Mise à jour de GeoNature Atlas

Requêtes liés aux zones géographiques de l'Atlas

L'Atlas nécessite de définir un territoire correspondant à l'emprise géographique des observations stockées dans le module Synthèse de GeoNature.

Dans le cadre du SINP PACA, nous avons gérées la définition du territoire et la suppression des zones géographiques inutiles via un script de gestion des zones géographiques liées à un territoire.

Dans le cadre du SINP AURA, un script basé sur le même principe est aussi disponible mais nous avons préféré utiliser le référentiel géographique construit par Flavia dans leur GeoNature. En effet, les contours des départements et de la région ont été construits à l'aide des contours des communes. Cela offre l'avantage d'avoir une parfaite superposition de ces différentes entités géographiques. De plus, l'intégration de différentes entités géographiques fournit par l'INPN a été réalisé et l'ensemble des entités géographiques hors région AURA supprimées.

Vous pouvez consulter les requêtes initialement exécutées pour créer le territoire et supprimer des zones géographiques dans le cadre du SINP PACA.

Adapatation des vues

Pour AURA

- Ressources : Script SQL dans le dépôt Github SINP AURA Data
- Se connecter au serveur "db-srv" en tant que *geonat* : `ssh geonat@db-<region>-sinp`
- Se placer dans le dossier *~/data* : `cd ~/data`
- Mettez à jour la VM `vm_observations` avec : ''psql -h localhost -U geonatadmin -d gnatlas -f ./db-atlas/data/sql/01_update_vm_observations.sql ''
- Corriger la base *gnatlas* en tant que super utilisateur : '' sudo -u postgres -s psql -d gnatlas -f ./db-atlas/data/sql/03_fix_as_superuser.sql ''
- Corriger la base *gnatlas* en tant que propriétaire : '' psql -h localhost -U geonatadmin -d gnatlas -f ./db-atlas/data/sql/04_fix_as_user.sql ''

Pour PACA

- Augmenter le nombre ligne récupérable en une seule fois par Postgresql au niveau des tables étrangères (*foreign table*), par défaut `100` le passé à 100 000 :

```
ALTER SERVER geonaturedbserver OPTIONS (SET fetch_size '100000');
```

  - Si l'option n'a jamais été ajouté, le faire avec :

```
ALTER SERVER geonaturedbserver OPTIONS (ADD fetch_size '100000');
```

- Remplir les la colonne `centroid` de la table `ref_geo.l_areas` car elle est utilisé par la vue `synthese.syntheseff` de l'Atlas. Or par défaut, cette colonne est vide :

```
UPDATE ref_geo.l_areas
SET centroid = ST_Centroid(geom)
WHERE centroid IS NULL ;
```

- TRAVAIL EN COURS : Modifier la vue `synthese.syntheseff` car la requête par défaut ne fonctionne pas avec 46 millions de lignes dans la table `cor_area_synthese`. C'est la requêtes associant les id_synthese avec les différents types de zone géo qui posent problème.
L'affichage dans l'Atlas se faisant sous forme de mailles, nous allons supprimé cette partie le temps de trouver une solution :

```
CREATE OR REPLACE VIEW synthese.syntheseff AS
SELECT
    s.id_synthese,
    s.id_dataset,
    s.cd_nom,
    s.date_min AS dateobs,
    s.observers AS observateurs,
    (s.altitude_min + s.altitude_max) / 2 AS altitude_retenue,
    st_transform(s.the_geom_point, 4326) AS the_geom_point,
    s.count_min AS effectif_total,
    c.insee,
    dl.cd_nomenclature::INTEGER AS diffusion_level
FROM synthese.synthese s
    JOIN atlas.l_communes AS c
        ON ( st_intersects(s.the_geom_point, c.the_geom) )
    LEFT JOIN synthese.t_nomenclatures AS dl
        ON (s.id_nomenclature_diffusion_level = dl.id_nomenclature)
    LEFT JOIN synthese.t_nomenclatures AS st
        ON (s.id_nomenclature_observation_status = st.id_nomenclature)
WHERE ( NOT dl.cd_nomenclature::text = '4'::text OR
s.id_nomenclature_diffusion_level IS NULL )
    AND st.cd_nomenclature::text = 'Pr'::TEXT ;
```

