText(u'5.Procédure de génération de métadonnées XML en cours')
self._5_Metadonne()
self.dlg.progressBar.setValue(100)
self.dlg.avancee.setText(u'Export terminé')
```

Illustration 45: Informer l'utilisateur de la progression de son export.

## première fonction : Enclencher le filtrage géographique.

Il se fait avec la fonction `_1_requeteGenerale()`, qui consiste en un appel de la fonction de récupération des ID dans PostgreSQL (`export._1_recupdesid()`), avec le module `PsycoPG`. Le passage d'une liste dans une fonction PostgreSQL est peu intuitif, et prend cette forme :

```
SELECT export._1_recupdesid('{%s}'::int[]) ;  
""" % self.recup_id()
```

*Illustration 46: Appel de la fonction de filtrage géographique*

Il y a des crochets, et une conversion à la volée en type liste d'integer ( `::int[]`). La récupération des ID `,recup_id()`, est assurée par l'appel à `pyQgis` et son iface. Les éléments sont récupérés dans une variable qui est itérée pour créer une liste d'ID :

```
def recup_id(self):  
    Mesobjet = self.iface.activeLayer().selectedFeatures ()  
    Mesid = ', '.join([unicode(f['id']) for f in Mesobjet ])  
    return Mesid
```

*Illustration 47: Récupération des ID de z\_export sélectionnés.*

## Deuxième fonction : enclencher le filtrage attributaire.

Cette fonction récupère l'essentiel des données renseignées par l'utilisateur dans le plugin, puis appelle la fonction PostgreSQL `_2_filtrage()` en lui passant 9 arguments récupérés.

Pour rappel, voici la fonction PostgreSQL de filtrage attributaire et les arguments qu'elle nécessite.

`export._2_filtrage`

**param\_espece** boolean,  
**param\_faune** boolean,  
**param\_flore** boolean,  
**param\_typedate** boolean,  
**param\_date\_debut** character varying,  
**param\_date\_fin** character varying,  
**param\_valide** boolean,  
**param\_present** boolean,  
**param\_externe** boolean)

L'utilisation de booléens est maximisée pour que la machine ait une lecture la plus rapide possible des paramètres et pour garder un modèle uniforme. La signification des booléens est explicitée dans la section de documentation du code SQL.

Pour les paramètres de type booléen, la fonction `isChecked()` permet de vérifier si les checkbox sont cochées ou non. Une variable python est ensuite créée avec la variable de type string qui contient `True` ou `False` le cas échéant.

```

#Récupération des ESPECES en checkant le RadioButton.
if self.dlg.rad_all_espece.isChecked() == True:
    self.boolespece = 'False'
elif self.dlg.rad_slc_espece.isChecked() == True:
    self.boolespece = 'True'

```

*Illustration 48: Exemple de récupération de paramètre booléen*

Pour les dates, un seul booléen n'est pas suffisant. Les dates sont transmises à PostgreSQL sous la forme de chaînes, à l'aide de la méthode `toString()`. Si le bouton radio d'observation est activé alors le formatage se fait sous la forme date-mois-année, et dans le cas d'observations seule l'année est prise en compte. Une variable booléenne demeure présente pour indiquer si l'export portera sur la date de saisie ou d'observation.

```

if self.dlg.radioSaisie.isChecked() == True:
    self.datemin = self.dlg.date_min_pec.date().toString("dd-MM-yyyy")
    self.datemax = self.dlg.date_max_pec.date().toString("dd-MM-yyyy")
    self.booldate = 'True'
elif self.dlg.radioObs.isChecked() == True:
    self.datemin = self.dlg.date_min_pec.date().toString("yyyy")
    self.datemax = self.dlg.date_max_pec.date().toString("yyyy")
    self.booldate = 'False'

```

*Illustration 49: Récupérer les paramètres de date: boolean & string*

Les paramètres sont ensuite passés dans une requête SQL enrobée :

```

marequete = """
SELECT export._2_filtrage(%s, %s, %s, %s, '%s', '%s', %s, %s, %s) ;
""" % (self.boolespece, self.boolfaune, self.boolflore, self.booldate,
      self.datemin, self.datemax, self.boolvalid, self.boolpresent,
      self.boolexterne)

```

*Illustration 50: Passage de la requête de filtrage attributaire*

### Troisième fonction : Sélection du type d'export et Génération des tables temporaires et CSV.

Cette fonction passe en premier lieu le type d'export choisi par l'utilisateur. Pour rappel, les exports sont stockés dans une table attributaire, ce qui permet de faire évoluer le module sans toucher au code python. La table w\_type\_export est organisée ainsi.

nom character varying(255)	fonction character varying(255)	ogr2ogr character varying	id integer
export brut	export._exportbrut()	exportbrut.bat	1
Export INPN	export._exportINPN()	exportinpn.bat	2

- Nom : la colonne nom est celle affichée dans le plugin
- fonction : le nom de la fonction PostgreSQL à appeler.
- Ogr2ogr : le nom du fichier .bat à lancer pour la quatrième étape de l'export.

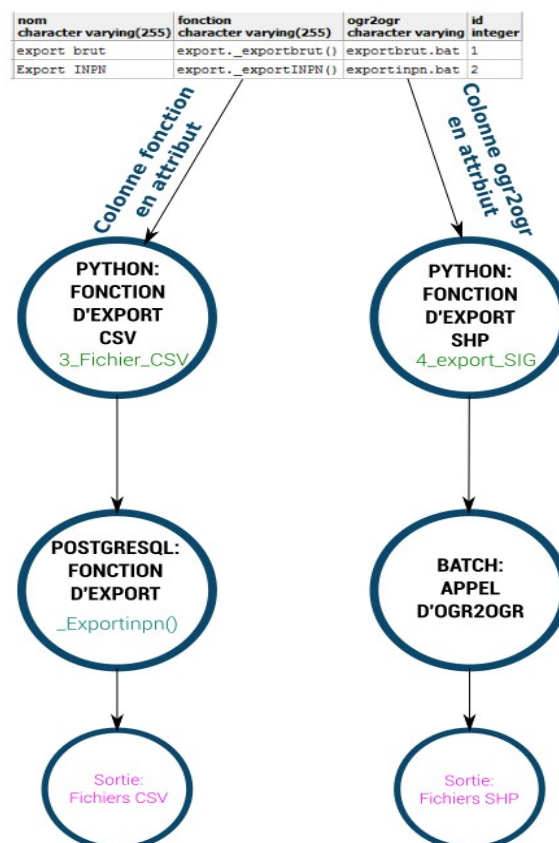


Illustration 51: Récupérer des instructions à exécuter.

Le nom de l'export choisi est stockée dans la variable typeExport, grâce à la méthode currentText sur la boîte combo. Puis, le texte associé dans la colonne fonction est passé pour appeler la fonction PostgreSQL qui a ce nom.

```
#fonction géénrique de recuperation dans notre table de type d export la fonction associée au type.
marequete = ""select fonction from export.w_type_export where nom = '%s' "" % typeExport
cur.execute(marequete)
myfunc = cur.fetchone()[0]
#Appel de la fonction en remplaçant le nom de variable par la variable en parametre.
marequete2 = "SELECT " + str(myfunc)
cur.execute(marequete2)
```

*Illustration 52: Appeler la fonction associée au nom de fonction.*

Pour ce système soit viable en terme d'évolutivité, les fonctions passées ne doivent pas contenir d'arguments.

## Quatrième fonction : Exécuter les fichiers .bat pour générer des shapefiles.

Le schéma de la troisième fonction montre un fonctionnement similaire à la quatrième fonction pour ce qui est de la récupération d'élément à exécuter, mais la colonne à utiliser est cette fois ogr2ogr. La colonne contient le nom du fichier bat et son extension, qu'il suffit d'exécuter avec la fonction call en concaténant le nom du fichier bat avec son répertoire.

```
#Chercher le .bat à lancer.
monpath = "S:/00_MODULE_EXPORT_BDCEN/batfiles/"
print os.path.normpath(monpath + myOgr)
call(os.path.normpath(monpath + myOgr),shell=True)
```

*Illustration 53: Appel des fichiers.bat*

### Structure d'un fichier .bat :

Si ogr2ogr est bien implanté dans le path, il n'est pas nécessaire de définir un répertoire de travail. Sinon, celui-ci doit être spécifié avec une commande Change Directory (cd) qui pointe dans le répertoire de GDAL, différent selon l'installateur choisi.

```
cd /d C:\0SGeo4W64\share\gdal\
```

Puis Ogr2ogr est appelé. La destination est indiquée avant la source, avec des paramètres de format et SCR. La source, de type base de donnée, doit contenir les coordonnées de la base de donnée, avec login et mot de passe. Le **répertoire des fichiers .bat doit donc être sécurisé en lecture.**

Enfin, la dernière est une requête SQL, qui dans le cas de ce module doit se contenter de reprendre les tables préparés dans PostgreSQL : Le but est une centralisation des traitements dans les fonctions.

```
ogr2ogr -overwrite -t_srs EPSG:2154 -geomfield geom -f "ESRI Shapefile" S:\00_MODULE_EXPORT_BDCEN\exports\st_sig_poly.shp ^
PG:"host=192.168.1.2 dbname=bdcenpicardie_dev user=postgres password=*****" ^
-sql "select * from export.e_st_sig_poly"
```

Illustration 54: Création d'un shapefile avec Ogr2Ogr

## Cinquième fonction : génération de la fiche de métadonnée.

La génération de la fiche de métadonnée en XML se base sur le module minidom de Python. Le fonctionnement consiste à générer des nœuds, qui peuvent contenir des attributs, puis d'imbriquer les nœuds les uns dans les autres. Les paramètres qui ont été passés dans les fonctions, gardés en mémoire dans python peuvent donc être réutilisés et placés dans les sections adéquates dans le fichier XML. Des résultats de requêtes SQL peuvent également être insérées, ce qui permet dans ce cas de faire des statistiques, sur le nombre d'espèces dans l'export par exemple :

```
#Somme taxons
marequete = """SELECT SUM (count)
FROM
(SELECT COUNT (DISTINCT t.taxon)
FROM
export.r_id_all i, bdfauneflore.t_bilan_faune t
WHERE i.id_perm = t.id_perm
AND i.presence IS TRUE
UNION
SELECT COUNT (DISTINCT t.taxon)
FROM
export.r_id_all i, bdfauneflore.t_bilan_flore t
WHERE i.id_perm = t.id_perm
AND i.presence IS TRUE) AS x"""
cur.execute(marequete)
```

Illustration 55: Connaître la somme des taxons de l'export.

La date d'export est récupérée avec la fonction time de Python, ce qui permet de retracer les exports.

```
dateExport = time.strftime("%d/%m/%Y")
```

D'autres champs sont implantés en dur, comme l'extension de fichier en Shapefile pour l'échange de données SIG, en Lambert 93. Cela devrait être le cas pour 99 % des exports. En cas de changement, une édition manuelle de la fiche de métadonnée XML est toujours possible.

Une fois les nœuds créés en rempli, la fonction toprettyxml permet de sortir un modèle xml bien organisé. La dernière étape est l'écriture de ce modèle avec les classiques fonctions write et open de Python.