- À partir de la v1.5 de l'Atlas, la VM `atlas.vm_communes` se comporte normalement. Elle met toujours un peu de temps à se générer (15mn en local sur disque SSD Nvme). Avant la version 1.5 de l'Atlas, la VM `atlas.vm_communes` ne contient pas toutes les communes du territoire. Il semblerait que les communes en bord de mer ne soient pas prise en compte même avec l'utilisation de la fonction Postgis `st_buffer()`... Le problème vient peut être des îles des communes côtières. De plus, la génération de son contenu prend énormément de temps, nous pouvons donc utiliser la subdivision du territoire à la place. Au final, nous pouvons utiliser à la place :

```
DROP MATERIALIZED VIEW IF EXISTS atlas.vm_stats ;
DROP MATERIALIZED VIEW IF EXISTS atlas.vm_communes ;

CREATE MATERIALIZED VIEW atlas.vm_communes
TABLESPACE pg_default
```

```
AS
    SELECT DISTINCT
        c.insee,
        c.commune_maj,
        c.the_geom,
        c.commune_geojson
    FROM atlas.l_communes c
        JOIN atlas.t_subdivided_territory t
            ON (st_intersects(t.geom, c.the_geom))
WITH DATA ;

-- View indexes:
CREATE UNIQUE INDEX vm_communes_insee_idx ON atlas.vm_communes USING
btree(insee) ;
CREATE INDEX vm_communes_commune_maj_idx ON atlas.vm_communes USING
btree(commune_maj) ;
CREATE INDEX index_gist_vm_communes_the_geom ON atlas.vm_communes USING
gist(the_geom) ;

CREATE MATERIALIZED VIEW atlas.vm_stats
TABLESPACE pg_default
AS SELECT 'observations'::text AS label,
    COUNT(*) AS RESULT
   FROM atlas.vm_observations
UNION
 SELECT 'municipalities'::text AS label,
    COUNT(*) AS RESULT
   FROM atlas.vm_communes
UNION
 SELECT 'taxons'::text AS label,
    COUNT(DISTINCT vm_taxons.cd_ref) AS RESULT
   FROM atlas.vm_taxons
UNION
 SELECT 'pictures'::text AS label,
    COUNT(DISTINCT m.id_media) AS RESULT
   FROM atlas.vm_medias m
     JOIN atlas.vm_taxons t ON t.cd_ref = m.cd_ref
  WHERE m.id_type = ANY (ARRAY[1, 2])
WITH DATA ;

-- View indexes:
CREATE UNIQUE INDEX vm_stats_label_idx ON atlas.vm_stats USING
btree(label) ;

-- Restore permissions
GRANT SELECT ON TABLE atlas.vm_communes TO geonatatlas ;
GRANT SELECT ON TABLE atlas.vm_stats TO geonatatlas ;
```

- Ajout d'un index sur la VM `atlas.vm_observations_mailles` avec :

```
CREATE UNIQUE INDEX vm_observations_mailles_id_obs_geojson_idx ON
```

```
atlas.vm_observations_mailles USING btree (id_observation,
geojson_maille) ;
```