```
sortieXML = modelExport.toprettyxml()
ecrirexml = open(os.path.normpath('S:/00_MODULE_EXPORT_BDCEN/exports/exportmetadata.xml'), 'w')
ecrirexml.write(sortieXML)
ecrirexml.close()
```

Illustration 56: Ecriture du fichier XML

# Code Python du module d'export.

## Classe principale

```
# -*- coding: utf-8 -*-
"""
/*****
ModuleExport

                                A QGIS plugin

export

                                -----
begin                          : 2016-04-14
git sha                        : $Format:%H$
copyright                      : (C) 2016 by Loic Martel
email                          : loic.martel@outlook.com
*****/

/*****
*
*   This program is free software; you can redistribute it and/or modify
*   it under the terms of the GNU General Public License as published by
*   the Free Software Foundation; either version 2 of the License, or
*   (at your option) any later version.
*
* *****/
"""
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
from PyQt4.QtGui import QAction, QIcon, QMessageBox, QTableWidgetItem
# Initialize Qt resources from file resources.py
import resources
# Import the code for the dialog
from ModuleExport_dialog import ModuleExportDialog
from DialogSelectFaune import DialogSelectFaune
from DialogSelectFlore import DialogSelectFlore
import os.path
import time
import psycpg2
from PyQt4.Qt import QPushButton, QSqlDatabase, QSqlTableModel
from xml.dom import minidom #Pour pouvoir générer notre fichier de metadata.
from subprocess import call

class ModuleExport:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):

        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
```



```

        self.plugin_dir,
        'i18n',
        'ModuleExport_{}.qm'.format(locale))

if os.path.exists(locale_path):
    self.translator = QTranslator()
    self.translator.load(locale_path)

if qVersion() > '4.3.3':
    QApplication.installTranslator(self.translator)

# Create the dialogs (after translation) and keep reference
self.dlg = ModuleExportDialog(self.iface)
self.dlg.setWindowTitle("module d'export")
self.dlg_F = DialogSelectFaune(self.dlg)
self.dlg_Flo = DialogSelectFlore(self.dlg)

#Fonctions esthétiques: ajout d'icônes.
self.dlg.ButtonFaune.setIcon(QIcon(self.plugin_dir+"/turtle-128.png"))
self.dlg.ButtonFaune.setIcon(QIcon(self.plugin_dir+"/turtle-128.png"))

self.dlg.ButtonFlore.setIcon(QIcon(self.plugin_dir+"/flower.png"))

self.dlg_F.vider_faune.clicked.connect(Lambda:self.vider_la_table('fa'))
self.dlg_Flo.vider_flore.clicked.connect(Lambda:self.vider_la_table('fl'))
self.dlg.ButtonFaune.clicked.connect(self.fenetre_select_faune)
self.dlg.ButtonFlore.clicked.connect(self.fenetre_select_flore)

#Relier le bouton d'ajout d'especefaune à mon SLOT dans DialogSelectFaune
transvas_espece.
    self.dlg_F.btn_ajoutfaune.clicked.connect(Lambda:
self.dlg_F.transvasajout__espece_faune())
    self.dlg_F.btn_rmfaune.clicked.connect(Lambda:
self.dlg_F.transvasretrait_espece_faune())

    self.dlg_Flo.btn_ajoutflore.clicked.connect(Lambda:
self.dlg_Flo.transvasajout__espece_flore())
    self.dlg_Flo.btn_rmflore.clicked.connect(Lambda:
self.dlg_Flo.transvasretrait_espece_flore())

    self.dlg_Flo.btn_visible.clicked.connect(Lambda :
self.dlg_Flo.visibilite(True))
    self.dlg_Flo.btn_invisible.clicked.connect(Lambda :
self.dlg_Flo.visibilite(False))

self.dlg_F.btn_visible.clicked.connect(Lambda : self.dlg_F.visibilite(True))
self.dlg_F.btn_invisible.clicked.connect(Lambda : self.dlg_F.visibilite(False))

# Declare instance attributes
self.actions = []
self.menu = self.tr(u'ModuleExport')

```

```

# TODO: We are going to let the user set this up in a future iteration
self.toolbar = self.iface.addToolBar(u'ModuleExport')
self.toolbar.setObjectName(u'ModuleExport')

#MEP du signal pour vider les selections.
self.dlg_F.unselect.clicked.connect(lambda: self.dlg_F.clear())
self.dlg_Flo.unselect.clicked.connect(lambda: self.dlg_Flo.clear())

#MEP du signal pour la fonction de recherche.
self.dlg_F.searchButton.clicked.connect(lambda: self.va_chercher('fa'))
self.dlg_Flo.searchButton.clicked.connect(lambda: self.va_chercher('fl'))

#Connexion des bouton de lancement et nettoyage de table
self.dlg.buttonLaunch.clicked.connect(lambda: self.lancement_des_operations())
self.dlg.killButton.clicked.connect(lambda: self.nettoyage_des_tables())
self.dlg.buttonCleanExport.clicked.connect(lambda:
self.nettoyage_dossier_export())

# noinspection PyMethodMayBeStatic

def tr(self, message):
    # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
    return QApplication.translate('ModuleExport', message)

def add_action(
    self,
    icon_path,
    text,
    callback,
    enabled_flag=True,
    add_to_menu=True,
    add_to_toolbar=True,
    status_tip=None,
    whats_this=None,
    parent=None):

    icon = QIcon(self.plugin_dir+"/icon.png")
    action = QAction(icon, "Module d'export", parent)
    action.triggered.connect(callback)
    action.setEnabled(enabled_flag)

    if status_tip is not None:
        action.setStatusTip(status_tip)

    if whats_this is not None:
        action.setWhatsThis(whats_this)

    if add_to_toolbar:
        self.toolbar.addAction(action)

    if add_to_menu:
        self.iface.addPluginToMenu(
            self.menu,
            action)

    self.actions.append(action)

```

```

        #Ajout de notre action a la liste d actions.
        return action

def initGui(self):
    """Create the menu entries and toolbar icons inside the QGIS GUI."""

    icon_path = self.plugin_dir+"/icon.png"
    self.add_action(
        icon_path,
        text=self.tr(u'Module d export'),
        callback=self.run,
        whats_this = self.tr(u'Module permettant d\'exporter des données Faune /
Flore vers une Shapefile ou un CSV.'),
        parent=self.iface.mainWindow())

def unload(self):
    """Removes the plugin menu item and icon from QGIS GUI."""
    for action in self.actions:
        self.iface.removePluginMenu(
            self.tr(u'&ModuleExport'),
            action)
        self.iface.removeToolBarIcon(action)
    # remove the toolbar
    del self.toolbar

def peuplerComboExport(self):
    #creation connexion PGQT pour peupler la liste des type_export.
    self.QConn = QSqlDatabase("QPSQL")
    self.QConn.setHostName(self.get_host_ip())
    self.QConn.setDatabaseName(self.get_db_name())
    self.QConn.setUserName(self.get_dbuser_name())
    self.QConn.setPassword(self.get_pswrd())
    self.QConn.open()

    #Creation du modele de donnée pour peupler laQcombobox.
    self.Model_export = QSqlTableModel(db = self.QConn)
    self.Model_export.setTable("export.w_type_export")
    self.Model_export.select()
    #Peupler la Qcombobox
    self.dlg.comboExport.setModel(self.Model_export)
    self.dlg.comboExport.setModelColumn(self.Model_export.fieldIndex("nom"))

#####
#####
#####
#####
##### FONCTIONS DE RECUP DE DONNEES #####
##### DANS PGSQL #####
#####
#####
#####

def get_table_name(self):
    #Fonction pour obtenir le nom réel de la couche sélectionnée. "Schema.table"
    table_name = str(self.iface.activeLayer().source())
    table_name = table_name.replace(table_name[0:table_name.find('table=')+6],
    "").replace("'", '').split()

    return table_name[0]

```



```

        idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(),
self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd())

        conn = psycopg2.connect(idconnexion)
        cur = conn.cursor()
        #fourniture d un echantillon basique et léger.
        marequete = """
        select w.espece_id, w.taxon, w.nomcommun, w.classe, w.ordre, w.cd_nom, w.cd_ref
,w.rarepic ,w.menapic
        from bdfauneflore.w_ref_faune w
        limit 100
        """

        cur.execute(marequete)

        for i in cur:
            self.dlg_F.peupler_table_from_base(self.dlg_F.matablewidget, i)

        cur.close()
        self.dlg_F.matablewidget.resizeColumnToContents(1)
        cur = conn.cursor()
        marequete2 = """
        select w.espece_id, w.taxon, w.nomcommun
        from export.w_faune_select w
        """
        cur.execute(marequete2)
        for i in cur:

self.dlg_F.peupler_table_from_base_selected(self.dlg_F.tab_faune_selected,i)
        cur.close()
        conn.close()

        #Si on appuie sur ok ou si l'utilisateur ferme la fenetre alors les listes sont
vidées graphiquement.

        #si l'utilisateur clique sur OK alors la liste d esepce selectionnee est
transferee dans la table sleect_faune---flore.
        if self.dlg_F.exec():
            self.ajout_ls_espece_pg('fa')

        if self.dlg_F.close():
            self.dlg_F.tab_faune_selected.clearContents()
            self.dlg_F.matablewidget.clearContents()
            self.dlg_F.tab_faune_selected.setRowCount(0)
            self.dlg_F.matablewidget.setRowCount(0)
            self.dlg_F.combosearch.clear()

def fenetre_select_flore(self):

        self.dlg_Flo.combosearch.addItem([self.tr('taxon'), self.tr('nomcommun'),
self.tr('classe'), self.tr('ordre'), self.tr('espece_id'),self.tr('cd_ref'),

```

```

self.tr('cd_nom'), self.tr('rarepic'), self.tr('menapic']])

        idconnexion = ("dbname=%s host=%s user=%s password=%s" % (self.get_db_name(),
self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd()))

        conn = psycopg2.connect(idconnexion)
        cur = conn.cursor()
        #fourniture d un echantillon basique et léger.
        marequete = """
        select w.espece_id, w.taxon, w.nomcommun, w.classe, w.ordre, w.cd_nom, w.cd_ref
,w.rarepic ,w.menapic
        from bdfauneflore.w_ref_flore w
        limit 100
        """

        cur.execute(marequete)

        for i in cur:
            self.dlg_Flo.peupler_table_from_base(self.dlg_Flo.matablewidget, i)

        cur.close()
        self.dlg_Flo.matablewidget.resizeColumnToContents(1)
        cur = conn.cursor()
        marequete2 = """
        select w.espece_id, w.taxon, w.nomcommun
        from export.w_flore_select w
        """
        cur.execute(marequete2)
        for i in cur:

self.dlg_Flo.peupler_table_from_base_selected(self.dlg_Flo.tab_flore_selected,i)
        cur.close()
        conn.close()

        #Si on appuie sur ok ou si l'utilisateur ferme la fenetre alors les listes sont
vidées graphiquement.

        #si l'utilisateur clique sur OK alors la liste d esepce selectionnee est
transferee dans la table sleect_faune--/--flore.
        if self.dlg_Flo.exec_():
            self.ajout_ls_espece_pg('fl')

        if self.dlg_Flo.close():
            self.dlg_Flo.tab_flore_selected.clearContents()
            self.dlg_Flo.matablewidget.clearContents()
            self.dlg_Flo.tab_flore_selected.setRowCount(0)
            self.dlg_Flo.matablewidget.setRowCount(0)
            self.dlg_Flo.combosearch.clear()

```

```
#####
##### FONCTIONS POUR FAUNE FLORE #####
#####
```

*#Fonction pour vider les tables de flore select et faune select.*

*#Lacement de la suppression en mettant un parametre de type (flore ou faune : fl/fa)*

```
def vider_la_table(self, typesuppr):
```

*#CONTROLE: transformation du parametre en nom de table.*

```
if typesuppr == 'fa':
```

```
    matable = 'export.w_faune_select'
```

```
elif typesuppr == 'fl':
```

```
    matable = 'export.w_flore_select'
```

```
else:
```

```
    print 'erreur sur la table à vider.'
```

```
        idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(),
self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd())
```

```
        conn = psycopg2.connect(idconnexion)
```

```
        cur = conn.cursor()
```

```
        marequete = """
```

```
DELETE FROM %s
```

```
""" % self.tr(matable)
```

```
print marequete
```

```
cur.execute(marequete)
```

*#passage du parametre pour taper dans daune ou flore.*

```
conn.commit()
```

```
conn.close()
```

*#On vide la boite de dialogue.*

```
if typesuppr == 'fa':
```

```
    self.dlg_F.tab_faune_selected.clearContents()
```

```
    self.dlg_F.tab_faune_selected.setRowCount(0)
```

```
if typesuppr == 'fl':
```

```
    self.dlg_Flo.tab_flore_selected.clearContents()
```

```
    self.dlg_Flo.tab_flore_selected.setRowCount(0)
```

```
def ajout_ls_espece_pg(self, typeadd):
```

*#CONTROLE: transformation du parametre en noms de tables.*

```
if typeadd == 'fa':
```

```
    matable_select = self.tr('export.w_faune_select')
```

```
    ma_wref = self.tr('bdfauneflore.w_ref_faune')
```

*#Création d'une liste convertie en string + guillemets +*

*parentheses pour la passer en parametre dans la requet SQL.*

```
compterow = self.dlg_F.tab_faune_selected.rowCount()
```

```
malisteid=[]
```

```
for i in xrange(0,compterow):
```

```
malisteid.append(self.dlg_F.tab_faune_selected.item(i,0).text())
```

```
elif typeadd == 'fl':
```

```
    matable_select = self.tr('export.w_flore_select')
```

```
    ma_wref = self.tr('bdfauneflore.w_ref_flore')
```

```
        #Création d'une liste convertie en string + guillemets +  
    parentheses pour la passer en parametre dans la requete SQL.
```