- Solution alternative à la vue syntheseff qui utilise une vue matérialisée :

```
CREATE MATERIALIZED VIEW atlas.cor_synthese_area
TABLESPACE pg_default
AS
    SELECT DISTINCT ON (sa.id_synthese, t.type_code)
        sa.id_synthese,
        sa.id_area,
        st_transform(a.centroid, 4326) AS centroid_4326,
        t.type_code,
        a.area_code
    FROM synthese.cor_area_synthese AS sa
         JOIN ref_geo.l_areas AS a
             ON (sa.id_area = a.id_area)
         JOIN ref_geo.bib_areas_types AS t
             ON (a.id_type = t.id_type)
    WHERE t.type_code IN ('M10', 'COM', 'DEP')
WITH DATA;

-- View indexes:
CREATE UNIQUE INDEX cas_pk_idx ON atlas.cor_synthese_area USING btree
(id_synthese, id_area);
CREATE INDEX cas_type_code_idx ON atlas.cor_synthese_area USING btree
(type_code);


CREATE OR REPLACE FUNCTION
atlas.get_blurring_centroid_geom_by_code(code CHARACTER VARYING,
idSynthese INTEGER)
 RETURNS geometry
 LANGUAGE plpgsql
 IMMUTABLE
AS $function$
    -- Function which return the centroid for a sensitivity or
diffusion_level code and an id synthese
    DECLARE centroid geometry;

    BEGIN
        SELECT INTO centroid csa.centroid_4326
        FROM atlas.cor_synthese_area AS csa
        WHERE csa.id_synthese = idSynthese
            AND csa.type_code = (CASE WHEN code = '1' THEN 'COM' WHEN
code = '2' THEN 'M10' WHEN code = '3' THEN 'DEP' END)
        LIMIT 1 ;

        RETURN centroid ;
    END;
```

```
$function$
;

CREATE OR REPLACE VIEW synthese.syntheseff AS
SELECT
    s.id_synthese,
    s.id_dataset,
    s.cd_nom,
    s.date_min AS dateobs,
    s.observers AS observateurs,
    (s.altitude_min + s.altitude_max) / 2 AS altitude_retenue,
    CASE
        WHEN (sens.cd_nomenclature::INT >= 1 AND
sens.cd_nomenclature::INT <= 3 AND dl.cd_nomenclature::INT >= 1 AND
dl.cd_nomenclature::INT <= 3) THEN
            CASE
                WHEN (sens.cd_nomenclature::INT >=
dl.cd_nomenclature::INT) THEN (
atlas.get_blurring_centroid_geom_by_code(sens.cd_nomenclature,
s.id_synthese)
                )
                WHEN (sens.cd_nomenclature::INT <
dl.cd_nomenclature::INT) THEN (
atlas.get_blurring_centroid_geom_by_code(dl.cd_nomenclature,
s.id_synthese)
                )
            END
        WHEN (sens.cd_nomenclature::INT >= 1 AND
sens.cd_nomenclature::INT <= 3) AND (dl.cd_nomenclature::INT < 1 OR
dl.cd_nomenclature::INT > 3) THEN (
atlas.get_blurring_centroid_geom_by_code(sens.cd_nomenclature,
s.id_synthese)
        )
           WHEN (dl.cd_nomenclature::INT >= 1 AND
dl.cd_nomenclature::INT <= 3) AND (sens.cd_nomenclature::INT < 1 OR
sens.cd_nomenclature::INT > 3) THEN (
atlas.get_blurring_centroid_geom_by_code(dl.cd_nomenclature,
s.id_synthese)
        )
        ELSE st_transform(s.the_geom_point, 4326)
    END AS the_geom_point,
    s.count_min AS effectif_total,
    c.insee,
    sens.cd_nomenclature AS sensitivity,
    dl.cd_nomenclature AS diffusion_level
FROM synthese.synthese s
    JOIN atlas.l_communes AS c
        ON ( st_intersects(s.the_geom_point, c.the_geom) )
    LEFT JOIN synthese.t_nomenclatures AS sens
        ON (s.id_nomenclature_sensitivity = sens.id_nomenclature)
    LEFT JOIN synthese.t_nomenclatures AS dl
```

```
        ON (s.id_nomenclature_diffusion_level = dl.id_nomenclature)
    LEFT JOIN synthese.t_nomenclatures AS st
        ON (s.id_nomenclature_observation_status = st.id_nomenclature)
WHERE ( NOT dl.cd_nomenclature = '4' OR
s.id_nomenclature_diffusion_level IS NULL )
    AND ( NOT sens.cd_nomenclature = '4' OR
s.id_nomenclature_sensitivity IS NULL )
    AND st.cd_nomenclature = 'Pr' ;
```