```
compterow = self.dlg_Flo.tab_flore_selected.rowCount()
```

```
malisteid=[]
```

```
for i in xrange(0,compterow):
```

```
    malisteid.append(self.dlg_Flo.tab_flore_selected.item(i,0).text())
```

```
if malisteid: #Syntaxe pour vérifier que la liste ne soit pas vide.
```

```
    malisteid = "','".join(malisteid)
```

```
    malisteid = self.tr("(" + malisteid + ")")
```

```
    idconnexion = ("dbname=%s host=%s user=%s password=%s") %
```

```
(self.get_db_name(),
```

```
self.get_host_ip(), self.get_dbuser_name(),
```

```
self.get_pswrd())
```

```
    conn = psycopg2.connect(idconnexion)
```

```
    cur = conn.cursor()
```

```
    #On vide la table pour éviter d'insérer des doublons (optimisation)
```

```
    #TODO: Ajouter un Vacuum pour éviter une surcharge de la table à long
```

```
terme.
```

```
    marequete1="""
```

```
delete from %s
```

```
""" %matable_select
```

```
cur.execute(marequete1)
```

```
conn.commit()
```

```
cur.close()
```

```
cur=conn.cursor()
```

```
    #Envoi de nos parametres dans la liste espece_select
```

```
    especeid = '(espece_id'
```

```
marequete2 = """
```

```
insert into %s %s, taxon, nomcommun%s
```

```
select espece_id, taxon, nomcommun
```

```
from %s bdf
```

```
where bdf.espece_id in %s
```

```
""" % (matable_select, self.tr('(espece_id)'), self.tr(')'), ma_wref,
```

```
malisteid)
```

```
print marequete2
```

```
cur.execute(marequete2)
```

```
conn.commit()
```

```
conn.close
```



*#Fonction pour lancer la recherche d'éléments dans la liste d'espèce.*

```
def va_chercher(self, typecherche):
    if typecherche == 'fa':
        matable_select = self.tr('export.w_faune_select')
        ma_wref = self.tr('bdfauneflore.w_ref_faune')
        maliste_wref = self.dlg_F.matablewidget
        marecherche = self.dlg_F.search_bar.text()
        moncritere = self.dlg_F.combosearch.currentText()
    elif typecherche == 'fl':
        matable_select = self.tr('export.w_flore_select')
        ma_wref = self.tr('bdfauneflore.w_ref_flore')
        maliste_wref = self.dlg_Flo.matablewidget
        marecherche = self.dlg_Flo.search_bar.text()
        moncritere = self.dlg_Flo.combosearch.currentText()

    if len(marecherche)<3:
        return
    else:
        maliste_wref.clearContents()
        maliste_wref.setRowCount(0)
        marecherche = '%' + marecherche + '%'
        idconnexion = ("dbname=%s host=%s user=%s password=%s") %
(self.get_db_name(), self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd())
        conn = psycopg2.connect(idconnexion)
        cur = conn.cursor()
        marequete = """
select w.espece_id, w.taxon, w.nomcommun, w.classe, w.ordre, w.cd_nom,
w.cd_ref ,w.rarepic ,w.menapic
from %s w
where %s%s%s iLIKE '%s'

""" % (ma_wref, 'CAST(', moncritere, ' AS TEXT)', marecherche)
# Le ILIKE permet une recherche insensible à la casse, mais augmente
considérablement le temps de requetage.

        cur.execute(marequete)

        if typecherche == 'fl':
            for i in cur:
                self.dlg_Flo.peupler_table_from_base(self.dlg_Flo.matablewidget,i)
                #Trier les résultats par taxon.
                self.dlg_Flo.matablewidget.sortItems(1)

        elif typecherche == 'fa':
            for i in cur:
                self.dlg_F.peupler_table_from_base(self.dlg_F.matablewidget,i)
                self.dlg_F.matablewidget.sortItems(1)

        cur.close()
        conn.close()
```



```

        self.booldate = 'True'
    elif self.dlg.radioObs.isChecked() == True:
        self.datemin = self.dlg.date_min_pec.date().toString("yyyy")
        self.datemax = self.dlg.date_max_pec.date().toString("yyyy")
        self.booldate = 'False'

```

*#Récupération des ESPECES en checkant Le RadioButton.*

```

if self.dlg.rad_all_espece.isChecked() == True:
    self.boolespece = 'False'
elif self.dlg.rad_slc_espece.isChecked() == True:
    self.boolespece = 'True'

```

*#Verification de si L on desire toute la flore ou toute la faune.*

```

if self.dlg.checkFlore.isChecked() == True:
    self.boolflore = 'True'
else:
    self.boolflore = 'False'

```

```

if self.dlg.checkFaune.isChecked() == True:
    self.boolfaune = 'True'
else:
    self.boolfaune = 'False'

```

*#Vérification de La checkbox du parametre pour définir si L'on veut Les ESPECES VALIDES.*

```

if self.dlg.obj_valide.isChecked() == True:
    self.boolvalid = 'True'
elif self.dlg.obj_valide.isChecked() == False:
    self.boolvalid = 'False'
else:
    return

```

*#Vérification de La checkbox de parametres pour définir si L'on veut Les espèces PRESENTES.*

```

if self.dlg.obj_present.isChecked() == True:
    self.boolpresent = 'True'
elif self.dlg.obj_present.isChecked() == False:
    self.boolpresent = 'False'
else:
    return

```

*#Vérification de La checkbox de parametres pour définir si L'on veut Les données externes.*

```

if self.dlg.radioExt.isChecked() == True:
    self.boolexterne = 'True'
elif self.dlg.radioExt.isChecked() == False:
    self.boolexterne = 'False'
else:
    return

```

```

#####
#APPEL DE LA PARTIE 2 / 3#
#####
conn = psycopg2.connect(idconnexion)
cur = conn.cursor()
marequete = """
SELECT export._2_filtrage(%s, %s, %s, %s, '%s', '%s', %s, %s, %s) ;
""" % (self.boolespece, self.boolfaune, self.boolflore, self.booldate,
       self.datemin, self.datemax, self.boolvalid, self.boolpresent,
       self.boolexterne)

cur.execute(marequete)
conn.commit()
#DEBUG
#Affichage des log PG
#for notice in conn.notices:
#    print notice
conn.close()

print marequete

```

```

#Appeler la fonction d export qui genere TABLES + CSV
def _3_Fichier_CSV(self, typeExport):

```

```

    #Définition du nom par défaut.
    nomCsv = "\\ST_PRINCIPAL.csv"
    idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(),
self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd())
    conn = psycopg2.connect(idconnexion)
    cur = conn.cursor()

    #fonction géénrique de recuperation dans notre table de type d export La
fonction associée au type.
    marequete = """select fonction from export.w_type_export where nom = '%s' """ %
typeExport
    cur.execute(marequete)
    myfunc = cur.fetchone()[0]
    #Appel de la fonction en remplaçant le nom de variable par la variable en
parametre.
    marequete2 = "SELECT " + str(myfunc)
    cur.execute(marequete2)
    conn.commit()
    conn.close()

```

```

#Fonction d Export SIG au format SHP.
def _4_Export_SIG(self, typeExport):
    idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(),
self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd())
    conn = psycopg2.connect(idconnexion)

```

```

        cur = conn.cursor()
        marequete = """select ogr2ogr from export.w_type_export where nom = '%s' """ %
typeExport
        cur.execute(marequete)
        myOgr = cur.fetchone()[0]

```

```

#Chercher le .bat à lancer.
monpath = "S:/00_MODULE_EXPORT_BDCEN/batfiles/"
print os.path.normpath(monpath + myOgr)
call(os.path.normpath(monpath + myOgr), shell=True)

```

```

def _5_Metadonne(self):
    idconnexion = ("dbname=%s host=%s user=%s password=%s") % (self.get_db_name(),
self.get_host_ip(), self.get_dbuser_name(), self.get_pswrd())
    conn = psycopg2.connect(idconnexion)
    cur = conn.cursor()

```

```

#Création du XML avec parametres adéquats.
#Création de mon arbre XML en mémoire VIVE.
modelExport = minidom.Document()

```

```

#Création de ma racine. ROOT
newroot = modelExport.createElement('root')
#On ajoute le root à l'arbre
modelExport.appendChild(newroot)

```

```

#Création de la grande section de paramexport
paramexport = modelExport.createElement('parametres_d_export')

```

```

#On ajoute la paramexport à la root
newroot.appendChild(paramexport)

```

```

#Création de OBJETSSIG
objetsig = modelExport.createElement('ObjetsSIG')
paramexport.appendChild(objetsig)

```

```

#Création des ENTITE dans objetSIG.
#Remplacer liste par mes id.

```

```

self.iface.activeLayer().selectedFeatures ()

```

```

for i in self.iface.activeLayer().selectedFeatures():
    entite = modelExport.createElement('id')
    entite.setAttribute('ID', str(i[0]))
    entite.setAttribute('libelle', str(i[1]))
    objetsig.appendChild(entite)

```

```

#Création de la section de PARAMETRE D OBJETS

```

```

paramobjets = modelExport.createElement('paramobjets')

#La donnée est-elle valide?
if self.boolvalid == True:
    nodeValid = "Valide uniquement"
else:
    nodeValid = "Valide et invalide"
paramobjets.setAttribute('Validite_de_la_donnee', nodeValid)

#La Présence de L espece est-elle une certitude?
if self.boolpresent == True:
    nodePresabs = "Présence attestée"
else:
    nodePresabs = "Absence"
paramobjets.setAttribute('Certitude_de_presence', nodePresabs)

#La donnee externe est elle exclue?
if self.boolexterne == True:
    nodeExt = "Inclusion"
else:
    nodeExt = "Exclusion"
paramobjets.setAttribute('Donnée_externe', nodeExt)

paramexport.appendChild(paramobjets)

#Création de La DATE.
date = modelExport.createElement('param_date')

#La date porte sur?
if self.booldate == True:
    nodeDate = "La saisie"
else:
    nodeDate = "La date d'observation"
date.setAttribute('date_portant_sur', nodeDate)

#date min de prise en compte
date.setAttribute('date_minimale_de_prise_en_compte', str(self.datemin))
#date max de prise en compte
date.setAttribute('date_maximale_de_prise_en_compte', str(self.datemax))

paramobjets.appendChild(date)

#####
# RESUME_EXPORT
resumexport = modelExport.createElement("Resume_export")
typeexp = self.dlg.comboExport.currentText()
typeexp = typeexp.encode('UTF-8','ignore')
resumexport.setAttribute('Type_export',typeexp )
newroot.appendChild(resumexport)

```

```

#Qui est destinataire?
ledestina= modelExport.createElement("Destinataire")

#nom export
nomlabel = str(self.dlg.libele.text())
nomlabel= nomlabel.encode('UTF-8', 'ignore')
ledestina.setAttribute('libele_export', nomlabel)

#interlocuteur
nomdest = str(self.dlg.destinataire.text())
nomdest=nomdest.encode('UTF-8', 'ignore')
ledestina.setAttribute('interlocuteur', nomdest)

#Structure
nomstruc = str(self.dlg.structure.text())
nomstruc=nomstruc.encode('UTF-8', 'ignore')
ledestina.setAttribute('structure', nomstruc)

resumexport.appendChild(ledestina)

#STATISTIQUES
#Somme individus
statists = modelExport.createElement("stats_especes")
marequete = "SELECT COUNT(i.id_perm) FROM export.r_id_all i WHERE i.presence IS
TRUE"

cur.execute(marequete)
totalent = str(cur.fetchone()[0])
statists.setAttribute("Nombre_total_d_entite",totalent)

#Somme taxons
marequete = ""SELECT SUM (count)
FROM
(SELECT COUNT (DISTINCT t.taxon)
FROM
export.r_id_all i, bdfauneflore.t_bilan_faune t
WHERE i.id_perm = t.id_perm
AND i.presence IS TRUE
UNION
SELECT COUNT (DISTINCT t.taxon)
FROM
export.r_id_all i, bdfauneflore.t_bilan_flore t
WHERE i.id_perm = t.id_perm
AND i.presence IS TRUE) AS x""
cur.execute(marequete)
totalent = str(cur.fetchone()[0])
statists.setAttribute("Nombre_total_de_taxons", totalent)

resumexport.appendChild(statists)

#NB FAUNE EXPORT

#compte d individus
enfa = modelExport.createElement("faune_exportees")
marequete= "SELECT COUNT (i.id_perm) FROM \
export.r_id_all i, bdfauneflore.t_bilan_faune t WHERE i.id_perm = t.id_perm AND
i.presence IS TRUE"
cur.execute(marequete)
nbenfa = str(cur.fetchone()[0])
enfa.setAttribute('nombre_d_entites', nbenfa)

```

```

# compte d especes.
marequete= "SELECT COUNT (DISTINCT t.taxon) FROM \
export.r_id_all i, bdfauneflore.t_bilan_faune t WHERE i.id_perm = t.id_perm AND
i.presence IS TRUE"
cur.execute(marequete)
nbenfa = str(cur.fetchone()[0])
enfa.setAttribute('nombre_d_especes', nbenfa)

statistics.appendChild(enfa)

#LISTE TAXONS
if nbenfa !=0:
    marequete= "SELECT DISTINCT t.taxon FROM export.r_id_all i,\
bdfauneflore.t_bilan_faune t WHERE i.id_perm = t.id_perm AND i.presence IS
TRUE"

    cur.execute(marequete)

    for i in cur:
        nbesfa = i[0]

        childEnfa = modelExport.createElement("taxon")
        if nbesfa is None:#resolution de bug: si type est null alors valeur
arbitraire

            childEnfa.setAttribute('Taxon', "NONETYPE")
            enfa.appendChild(childEnfa)
        else:
            nbesfa = nbesfa.encode('UTF-8','ignore')#necessite de reencoder
            childEnfa.setAttribute('Taxon', nbesfa)
            enfa.appendChild(childEnfa)

#NB FLORE EXPORT

#nb entite flore exportees
efle = modelExport.createElement("flore_exportees")
marequete= "SELECT COUNT (i.id_perm) FROM \
export.r_id_all i, bdfauneflore.t_bilan_flore t WHERE i.id_perm = t.id_perm AND
i.presence IS TRUE"
cur.execute(marequete)
nbefle = str(cur.fetchone()[0])
efle.setAttribute('nombre_d_entites', nbefle)

# compte d especes.
marequete= "SELECT COUNT (DISTINCT t.taxon) FROM \
export.r_id_all i, bdfauneflore.t_bilan_flore t WHERE i.id_perm = t.id_perm AND
i.presence IS TRUE"
cur.execute(marequete)
nbefle = str(cur.fetchone()[0])
efle.setAttribute('nombre_d_especes', nbefle)

statistics.appendChild(efle)

#LISTE TAXON

```



```

if nbefle != 0:
    marequete= "SELECT DISTINCT t.taxon FROM export.r_id_all i,\
    bdfauneflore.t_bilan_flore t WHERE i.id_perm = t.id_perm AND i.presence IS
TRUE"

    cur.execute(marequete)

    for i in cur:
        nbefle = i[0]
        childEsle = modelExport.createElement("taxon")
        if nbefle is None:
            childEsle.setAttribute('Taxon', "NONETYPE")
            efle.appendChild(childEsle)
        else:
            nbefle = nbefle.encode('UTF-8','ignore')
            childEsle.setAttribute('Taxon', nbefle)
            efle.appendChild(childEsle)

conn.commit()
conn.close()

#####

#DESCRIPTION DE LA DONNEE
descdata = modelExport.createElement("Description_des_donnees")
newroot.appendChild(descdata)

# dateexport

# resume des formats
lesformats = modelExport.createElement("formats")
dateExport = time.strftime("%d/%m/%Y")
lesformats.setAttribute('date_export',dateExport)
descdata.appendChild(lesformats)

# Recap du CSV
csv_xml = modelExport.createElement("csv_format")

#Encodage
csv_xml.setAttribute('encodage','UTF-8')

#separateur
csv_xml.setAttribute('separateur',';')

lesformats.appendChild(csv_xml)

#sig_format
sig_xml = modelExport.createElement("sig_format")

# shapefile?
sig_xml.setAttribute('extension',"Shapefile")
# SCR?
sig_xml.setAttribute('SCR',"Lambert 93")

lesformats.appendChild(sig_xml)

```



```

        QMessageBox.warning(self.dlg, 'Information', u'La date minimum
d\'observation doit être inférieure à la maximale')
        return

    self.dlg.progressBar.show()
    self.dlg.avancee.show()

    self.dlg.avancee.setText(u'1.Procédure de filtrage géographique en cours.')
    self._1_requeteGenerale()
    self.dlg.progressBar.setValue(25)

    self.dlg.avancee.setText(u'2.Procédure de filtrage attributaire en cours.')
    self._2_moissonneuse_bateuse()
    self.dlg.progressBar.setValue(40)

    self.dlg.avancee.setText(u"3.Procédure de génération des tables et CSV en
cours.")
    self._3_Fichier_CSV(typeExport = self.dlg.comboExport.currentText())
    self.dlg.progressBar.setValue(50)

    self.dlg.avancee.setText(u"4.Procédure d'export des éléments géographiques en
cours")
    self._4_Export_SIG(typeExport = self.dlg.comboExport.currentText())
    self.dlg.progressBar.setValue(85)

    self.dlg.avancee.setText(u"5.Procédure de génération de métadonnées XML en
cours")
    self._5_Metadonne()
    self.dlg.progressBar.setValue(100)
    self.dlg.avancee.setText(u"Export terminé")

    print self.get_db_name()
    print self.get_host_ip()
    print self.get_table_name()
    print self.get_dbuser_name()
    print self.get_pswrd()

    QMessageBox.information(self.dlg, 'Information', u'L\'export a été réalisé avec
succès!')

    if self.dlg.checkClean.isChecked():
        self.nettoyage_des_tables()
    self.dlg.progressBar.hide()
    self.dlg.avancee.hide()
def run(self):

    #CONTROLE : verifier si une couche est bien sélectionnée
    if self.iface.activeLayer() == None :
        QMessageBox.information(self.iface.mainWindow(), 'Information', u'Veuillez
selectionner une couche à connecter')
        return
    else :
        #CONTROLE : : Si couche sélectionné est un raster => exit
        #MODIF LOIC

```

```

        if self.iface.activeLayer().type() == 2 or self.iface.activeLayer().type()
== 1:

    QMessageBox.information(self.iface.mainWindow(), 'Information', u'Veuillez selectionner
une couche vecteur')
        return

    #CONTROLE : verifier si des objets sont selectionnés
    if self.iface.activeLayer().selectedFeatureCount () <= 0 :
        QMessageBox.information(self.iface.mainWindow(), 'Information', u'Vous devez
selectionner au moins un objet')
        return

    #CONTROLE : Si ma couche ne s'appelle pas z_export, alors erreur.
    if self.iface.activeLayer().name() != u"z_export" :
        QMessageBox.information(self.iface.mainWindow(), 'Information', u'Vous devez
selectionner la couche z_export')
        return

    self.peuplerComboExport()
    self.dlg.show()

```

## Classe de Dialogue de ModulExport :

```
# -*- coding: utf-8 -*-
"""
/*****
ModuleExportDialog
                                A QGIS plugin

export
                                -----
        begin                    : 2016-04-14
        git sha                  : $Format:%H$
        copyright                : (C) 2016 by Loic Martel
        email                    : loic.martel@outlook.com
        *****/

/*****
*
*   This program is free software; you can redistribute it and/or modify
*   it under the terms of the GNU General Public License as published by
*   the Free Software Foundation; either version 2 of the License, or
*   (at your option) any later version.
*
* *****/
"""

import os

from PyQt4 import QtGui, uic
from PyQt4.Qt import QDateTime, QPushButton
from PyQt4.QtGui import QProgressBar
FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'ModuleExport_dialog_base.ui'))

class ModuleExportDialog(QtGui.QDialog, FORM_CLASS):
    def __init__(self, iface, parent=None):
        """Constructor."""
        super(ModuleExportDialog, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.setupUi(self)
        self.iface = iface

        #Forcer le passage en premeir plan
        self.setModal(1)

        #Méthodes pour passer les dates en format année ou jour
        self.date_min_pec.setDisplayFormat('yyyy')
        self.date_max_pec.setDisplayFormat('yyyy')
        self.radioSaisie.toggled.connect(lambda:
```

```

self.date_min_pec.setDisplayFormat('dd-MM-yyyy'))
    self.radioSaisie.toggled.connect(Lambda:
self.date_max_pec.setDisplayFormat('dd-MM-yyyy'))
    self.radioObs.toggled.connect(Lambda:
self.date_min_pec.setDisplayFormat('yyyy'))
    self.radioObs.toggled.connect(Lambda:
self.date_max_pec.setDisplayFormat('yyyy'))
    self.radioSaisie.toggle()

#self.ButtonFlore.setEnabled(0)
#Méthodes pour afficher ou faire disparaître les boutons de sélection d
especes..
self.rad_all_espece.toggled.connect(Lambda: self.ButtonFlore.hide())
self.rad_all_espece.toggled.connect(Lambda: self.ButtonFaune.hide())
self.rad_all_espece.toggled.connect(Lambda: self.checkFlore.hide())
self.rad_all_espece.toggled.connect(Lambda: self.checkFaune.hide())
self.rad_slc_espece.toggled.connect(Lambda: self.ButtonFaune.show())
self.rad_slc_espece.toggled.connect(Lambda: self.ButtonFlore.show())
self.rad_slc_espece.toggled.connect(Lambda: self.checkFlore.show())
self.rad_slc_espece.toggled.connect(Lambda: self.checkFaune.show())
self.rad_all_espece.toggle()
self.checkFlore.clicked.connect(Lambda: self.griser_bouton(self.ButtonFlore))
self.checkFaune.clicked.connect(Lambda: self.griser_bouton(self.ButtonFaune))

# Rendre le bouton d'export plus explicite.
self.buttonLaunch.setStyleSheet('QPushButton {background-color: #CCF390;font-
weight: bold}')
self.killButton.setStyleSheet('QPushButton {background-color: #FC9D9A;font-
weight: bold}')
self.buttonCleanExport.setStyleSheet('QPushButton {background-color:
#FC9D9A;font-weight: bold}')
self.checkClean.setStyleSheet('QCheckBox{font-weight: bold}')
#Cacher la progressbar
self.progressBar.hide()
self.avancee.hide()

def griser_bouton(self, monbouton):
    if monbouton.isEnabled() == True:
        return monbouton.setEnabled(0)
    else:
        return monbouton.setEnabled(1)

```

# Classe de dialogue de la Faune.

```
# -*- coding: utf-8 -*-
"""
/*****
DialogSelectFaune
                                A QGIS plugin

export
                                -----
begin                          : 2016-04-19
git sha                        : $Format:%H$
copyright                      : (C) 2016 by Loic Martel
email                          : loic.martel@outlook.com
*****/

/*****
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
*****/
"""

import os

from PyQt4 import QtGui, uic
from PyQt4.QtCore import *
from PyQt4.Qt import QTableWidgetItem, QAbstractItemView, QTableWidgetItem, QMessageBox
from PyQt4.QtGui import QIcon

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'ModuleExport_dialog_faune.ui'))

class DialogSelectFaune(QtGui.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(DialogSelectFaune, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.setupUi(self)
        self.setWindowTitle("Selection de la faune.")

        #définir les titres de mes colonnes
        header_labels = ["espece_id", "taxons", "nomcommun", 'class', 'ordre',
"cd_nom", "cd_ref", "rarete", "menace" ]
        self.matablewidget.setHorizontalHeaderLabels(header_labels)
        #On cache la colonne d espece

        for i in range(3,9):
```

```

        self.matablewidget.setColumnHidden(i, True)

#La selection sur la vue wref ne pourra se faire que par lignes
self.matablewidget.setSelectionBehavior(QAbstractItemView.SelectRows)
#Notre table ne doit pas être éditable
self.matablewidget.setEditTriggers(QAbstractItemView.NoEditTriggers)

header_labels = ["espece_id", "taxon", "nomcommun"]
self.tab_faune_selected.setHorizontalHeaderLabels(header_labels)
#On cache la colonne d espece
self.tab_faune_selected.setColumnHidden(2, True)

#La selection sur la vue wref ne pourra se faire que par lignes
self.tab_faune_selected.setSelectionBehavior(QAbstractItemView.SelectRows)
#Notre table ne doit pas être éditable
self.tab_faune_selected.setEditTriggers(QAbstractItemView.NoEditTriggers)
self.plugin_dir = os.path.dirname(__file__)
self.btn_ajoutfaune.setIcon(QIcon(self.plugin_dir+"/btn_add.png"))
self.btn_rmfaune.setIcon(QIcon(self.plugin_dir+"/btn_remove.png"))
self.vider_faune.setIcon(QIcon(self.plugin_dir+"/btn_suppr.png"))
self.unselect.setIcon(QIcon(self.plugin_dir+"/unselect.png"))
self.btn_visible.setIcon(QIcon(self.plugin_dir+"/visible.png"))
self.btn_invisible.setIcon(QIcon(self.plugin_dir+"/invisible.png"))

self.searchButton.setStyleSheet('QPushButton {background-color: #CCF390;font-
weight: bold}')

def visibilite(self, bool = False):
    if bool == False:
        for i in range(3,9):
            self.matablewidget.setColumnHidden(i, True)
            self.tab_faune_selected.setColumnHidden(2, True)

    if bool == True:
        for i in range(3,9):
            self.matablewidget.setColumnHidden(i, False)
            self.tab_faune_selected.setColumnHidden(2, False)

#Méthode d'ajout d'espèce dans les colonnes de selection
#On supprime les doublons également

def transvasajout__espece_faune(self):

    #CREATION D UNE LISTE ANTI DOUBLON
    #Variables qui contiennent les elements des cellules dans les rows selectionnés
    #Quadriliste de non-doublons

    comptrow = self.tab_faune_selected.rowCount()
    old_maliste0=[]
    for i in xrange(0,comptrow):
        old_maliste0.append(self.tab_faune_selected.item(i,0).text())

    #MON TRANSVASAGE
    #Controle

```



```

        if self.matablewidget.selectionModel().hasSelection() != True:
            QMessageBox.information(self, 'Information', u'Veuillez selectionner des entites a transférer.')

        else:

            #Variables qui contiennent les elements des cellules dans les rows selectionnés
            mesrows0 = self.matablewidget.selectionModel().selectedRows(0)
            mesrows1 = self.matablewidget.selectionModel().selectedRows(1)
            mesrows2 = self.matablewidget.selectionModel().selectedRows(2)

            #Quadriliste
            maliste0 = []
            for i in mesrows0:
                if i.data() not in old_maliste0:
                    maliste0.append(i.data())

            maliste1 = []
            for i in mesrows1:
                maliste1.append(i.data())

            maliste2 = []
            for i in mesrows2:
                maliste2.append(i.data())

            #On répartit les listes dans des QTableWidgetItem!
            for i in xrange(0, len(maliste0)):
                self.tab_faune_selected.insertRow(self.tab_faune_selected.rowCount())
                self.tab_faune_selected.setItem(self.tab_faune_selected.rowCount()-1,0
,QTableWidgetItem(self.tr(maliste0[i])))
                self.tab_faune_selected.setItem(self.tab_faune_selected.rowCount()-1,1
,QTableWidgetItem(maliste1[i]))
                self.tab_faune_selected.setItem(self.tab_faune_selected.rowCount()-1,2
,QTableWidgetItem(self.tr(maliste2[i])))

            #Méthode de retrait d'espèce dans la colonne de selection
            def transvasretrait_espece_faune(self):

                rows = self.tab_faune_selected.selectionModel().selectedRows()
                #liste inversée pour régler les pb d'index.
                rows = sorted(rows, reverse = True)
                for r in rows:
                    self.tab_faune_selected.removeRow(r.row())

            #SLOT pour désélectionner tout les éléments des 2 tableaux.
            def clear(self):

                #deselection tableau
                self.matablewidget.clearSelection()
                self.tab_faune_selected.clearSelection()

```

```
def peupler_table_from_base(self, tableurcible, iter):
    tableurcible.insertRow(tableurcible.rowCount())
    for i in range(0,9):
        tableurcible.setItem(tableurcible.rowCount()-1,i
, QTableWidgetItem(unicode(iter[i])))
```

```
def peupler_table_from_base_selected(self, tableurcible, iter):
    tableurcible.insertRow(tableurcible.rowCount())
    tableurcible.setItem(tableurcible.rowCount()-1,0
, QTableWidgetItem(unicode(iter[0])))
    tableurcible.setItem(tableurcible.rowCount()-1,1
, QTableWidgetItem((iter[1])))
    tableurcible.setItem(tableurcible.rowCount()-1,2
, QTableWidgetItem((iter[2])))
```

# Code SQL du module d'export.

## Récupération des ID :

```
-- Function: export._1_recupdesid(integer[])
```

```
-- DROP FUNCTION export._1_recupdesid(integer[]);
```

```
CREATE OR REPLACE FUNCTION export._1_recupdesid(param_listeid integer[])
```

```
    RETURNS void AS
```

```
$BODY$
```

```
BEGIN
```

```
-- 1 Récupération de la liste des ID
```

```
DROP table IF EXISTS ids;
```

```
CREATE TEMP TABLE ids AS(
```

```
SELECT *
```

```
FROM export.z_export e
```

```
WHERE e.id IN (select(unnest(PARAM_listeid))))
```

```
;
```

```
-- 2 Création de la table d'accueil des ID_PERM
```

```
DROP TABLE IF EXISTS export.r_id_all CASCADE;
```

```
CREATE TABLE export.r_id_all(
```

```
id_perm character varying(100),
```

```
presence boolean DEFAULT 1::boolean)
```

```
;
```

```

-- 3 Peuplement de la table avec les ID geo
INSERT INTO export.r_id_all(id_perm)
SELECT DISTINCT fpo.id_perm
FROM ids decoup,
bdfauneflore.sig_faune_poly fpo
WHERE st_intersects(fpo.geom, decoup.geom)
UNION
SELECT DISTINCT fal.id_perm
FROM ids decoup,
bdfauneflore.sig_faune_line fal
WHERE st_intersects(fal.geom, decoup.geom)
UNION
SELECT DISTINCT fpoi.id_perm
FROM ids decoup,
bdfauneflore.sig_faune_point fpoi
WHERE st_intersects(fpoi.geom, decoup.geom)
UNION
SELECT DISTINCT fll.id_perm
FROM ids decoup,
bdfauneflore.sig_flore_line fll
WHERE st_intersects(fll.geom, decoup.geom)
UNION
SELECT DISTINCT flo.id_perm
FROM ids decoup,
bdfauneflore.sig_flore_poly flo
WHERE st_intersects(flo.geom, decoup.geom)
UNION
SELECT DISTINCT flp.id_perm
FROM ids decoup,
bdfauneflore.sig_flore_point flp
WHERE st_intersects(flp.geom, decoup.geom)

```

;

--- 4 Recuperation les des sites (codesitep)

DROP TABLE if EXISTS export.r\_codes\_des\_sites;

CREATE TABLE export.r\_codes\_des\_sites AS

WITH site AS (

SELECT geo\_site.codesitep, code\_site\_mere,

st\_union(geo\_site.geom) AS geom

FROM bd\_site\_cen.geo\_site

GROUP BY geo\_site.codesitep, code\_site\_mere

)

SELECT codesitep,code\_site\_mere, site.geom

FROM site, ids decoup

WHERE st\_intersects(site.geom, decoup.geom)

;

--5idem avec les secteur => recupere secteur\_id + geom

drop table if exists export.r\_id\_des\_secteurs;

create table export.r\_id\_des\_secteurs as

select secteur\_id, secteur.geom

from bdfauneflore.secteur,ids decoup

where st\_intersects(secteur.geom, decoup.geom)

;

-- Tout les Id / meme non geo.

--6recuperer liste des idperm where : id\_perm in (ma liste d id\_perm geo) or (secteur\_id in( ma liste)) or codesite in (ma liste)...

INSERT INTO export.r\_id\_all(id\_perm)

select DISTINCT bdf.id\_perm

from bdfauneflore.t\_bilan\_flore bdf, export.r\_id\_des\_secteurs sec

where

(bdf.secteur\_id in (sec.secteur\_id))

AND bdf.type\_object not like ('sig%')

UNION

select DISTINCT bdf.id\_perm

from bdfauneflore.t\_bilan\_flore bdf, export.r\_codes\_des\_sites co

where

(bdf.codesitep in (co.codesitep) or bdf.codesitep in (co.code\_site\_mere))

AND bdf.type\_object not like ('sig%')

UNION

select DISTINCT bdf.id\_perm

from bdfauneflore.t\_bilan\_faune bdf, export.r\_id\_des\_secteurs sec

where

(bdf.secteur\_id in (sec.secteur\_id))

AND bdf.type\_object not like ('sig%')

UNION

select DISTINCT bdf.id\_perm

from bdfauneflore.t\_bilan\_faune bdf, export.r\_codes\_des\_sites co

where

(bdf.codesitep in (co.codesitep) or bdf.codesitep in (co.code\_site\_mere))

AND bdf.type\_object not like ('sig%')

;

--Creation d un index sur id\_perm

CREATE INDEX r\_id\_all\_id\_perm\_idx

ON export.r\_id\_all

USING btree

(id\_perm)

;

END

;

\$BODY\$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION export.\_1\_recupdesid(integer[])

OWNER TO postgres;

## 2. Fonction de filtrage attributaire.

-- Function: export.\_2\_filtrage(boolean, boolean, boolean, boolean, character varying, character varying, boolean, boolean, boolean)

-- DROP FUNCTION export.\_2\_filtrage(boolean, boolean, boolean, boolean, character varying, character varying, boolean, boolean, boolean);

```
CREATE OR REPLACE FUNCTION export._2_filtrage(
    param_espece boolean,
    param_faune boolean,
    param_flore boolean,
    param_typedate boolean,
    param_date_debut character varying,
    param_date_fin character varying,
    param_valide boolean,
    param_present boolean,
    param_externe boolean)
RETURNS void AS
$BODY$
```

BEGIN

```
-----
-----
----- CREER TABLE DE RETOCKAGE -----
-----
```

DROP TABLE IF EXISTS export.r\_retock CASCADE;

```
CREATE TABLE export.r_retock(
id_perm character varying(100),
raison character varying(200)
)
```

;

--Regle pour empecher l'insertion de doublons.

CREATE OR REPLACE RULE no\_noublon\_retock AS

ON INSERT TO export.r\_retock

WHERE (EXISTS ( SELECT 1

FROM export.r\_retock k

WHERE k.id\_perm = NEW.id\_perm)) DO INSTEAD NOTHING

;

```
-----
```



-----  
----- EJECTER D'OFFICE TOUTE -----  
----- DONNEE INCOMPLETE -----  
-----  
-----

---  
-- VIRER TOUTE ESPECE QUI N A PAS D ID.  
---

-- r\_retock  
INSERT INTO export.r\_retock(id\_perm, raison)  
SELECT i.id\_perm, 'Absence ID' AS raison  
FROM bdfauneflore.t\_bilan\_faune t  
JOIN export.r\_id\_all i ON (i.id\_perm = t.id\_perm)  
WHERE t.espece\_id = 0 OR t.espece\_id IS NULL  
UNION  
SELECT i.id\_perm, 'Absence ID' AS raison  
FROM bdfauneflore.t\_bilan\_flore t  
JOIN export.r\_id\_all i ON (i.id\_perm = t.id\_perm)  
WHERE t.espece\_id = 0 OR t.espece\_id IS NULL  
  
;

-- update  
UPDATE export.r\_id\_all  
SET presence = 0::boolean  
where r\_id\_all.id\_perm  
IN (  
SELECT i.id\_perm  
FROM bdfauneflore.t\_bilan\_faune t  
JOIN export.r\_id\_all i ON (i.id\_perm = t.id\_perm)  
WHERE t.espece\_id = 0 OR t.espece\_id IS NULL  
UNION  
SELECT i.id\_perm  
FROM bdfauneflore.t\_bilan\_flore t  
JOIN export.r\_id\_all i ON (i.id\_perm = t.id\_perm)  
WHERE t.espece\_id = 0 OR t.espece\_id IS NULL  
)  
  
;

```

---
-- SUPPRIMER TOUTE DATE D ANNE NULLE.
---
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'annee incomplete' AS raison
      FROM bdfauneflore.t_bilan_faune t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.date_annee = '0'
            OR t.date_annee IS NULL OR t.date_annee = ''
UNION
SELECT i.id_perm, 'annee incomplete' AS raison
      FROM bdfauneflore.t_bilan_flore t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.date_annee = '0'
            OR t.date_annee IS NULL OR t.date_annee = ''

;

```

```

-- update
UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm
IN (
      SELECT i.id_perm
      FROM bdfauneflore.t_bilan_faune t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.date_annee = '0'
            OR t.date_annee IS NULL OR t.date_annee = ''
UNION
SELECT i.id_perm
      FROM bdfauneflore.t_bilan_flore t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.date_annee = '0'
            OR t.date_annee IS NULL OR t.date_annee = ''
)

;

```

```

---
-- VIRER TOUTE ID OBS NUL.
---
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'idobs incomplet' AS raison
      FROM bdfauneflore.t_bilan_faune t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.id_obs_v = 0
            OR t.id_obs_v IS NULL

```

```

UNION
SELECT i.id_perm, 'idobs incomplet' AS raison
      FROM bdfauneflore.t_bilan_flore t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.id_obs_v = 0
            OR t.id_obs_v IS NULL

```

```
;
```

```

-- update
UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm
IN (
      SELECT i.id_perm
      FROM bdfauneflore.t_bilan_faune t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.id_obs_v = 0
            OR t.id_obs_v IS NULL
UNION
SELECT i.id_perm
      FROM bdfauneflore.t_bilan_flore t
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
      WHERE t.id_obs_v = 0
            OR t.id_obs_v IS NULL
)

```

```
;
```

```

-----
-----
----- PASSER CRITERE ESPECES -----
-----
-----

```

```

--param_espece
-- TRUE = Selection sur un pool d espece
-- FALSE = Selection de toutes les especes

```

```

--param_flore/faune
--TRUE = Selection de toute la Flore/faune
-- False = Selection de la selection dans la boite de dialog.

```

```
-- Case de choix d espece cochee
```

```
--FAUNE
```

```
-- si selection sur pool de faune SELECTIONNE.  
IF param_espece IS TRUE AND param_faune IS FALSE THEN
```

```
-- r_retock  
INSERT INTO export.r_retock(id_perm, raison)  
SELECT i.id_perm, 'espece indésirable' AS raison  
      FROM bdfauneflore.t_bilan_faune t  
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)  
      WHERE t.espece_id  
      NOT IN (SELECT espece_id  
              FROM export.w_faune_select)
```

```
;
```

```
-- update  
UPDATE export.r_id_all  
SET presence = 0::boolean  
where r_id_all.id_perm  
IN (  
    SELECT i.id_perm  
    FROM bdfauneflore.t_bilan_faune t  
    JOIN export.r_id_all i ON (i.id_perm = t.id_perm)  
    WHERE t.espece_id  
    NOT IN (SELECT espece_id  
            FROM export.w_faune_select)  
    )  
;  
END IF  
;
```

```
--FLORE  
-- si selection sur pool de flore SELECTIONNE.  
IF param_espece IS TRUE AND param_flore IS FALSE THEN
```

```
-- r_retock  
INSERT INTO export.r_retock(id_perm, raison)  
SELECT i.id_perm, 'espece indésirable' AS raison  
      FROM bdfauneflore.t_bilan_faune t  
      JOIN export.r_id_all i ON (i.id_perm = t.id_perm)  
      WHERE t.espece_id  
      NOT IN (SELECT espece_id
```

```

FROM export.w_faune_select)
;
-- Maj r_id_all non select
UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm
IN (
    SELECT i.id_perm
    FROM bdfauneflore.t_bilan_flore t
    JOIN export.r_id_all i ON (i.id_perm = t.id_perm)
    WHERE t.espece_id
    NOT IN (SELECT espece_id
            FROM export.w_flore_select)
    )
;
END IF
;

```

```

-----
-----
----- PASSER LE CRITERE DES DATES ----- OK
-----
-----
-- TRUE = Date Saisie
-- FALSE = Date Observation

```

```

-- SI SELECTION OBSERVATION
--FAUNE
IF PARAM_typedate is FALSE then

--retock
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'date d observation indésirable' AS raison
    FROM bdfauneflore.t_bilan_faune bf
    JOIN export.r_id_all i ON i.id_perm = bf.id_perm
    WHERE bf.date_annee
    NOT BETWEEN PARAM_date_debut AND PARAM_date_fin

UNION
SELECT i.id_perm, 'date d observation indésirable' AS raison
    FROM bdfauneflore.t_bilan_flore bf
    JOIN export.r_id_all i ON i.id_perm = bf.id_perm
    WHERE bf.date_annee

```

```
NOT BETWEEN PARAM_date_debut AND PARAM_date_fin  
AND i.presence <> 1::boolean
```

```
;
```

```
--update
```

```
UPDATE export.r_id_all
```

```
SET presence = 0::boolean
```

```
where r_id_all.id_perm
```

```
IN (
```

```
    SELECT i.id_perm
```

```
    FROM bdfauneflore.t_bilan_faune bf
```

```
    JOIN export.r_id_all i ON i.id_perm = bf.id_perm
```

```
    WHERE bf.date_annee
```

```
    NOT BETWEEN PARAM_date_debut AND PARAM_date_fin
```

```
    UNION
```

```
    SELECT i.id_perm
```

```
    FROM bdfauneflore.t_bilan_flore bf
```

```
    JOIN export.r_id_all i ON i.id_perm = bf.id_perm
```

```
    WHERE bf.date_annee
```

```
    NOT BETWEEN PARAM_date_debut AND PARAM_date_fin
```

```
)
```

```
;
```

```
end if
```

```
;
```

```
-- SI SELECTION SAISIE OK
```

```
IF PARAM_typedate is TRUE then
```

```
--retock
```

```
INSERT INTO export.r_retock(id_perm, raison)
```

```
SELECT i.id_perm, 'date de saisie indesirable' AS raison
```

```
FROM (
```

```
SELECT
```

```
    j.id_perm,
```

```
    CASE
```

```
        WHEN f.maj_date IS NULL THEN f.create_date
```

```
        WHEN f.create_date >= f.maj_date THEN f.create_date
```

```
        WHEN f.create_date <= f.maj_date THEN f.maj_date
```

```
        ELSE f.create_date
```

```
    END AS create_date
```

```
FROM bdfauneflore.t_bilan_faune f
```

```

JOIN export.r_id_all j ON j.id_perm = f.id_perm
WHERE create_date
NOT BETWEEN PARAM_date_debut::timestamp AND PARAM_date_fin::timestamp
UNION
SELECT
    j.id_perm,
    CASE
        WHEN f.maj_date IS NULL THEN f.create_date
        WHEN f.create_date >= f.maj_date THEN f.create_date
        WHEN f.create_date <= f.maj_date THEN f.maj_date
        ELSE f.create_date
    END AS create_date
FROM bdfauneflore.t_bilan_flore f
JOIN export.r_id_all j ON j.id_perm = f.id_perm
WHERE create_date
NOT BETWEEN PARAM_date_debut::timestamp AND PARAM_date_fin::timestamp) as i

```

```

;
--update
UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm
IN(

```

```

SELECT i.id_perm

```

```

FROM (
SELECT
    j.id_perm,
    CASE
        WHEN f.maj_date IS NULL THEN f.create_date
        WHEN f.create_date >= f.maj_date THEN f.create_date
        WHEN f.create_date <= f.maj_date THEN f.maj_date
        ELSE f.create_date
    END AS create_date
FROM bdfauneflore.t_bilan_faune f
JOIN export.r_id_all j ON j.id_perm = f.id_perm
WHERE create_date
NOT BETWEEN PARAM_date_debut::timestamp AND PARAM_date_fin::timestamp
UNION
SELECT
    j.id_perm,
    CASE
        WHEN f.maj_date IS NULL THEN f.create_date
        WHEN f.create_date >= f.maj_date THEN f.create_date

```

```

        WHEN f.create_date <= f.maj_date THEN f.maj_date
        ELSE f.create_date
    END AS create_date
FROM bdfauneflore.t_bilan_flore f
JOIN export.r_id_all j ON j.id_perm = f.id_perm
WHERE create_date
NOT BETWEEN PARAM_date_debut::timestamp AND PARAM_date_fin::timestamp) as i)

;

```

```

END IF;

```

```

-----
-----

```

```

-----
-----
----- PASSER LE CRITERE DE VALIDITE -----OK
-----
-----

```

```

-- TRUE = Uniquement les objets valides.
-- FALSE = Tout les objets

```

```

-- SI SELECTION DE Valides
-- FAUNE
IF param_valide IS TRUE THEN

```

```

--retock
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'Entite non valide indesirable' AS raison
    FROM bdfauneflore.t_bilan_faune f
    JOIN export.r_id_all i ON i.id_perm = f.id_perm
    WHERE f.valid <> 'Valide'
    UNION
    SELECT i.id_perm, 'Entite non valide indesirable' AS raison
    FROM bdfauneflore.t_bilan_flore f
    JOIN export.r_id_all i ON i.id_perm = f.id_perm
    WHERE f.valid <> 'Valide'

```



```

;
-- update
UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm
IN (
    SELECT i.id_perm
    FROM bdfauneflore.t_bilan_faune f
    JOIN export.r_id_all i ON i.id_perm = f.id_perm
    WHERE f.valid <> 'Valide'
    UNION
    SELECT i.id_perm
    FROM bdfauneflore.t_bilan_flore f
    JOIN export.r_id_all i ON i.id_perm = f.id_perm
    WHERE f.valid <> 'Valide')

;
END IF
;

-----
-----
----- PASSER LE CRITERE DE PRESENCE -----OK
-----
-----

-- TRUE = Uniquement les objets présents
-- FALSE = Tout les objets, présents ou non (field)

-- POUR LE CHAMP PRESABS VERIFIER SI 1 = PRESENT OU L INVERSE
-----
-- SI SELECTION DES OBJ PRESENTS
IF param_present IS TRUE THEN

--retock
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'Entite absente indésirable' AS raison
    FROM bdfauneflore.t_bilan_faune f
    JOIN export.r_id_all i ON i.id_perm = f.id_perm
    WHERE f.presabs <> '1'
UNION
SELECT i.id_perm, 'Entite absente indésirable' AS raison
    FROM bdfauneflore.t_bilan_flore f
    JOIN export.r_id_all i ON i.id_perm = f.id_perm
    WHERE f.presabs <> '1'

```

```

;
--update
UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm IN(
SELECT i.id_perm
      FROM bdfauneflore.t_bilan_faune f
      JOIN export.r_id_all i ON i.id_perm = f.id_perm
      WHERE f.presabs <> '1'
UNION
      SELECT i.id_perm
      FROM bdfauneflore.t_bilan_flore f
      JOIN export.r_id_all i ON i.id_perm = f.id_perm
      WHERE f.presabs <> '1')
;

END IF
;

-----
-----
----- PASSER LE PARAM DE DONNES-----OK
-----  EXTERNES OU INTERNES  -----
-----
-----
--TRUE = DONNES EXTERNES INCLUSES
-- FALSE = DONNEES EXTERNES EXCLUES
if param_externe IS FALSE THEN

--retock
INSERT INTO export.r_retock(id_perm, raison)
SELECT i.id_perm, 'Entite externe indésirable' AS raison
      FROM bdfauneflore.t_bilan_faune f
      RIGHT JOIN export.r_id_all i ON i.id_perm = f.id_perm
      WHERE f.transmis <> '0'
      UNION
SELECT i.id_perm, 'Entite externe indésirable' AS raison
      FROM bdfauneflore.t_bilan_flore f
      RIGHT JOIN export.r_id_all i ON i.id_perm = f.id_perm
      WHERE f.transmis <> '0'
;
--update

```

```

UPDATE export.r_id_all
SET presence = 0::boolean
where r_id_all.id_perm IN(
SELECT i.id_perm
      FROM bdfauneflore.t_bilan_faune f
      JOIN export.r_id_all i ON i.id_perm = f.id_perm
      WHERE f.transmis <> '0'
UNION
      SELECT i.id_perm
      FROM bdfauneflore.t_bilan_flore f
      JOIN export.r_id_all i ON i.id_perm = f.id_perm
      WHERE f.transmis <> '0'
)
;

-- Créer un index sur la colonne des zero pour accélérer les requête ultérieures.
CREATE INDEX r_id_all_id_pres_idx
  ON export.r_id_all
  USING btree
  (presence)
;

END IF
;

--+++++
--+++++
--+++++
--+++++
--+++++RECUPERER LA LISTE+++++
--+++++DES OBSERVATEURS+++++
--+++++
--+++++
--+++++
--+++++

-- Liste des observateurs virtuels présents dans pool de id faune flore

drop table if exists export.r_observateur;
create table export.r_observateur as

SELECT V.id_obs_v, w.titre

```

```
FROM bdfauneflore.asobs a
JOIN (SELECT i.id_perm, f.id_obs_v
      FROM bdfauneflore.t_bilan_faune f
      RIGHT JOIN export.r_id_all i ON i.id_perm = f.id_perm
      UNION
      SELECT i.id_perm, f.id_obs_v
      FROM bdfauneflore.t_bilan_flore f
      RIGHT JOIN export.r_id_all i ON i.id_perm = f.id_perm) as V
ON (a.id_obs_v = V.id_obs_V)
JOIN bdfauneflore.w_observateur w on (a.id_obs_r = w.id)
GROUP BY w.id, w.titre, V.id_obs_v
ORDER BY w.titre
```

;

END

;

\$BODY\$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION export.\_2\_filtrage(boolean, boolean, boolean, boolean, character varying,  
character varying, boolean, boolean, boolean)

OWNER TO postgres;

### 3. Fonction d'export INPN.

```
-- Function: export._exportinpn()

-- DROP FUNCTION export._exportinpn();

CREATE OR REPLACE FUNCTION export._exportinpn()
  RETURNS void AS
$BODY$

BEGIN
-----
-----
---- CREER MODELE SUR LA FAUNE ----
----- Et FLORE -----
-----

DROP TABLE IF EXISTS export.e_St_Principal;
Create TABLE export.e_St_Principal as(
--CLEOBJ
--Constitution d une liste de tout nos IDPerm.
WITH T_cleObs as (
SELECT i.id_perm
FROM export.r_id_all i
WHERE i.presence = 1::boolean

)
,
-- CHAMPS BRUTS
-- Une table qui contient les champs "tels quels" de bilan
T_inchanged as(
SELECT i.id_perm, t.cd_ref as cdRef, t.cd_nom as cdNom, t.taxon as nomCite
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm, t.cd_ref as cdRef, t.cd_nom as cdNom, t.taxon as nomCite
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean)
,
--STATSOURCE
T_statSource as(
SELECT i.id_perm,
      CASE
        WHEN t.origine ilike 'Donnée terrain' THEN 'Te'
        WHEN t.origine ilike 'Donnée Biblio' THEN 'Li'
```

```

        WHEN t.origine ilike 'Com.pers.' THEN 'NSP'
        ELSE 'NSP'
    END as statSource
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
    CASE
        WHEN t.origine ilike 'Donnée terrain' THEN 'Te'
        WHEN t.origine ilike 'Donnée Biblio' THEN 'Li'
        WHEN t.origine ilike 'Com.pers.' THEN 'NSP'
        ELSE 'NSP'
    END as statSource
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)
,

-- REF BIBLIO
-- RIEN

--JDD CODE
-- Metadata

-- ID ORIGINE
-- AJOUTER export.r_faune.id_perm lors de la fusion as idOrigin

-- ORGGESTDAT

T_orgagest as(
SELECT i.id_perm,
    CASE
        WHEN t.source_type IS NULL THEN 'Nsp'
        WHEN t.source_type <> 'prestation' THEN
            CASE
                WHEN t.source_org = " OR t.source_org IS NULL THEN
'Nsp'
                ELSE t.source_org
            END
        WHEN t.source_type ilike 'prestation' THEN 'CEN Picardie' --CenPicardie
    END as orgGestDat

```

```

FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
       CASE
         WHEN t.source_type IS NULL THEN 'Nsp'
         WHEN t.source_type <> 'prestation' THEN
           CASE
             WHEN t.source_org = " OR t.source_org IS NULL THEN
               'Nsp'
             ELSE t.source_org
           END
         WHEN t.source_type ilike 'prestation' THEN 'CEN Picardie' --CenPicardie
       END as orgGestDat
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)
,

```

```

-- DSPUBLIC
T_dsPublic as(

```

```

SELECT i.id_perm,
       CASE
         WHEN t.source_org ilike 'CEN Picardie' AND t.source_type ilike 'bénévole' THEN 'Pr'
         WHEN t.source_org ilike 'CEN Picardie' AND t.source_type ilike 'interne' THEN 'Re'
         WHEN t.source_type <> 'CEN Picardie' AND t.source_type <> 'prestation' THEN 'Pr'
         WHEN (t.source_type IS NULL OR t.source_type = "") AND (t.source_org IS NULL OR
t.source_org=") THEN 'Nsp'
         WHEN t.source_type ilike 'prestation' THEN 'Ac'
         ELSE t.source_type
       END as DsPublic
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
       CASE
         WHEN t.source_org ilike 'CEN Picardie' AND t.source_type ilike 'bénévole' THEN 'Pr'
         WHEN t.source_org ilike 'CEN Picardie' AND t.source_type ilike 'interne' THEN 'Re'
         WHEN t.source_type <> 'CEN Picardie' AND t.source_type <> 'prestation' THEN 'Pr'
         WHEN (t.source_type IS NULL OR t.source_type = "") AND (t.source_org IS NULL OR

```

```

t.source_org=") THEN 'Nsp'
    WHEN t.source_type ilike 'prestation' THEN 'Ac'
    ELSE t.source_type
END as DsPublic
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)

,

-- DEBUG select * from dsPublic where dspublic like " or dspublic is NULL


-- STATOBS
T_StatObs As(
SELECT i.id_perm,
    CASE
        WHEN t.presabs = '0' THEN 'No'
        WHEN t.presabs = '1' THEN 'Pr'
        WHEN t.presabs IS NULL OR t.presabs="" THEN 'Nsp'
        ELSE 'Nsp'
    END as statObs

FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
    CASE
        WHEN t.presabs = '0' THEN 'No'
        WHEN t.presabs = '1' THEN 'Pr'
        WHEN t.presabs IS NULL OR t.presabs="" THEN 'Nsp'
        ELSE 'Nsp'
    END as statObs
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)

,

-- VTAXREF

```



-- v9.0

```
T_vTAXREF as(
SELECT i.id_perm, 'v9.0' as vTAXREF
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm, 'v9.0' as vTAXREF
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)

,
-- DENBRMIN / MAX
-- si eff_min/max exist then eff_min/max | sinon denbrmin/max = effectif
T_Denbrmin as(
SELECT i.id_perm,
CASE
WHEN t.ffmpeg = " OR t.ffmpeg IS NULL THEN t.effectif
ELSE t.ffmpeg
END as denbrMin
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
CASE
WHEN t.ffmpeg = " OR t.ffmpeg IS NULL THEN t.effectif
ELSE t.ffmpeg
END as denbrMin
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)

,
T_Denbrmax as(
SELECT i.id_perm,
CASE
WHEN t.ffmpeg = " OR t.ffmpeg IS NULL THEN t.effectif
ELSE t.ffmpeg
END as denbrMax
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
```

```

WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
       CASE
         WHEN t.effmax = " OR t.effmax IS NULL THEN t.effectif
         ELSE t.effmax
       END as denbrMax
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean)
,

-- OBJDENBR-----+++++++-----
-- Voir avec gratien
T_objdenbr as(
WITH brut AS(
SELECT i.id_perm, u.unite_dee
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON i.id_perm = t.id_perm
left JOIN bdfauneflore.w_unite u ON u.unite = t.unite
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm, u.unite_dee
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON i.id_perm = t.id_perm
left JOIN bdfauneflore.w_unite u ON u.unite = t.unite
WHERE i.presence = 1::boolean
)
SELECT b.id_perm,
       CASE
         WHEN b.unite_dee = " OR b.unite_dee IS NULL THEN 'AUTR'
         ELSE b.unite_dee
       END as objDenbr
FROM brut b
)
,

-- Commentaire
-- non

-- --DATETEBU.---

```

```

T_dateDebut as(
    WITH jourplein as(
        select i.id_perm,
            CASE
                WHEN t.date_jour = " Or t.date_jour is null THEN '01'
                ELSE t.date_jour
            END jp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        select i.id_perm,
            CASE
                WHEN t.date_jour = " Or t.date_jour is null THEN '01'
                ELSE t.date_jour
            END jp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean),
    moisplein as(
        select i.id_perm,
            CASE
                WHEN t.date_mois = " Or t.date_mois is null THEN '01'
                ELSE t.date_mois
            END mp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        select i.id_perm,
            CASE
                WHEN t.date_mois = " Or t.date_mois is null THEN '01'
                ELSE t.date_mois
            END mp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean),
    annee as(
        SELECT i.id_perm, t.date_annee
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        SELECT i.id_perm, t.date_annee
        FROM export.r_id_all i

```

```

        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean)
    select i.id_perm, to_char(((j.jp || '/' || m.mp || '/' || a.date_annee)::date), 'DD/MM/YYYY') as
dateDebut
    from export.r_id_all i
    Join jourplein j on (i.id_perm = j.id_perm)
    Join moisplein m on (i.id_perm = m.id_perm)
    join annee a on (i.id_perm = a.id_perm)
)
,

```

```

--DATEFIN
T_dateFin as(

```

```

    WITH moisplein as(
        select i.id_perm,
            CASE
                WHEN t.date_mois = " Or t.date_mois is null THEN '12'
                ELSE t.date_mois
            END mp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean
        UNION
        select i.id_perm,
            CASE
                WHEN t.date_mois = " Or t.date_mois is null THEN '12'
                ELSE t.date_mois
            END mp
        FROM export.r_id_all i
        JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
        WHERE i.presence = 1::boolean)

```

```

    ,
    jourplein as(
        select i.id_perm,
            CASE
                WHEN t.date_jour = " Or t.date_jour is null
                THEN
                CASE
                    WHEN moisplein.mp IN('01','03','05','07','08','10','12')
                    THEN '31'
                    WHEN moisplein.mp IN ('04','06','09','11')
                    THEN '30'
                    WHEN moisplein.mp = '02' THEN
                        CASE

```

```

        WHEN
        t.date_annee::integer % 4 = 0
        AND
        t.date_annee::integer % 100 <> 0
        OR
        t.date_annee::integer % 400 = 0
        THEN '29'
        ELSE '28'
        END
    END

    ELSE t.date_jour
END jp
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
JOIN moisplein ON moisplein.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
select i.id_perm,
    CASE
        WHEN t.date_jour = " Or t.date_jour is null
        THEN
        CASE
            WHEN moisplein.mp IN('01','03','05','07','08','10','12')
            THEN '31'
            WHEN moisplein.mp IN ('04','06','09','11')
            THEN '30'
            WHEN moisplein.mp = '02' THEN
                CASE
                    WHEN
                    t.date_annee::integer % 4 = 0
                    AND
                    t.date_annee::integer % 100 <> 0
                    OR
                    t.date_annee::integer % 400 = 0
                    THEN '29'
                    ELSE '28'
                    END
            END
        END

    ELSE t.date_jour
END jp
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
JOIN moisplein ON moisplein.id_perm = i.id_perm
WHERE i.presence = 1::boolean),

```

```

annee as(
    SELECT i.id_perm, t.date_annee
    FROM export.r_id_all i
    JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
    WHERE i.presence = 1::boolean
    UNION
    SELECT i.id_perm, t.date_annee
    FROM export.r_id_all i
    JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
    WHERE i.presence = 1::boolean)

```

```

select i.id_perm, to_char(((j.jp || '/' || m.mp || '/' || a.date_annee)::date), 'DD/MM/YYYY') as
dateFin
from export.r_id_all i
Join jourplein j on (i.id_perm = j.id_perm)
Join moisplein m on (i.id_perm = m.id_perm)
join annee a on (i.id_perm = a.id_perm)
)
,

```

```

--natObjGeo : voir champ type_objetc données issus des couche sig_xxx => 'ST' | données Table =>
'In'
T_natObjGeo AS(
SELECT i.id_perm,
    CASE
        WHEN t.type_object LIKE 'sig_%' THEN 'St'
        WHEN t.type_object LIKE 't_obs%' THEN 'In'
        ELSE t.type_object
    END as natObjGeo
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm,
    CASE
        WHEN t.type_object LIKE 'sig_%' THEN 'St'
        WHEN t.type_object LIKE 't_obs%' THEN 'In'
        ELSE t.type_object
    END as natObjGeo
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)

```

```

        END as natObjGeo
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)
,

```

```

-- difnotepre : 1 partt pr l inst.

```

```

T_diff as(
SELECT i.id_perm, 1 AS diflocpre, 1 AS diffMaille, 1 AS diffCom, 1 AS diffTotale, 0 AS floutage
FROM export.r_id_all i
WHERE i.presence =1::boolean)
,

```

```

-- identObs
tmp AS (
    SELECT t1.id_obs_v,
           string_agg(t2.titre_court::text, ' ' '::text) AS identObs,
           string_agg(COALESCE(t2.observateur_organisme::text,'INCONNU'), ' ' '::text) as orgObs
    FROM bdfauneflore.asobs t1
    JOIN bdfauneflore.w_observateur t2 ON t1.id_obs_r = t2.id
    GROUP BY t1.id_obs_v
)
,

```

```

T_Observateurs AS(
select
t.id_perm,
tmp.id_obs_v,
tmp.orgObs,
tmp.identObs
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
LEFT JOIN tmp ON (t.id_obs_v = tmp.id_obs_v)
WHERE i.presence =1::boolean

```

```

UNION

```

```

select
t.id_perm,
tmp.id_obs_v,
tmp.orgObs,
tmp.identObs

```

```

FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
LEFT JOIN tmp ON (t.id_obs_v = tmp.id_obs_v)
WHERE i.presence = 1::boolean)

```

,

```

-- nivValid = Ref au champ [Valide], 'Valide' = 1, 'a confirmer' = 2 'invalide' = 4
T_nivValide AS(
SELECT i.id_perm,
CASE
WHEN t.valid ilike 'Valide' THEN '1'
WHEN t.valid ilike 'A confirmer' THEN '2'
WHEN t.valid ilike 'Invalide' THEN '4'
WHEN t.valid ilike " OR t.valid IS NULL THEN '6' --En cas de vide 'non évalué'
ELSE t.valid
END nivValid
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean

```

UNION

```

SELECT i.id_perm,
CASE
WHEN t.valid ilike 'Valide' THEN '1'
WHEN t.valid ilike 'A confirmer' THEN '2'
WHEN t.valid ilike 'Invalide' THEN '4'
WHEN t.valid ilike " OR t.valid IS NULL THEN '6' --En cas de vide 'non évalué'
ELSE t.valid
END nivValid
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
)

```

,

-- Cleobject = lien vers objet geographiques

```

T_cleobjet as (
select i.id_perm,
CASE
WHEN t.type_object ilike 'sig_%' then i.id_perm
WHEN t.type_object ilike 't_obs_%' and (t.secteur_id is not null and secteur_id <>")
then t.secteur_id

```



```

                WHEN t.type_object ilike 't_obs_%' and (t.secteur_id is null or secteur_id = "") then
t.codesitep
                ELSe 'Nsp'
            END cleobjet

FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean

UNION

select i.id_perm,
       CASE
           WHEN t.type_object ilike 'sig_%' then i.id_perm
           WHEN t.type_object ilike 't_obs_%' and (t.secteur_id is not null and secteur_id <>")
then t.secteur_id
           WHEN t.type_object ilike 't_obs_%' and (t.secteur_id is null or secteur_id = "") then
t.codesitep
           ELSe 'Nsp'
       END cleobjet

FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean

)

```

```

-----
-----
-----RASSEMBLEMENT-----
-----
-----

```

```

SELECT
    c.id_perm as cleObs,
    T_statSource.statsource::character varying(3),
    T_orgagest.orggestdat,
    T_dsPublic.dspublic,
    T_StatObs.statobs::character varying(3),
    T_inchanged.cdref,
    T_inchanged.cdnom,
    T_vTAXREF.vtaxref::character varying(10),
    T_inchanged.nomcite,
    T_Denbrmin.denbrmin,
    T_Denbrmax.denbrmax,

```

T\_objdenbr.objdenbr,  
 T\_dateDebut.datedebut::character varying(10),  
 T\_dateFin.datefin::character varying(10),  
 T\_cleobjet.cleobjet as cleObjet,  
 T\_natObjGeo.natobjgeo::character varying(3),  
 T\_diff.diflocpre,  
 T\_diff.difffmaille,  
 T\_diff.difffcom,  
 T\_diff.difftotale,  
 T\_diff.floutage,  
 T\_Observateurs.identobs,  
 T\_Observateurs.orgObs,  
 T\_nivValide.nivvalid

FROM T\_cleObs as c  
 JOIN T\_statSource ON (T\_statSource.id\_perm = c.id\_perm)  
 JOIN T\_orgagest ON (T\_orgagest.id\_perm = c.id\_perm)  
 JOIN T\_dsPublic ON (T\_dsPublic.id\_perm = c.id\_perm)  
 JOIN T\_StatObs ON (T\_StatObs.id\_perm = c.id\_perm)  
 JOIN T\_vTAXREF ON (T\_vTAXREF.id\_perm = c.id\_perm)  
 JOIN T\_Denbrmin ON (T\_Denbrmin.id\_perm = c.id\_perm)  
 JOIN T\_Denbrmax ON (T\_Denbrmax.id\_perm = c.id\_perm)  
 JOIN T\_objdenbr ON (T\_objdenbr.id\_perm = c.id\_perm)  
 JOIN T\_dateDebut ON (T\_dateDebut.id\_perm = c.id\_perm)  
 JOIN T\_dateFin ON (T\_dateFin.id\_perm = c.id\_perm)  
 JOIN T\_natObjGeo ON (T\_natObjGeo.id\_perm = c.id\_perm)  
 JOIN T\_diff ON (T\_diff.id\_perm = c.id\_perm)  
 JOIN T\_inchanged ON (T\_inchanged.id\_perm = c.id\_perm)  
 JOIN T\_cleobjet ON (T\_cleobjet.id\_perm = c.id\_perm)  
 LEFT JOIN T\_Observateurs ON (T\_Observateurs.id\_perm = c.id\_perm) --LEFT JOIN CAR 1 OBS  
 VIDE  
 JOIN T\_nivValide ON (T\_nivValide.id\_perm = c.id\_perm)

--fin de TABLE ST PRINCIAPL

;

Copy (Select \* From export.e\_St\_Principal) to  
 'D:\sig\00\_MODULE\_EXPORT\_BDCEN\exports\St\_principal.csv' With DELIMITER ';' CSV  
 HEADER

;

-- CSV ATTRADD

DROP TABLE IF EXISTS export.e\_St\_AttrAdd;

CREATE TABLE export.e\_St\_AttrAdd--creation de table d accueil

(CleObs character varying(100),  
NomParam character varying(45),  
Parametre character varying(45),  
typeParam character varying(45),  
Valeur character varying(45),  
unitParam character varying(45)  
)

;

--SEXE

INSERT INTO export.e\_St\_AttrAdd(CleObs,NomParam,Parametre,TypeParam,Valeur)

-- seule la faune est sexuée

(SELECT i.id\_perm, 'sexe' AS nomParam, 'Sexe de l individu' AS Parametre, 'QUAL' AS  
TypeParam, t.sexe

FROM export.r\_id\_all i

JOIN bdfauneflore.t\_bilan\_faune t ON t.id\_perm = i.id\_perm

WHERE i.presence = 1::boolean

AND t.sexe IS NOT NULL

AND t.sexe != "")

;

--Age

INSERT INTO export.e\_St\_AttrAdd(CleObs,NomParam,Parametre,TypeParam,Valeur, unitParam)

-- Seule la faune a un age

(SELECT i.id\_perm, 'Age' AS nomParam, 'Age l individu' AS Parametre, 'QTA' AS TypeParam,  
t.age, 'syst CenPic' as unitParam

FROM export.r\_id\_all i

JOIN bdfauneflore.t\_bilan\_faune t ON t.id\_perm = i.id\_perm

WHERE i.presence = 1::boolean

AND t.age IS NOT NULL

AND t.age != "")

;

--comportement

INSERT INTO export.e\_St\_AttrAdd(CleObs,NomParam,Parametre,TypeParam,Valeur)

--Seul la faune a un comportement releve.

(SELECT i.id\_perm, 'Comportement' AS nomParam, 'comportement de l individu' AS Parametre,  
'QUAL' AS TypeParam, t.comportement

FROM export.r\_id\_all i

JOIN bdfauneflore.t\_bilan\_faune t ON t.id\_perm = i.id\_perm

WHERE i.presence = 1::boolean

AND t.comportement IS NOT NULL

AND t.comportement != "")

;

```

--Unite
INSERT INTO export.e_St_AttrAdd(CleObs,NomParam,Parametre,TypeParam,Valeur)
(SELECT i.id_perm, 'Unite' AS nomParam, 'Unite' AS Parametre, 'QUAL' AS TypeParam, t.unite
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
AND t.unite IS NOT NULL
AND t.unite != "
UNION
SELECT i.id_perm, 'Unite' AS nomParam, 'Unite' AS Parametre, 'QUAL' AS TypeParam, t.unite
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
WHERE i.presence = 1::boolean
AND t.unite IS NOT NULL
AND t.unite != ")
--concerne faune flore
;
Copy (Select * From export.e_St_AttrAdd) to
'D:\sig\00_MODULE_EXPORT_BDCEN\exports\St_AttrAdd.csv' With DELIMITER ';' CSV
HEADER
;

-- CSV ST_EN
DROP TABLE IF EXISTS export.e_St_EN;
CREATE TABLE export.e_St_EN AS(--creation de table d accueil
select id_perm as cleobs, 'SCEN' as typeEN , s.id_mnhn as codeEN, 2 as typinfGeo
from bdfauneflore.t_bilan_faune t
join bd_site_cen.site_cen s on s.code_site_mere = t.codesitep
where s.id_mnhn is not null)
;
Copy (Select * From export.e_St_EN) to
'D:\sig\00_MODULE_EXPORT_BDCEN\exports\St_EN.csv' With DELIMITER ';' CSV HEADER
;

-----
-----
-----
----- CREATION DES COUCHES SIG -----
-----
-----
-----

--r_st_sig_poly (union des polygone issu des couche sig_xxx + geom des sites moissonnés + geom

```

```

des secteur moissoné)
DROP TABLE IF EXISTS export.e_st_sig_poly;
CREATE TABLE export.e_st_sig_poly AS(
WITH geo_site AS (--reconstitution des sitep avec union
    SELECT geo_site.codesitep,
           st_union(geo_site.geom) AS geom
    FROM bd_site_cen.geo_site
    GROUP BY geo_site.codesitep)

,
r_all_poly AS(
-- Set réduit & compact r_

SELECT i.id_perm, t.secteur_id, t.codesitep, p.geom, t.type_object
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_faune_poly p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
-----doubler
SELECT i.id_perm, t.secteur_id, t.codesitep, p.geom, t.type_object
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_flore_poly p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean)

,

temp_st_sig_poly AS(

select -- Liste des geometry unique SIG_poly_xxx
id_perm as cleObjet,
geom as geom
from r_all_poly

union

select s.secteur_id as cleObjet, -- Liste des secteur avec données flore
s.geom as geom
from export.r_id_all r
join bdfauneflore.t_bilan_flore t on t.id_perm = r.id_perm
join bdfauneflore.secteur s on t.secteur_id = s.secteur_id
Where r.presence = 1::boolean and t.type_object like 't_obs%'
group by s.secteur_id, s.geom

```

union

```
select s.secteur_id as cleObjet, -- Liste des secteur avec données faune (données en table
uniquement)
s.geom as geom
from export.r_id_all r
join bdfauneflore.t_bilan_faune t on t.id_perm = r.id_perm
join bdfauneflore.secteur s on t.secteur_id = s.secteur_id
Where r.presence = 1::boolean and t.type_object like 't_obs%'
group by s.secteur_id, s.geom
```

union

```
select
s.codesitep as cleObjet,-- Liste des sites avec données flore (données en table uniquement)
s.geom
from export.r_id_all r
join bdfauneflore.t_bilan_flore t on t.id_perm = r.id_perm
join geo_site s on t.codesitep = s.codesitep
where r.presence = 1::boolean and t.type_object like 't_obs%' and (t.secteur_id is null or t.secteur_id
= "")
group by s.codesitep, s.geom
```

union

```
select
s.codesitep as cleObjet,-- Liste des sites avec données faune (données en table uniquement)
s.geom
from export.r_id_all r
join bdfauneflore.t_bilan_faune t on t.id_perm = r.id_perm
join geo_site s on t.codesitep = s.codesitep
where r.presence = 1::boolean and t.type_object like 't_obs%' and (t.secteur_id is null or t.secteur_id
= "")
group by s.codesitep, s.geom
```

)

```
SELECT te.cleobjet, 'false' as objCompos, te.geom FROM temp_st_sig_poly te)
;
```

--g\_st\_sig\_point

```
DROP TABLE IF EXISTS export.e_st_sig_point;
CREATE TABLE export.e_st_sig_point AS(
```

```

SELECT i.id_perm as cle_objet, 'false' AS objCompos, p.geom
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_faune_point p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm as cle_objet, 'false' AS objCompos, p.geom
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_flore_point p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean)
;

```

```

--r_st_sig_lignes
DROP TABLE IF EXISTS export.e_st_sig_lignes;
CREATE TABLE export.e_st_sig_lignes AS(
SELECT i.id_perm as cle_objet, 'false' AS objCompos, p.geom
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_flore t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_flore_line p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean
UNION
SELECT i.id_perm as cle_objet, 'false' AS objCompos, p.geom
FROM export.r_id_all i
JOIN bdfauneflore.t_bilan_faune t ON t.id_perm = i.id_perm
JOIN bdfauneflore.sig_faune_line p ON p.id_perm = i.id_perm
WHERE i.presence = 1::boolean)
;
END;

```

```

$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
ALTER FUNCTION export._exportinpn()
OWNER TO postgres;

```

## 4. Fonction de suppression de tables.

```
-- Function: export.kill_table()

-- DROP FUNCTION export.kill_table();

CREATE OR REPLACE FUNCTION export.kill_table()
  RETURNS void AS
$BODY$

declare row record;

BEGIN

for row in
    select *
    from pg_tables p
    where p.schemaname = 'export'
    and tablename not ilike 'w_%'
    and tablename not ilike 'z_%'
    and tablename not ilike 'p_%'
loop
    begin
        raise notice 'requete effectué sur(%)', row.tablename;
        execute 'DROP TABLE export.' || row.tablename || ' cascade';

    end;

end loop;

END;

$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
ALTER FUNCTION export.kill_table()
  OWNER TO postgres;
```