Mise à jour de l'application

En utilisant une release de l'Altas

- **Ressources** : https://github.com/PnX-SI/GeoNature-atlas/blob/master/docs/installation.rst#mise-%C3%A0-jour-de-lapplication
- Sur "*web-srv*" :
  - Se connecter au serveur : `ssh geonat@<sinp-web>`
  - Basculer l'Atlas en mode maintenance :
    - `sudo nginx_dissite atlas.conf ; sudo nginx_ensite atlas_maintenance.conf ; sudo service nginx reload`
    - Arrêter le service de *Supervisor* avant de faire la mise à jour : `sudo supervisorctl stop atlas`
  - Se placer dans le dossier *dwl* de l'utilisateur *geonat* : `cd ~/dwl/`
  - Exporter la dernière version de GeoNature-Atals dans une variable d'env locale à la session : `export GNAV=$(curl -s https://api.github.com/repos/PnX-SI/GeoNature-atlas/releases/latest | grep tag_name | cut -d\" -f4)`
    - Vérifier la version en question : `echo "${GNAV}"`
  - Télécharger l'archive : `wget https://github.com/PnX-SI/GeoNature-atlas/archive/${GNAV}.zip -O atlas_v${GNAV}.zip`
  - Décompresser l'archive dans le dossier *www* de l'utilisateur *geonat* : `unzip atlas_v${GNAV}.zip -d ~/www/`
  - Se rendre dans le dossier *www* : `cd ~/www/`
  - Renommer le dossier au format GeoNature (⇒ uniformité): `mv GeoNature-atlas-${GNAV} atlas_v${GNAV}`
  - Pour faciliter la migration créer le lien symbolique suivant :
    - `cd ~/www; rm -f atlas_old; ln -s atlas_v<ancienne-version> atlas_old`
  - Copier les fichiers de config de l'ancien dossier vers le nouveau :
    - le fichier de paramètres pour les scripts d'installation et Gunicorn : `cp ~/www/atlas_old/atlas/configuration/settings.ini ~/www/atlas_v${GNAV}/atlas/configuration/settings.ini`
    - le fichier de config du backend Python : `cp ~/www/atlas_old/atlas/configuration/config.py ~/www/atlas_v${GNAV}/atlas/configuration/config.py`

    - Vérifier l'existence de nouveaux paramètres de config :
        - Dans *settings.ini* : '' diff ~/www/atlas_old/atlas/configuration/settings.ini.sample ~/www/atlas_v${GNAV}/atlas/configuration/settings.ini.sample ''
        - Dans *config.py* : '' diff ~/www/atlas_old/atlas/configuration/config.py.sample ~/www/atlas_v${GNAV}/atlas/configuration/config.py.sample''
    - Copier le contenu du dossier "*static*" : '' cp -aR ~/www/atlas_old/static/custom/ ~/www/atlas_v${GNAV}/static ''
    - Modifier le lien symbolique pour pointer vers la nouvelle version : `cd ~/www; rm -f atlas; ln -s "atlas_v${GNAV}" atlas`
    - Consulter les éventuelles notes de version spécifiques décrites au niveau de chaque version : https://github.com/PnX-SI/GeoNature-atlas/releases
        - **ATTENTION** : Avant de lancer un script, l'éditer et corriger les chemins vers l'installation de GeoNature
        - Sur l'instance "*web-srv*" exécuter tous les scripts en lien avec l'interface.
    - Surtout, si les notes de version indiquent qu'il faut exécuter un script de migration de base de données synchroniser "*db-srv*" :
        - Synchroniser l'ensemble de "www" de "web-srv" vers "db-srv" : `rsync -av -e "ssh -p <port-ssh-db>" /home/geonat/www/ geonat@db-<region>-sinp:/home/geonat/www/`
- Sur "*db-srv*" :
    - Se connecter au serveur : `ssh geonat@<sinp-db>`
    - Exécuter les scripts de mise à jour de la base de données. Vous pouvez utiliser une commande du type : `psql -h localhost -U geonatadmin -d gnatlas -f ~/www/atlas/data/<mon-fichier-de-mise-a-jour>.sql`
        - **Note** : la requête doit être exécuter avec l'utilisateur *geonatadmin* et non *geonatatlas* car les vues matérialisées de la base *gnatlas* appartiennent à *geonatadmin*. Il aurait peut être été mieux d'installer la base de l'atlas avec *geonatatlas*...
- Sur "*web-srv*" :
    - Relancez l'installation automatique de l'application : '' cd ~/www/atlas/ ; ./install_app.sh ''
    - Consulter les logs de l'installation dans `~/www/atlas/log/errors_atlas.log` avec : `tail -f ~/www/atlas/log/errors_atlas.log '' * Réactiver le service de *Supervisor* : ''supervisorctl start atlas`
    - Désactiver le mode maintenance de l'Atlas : `sudo nginx_dissite atlas_maintenance.conf ; sudo nginx_ensite atlas.conf ; sudo service nginx reload`

À partir d'une branche

Toutes étapes suivantes se déroulent sur "*web-srv*" à l'exception des mises à jour de la base à exécuter via `psql` sur "*db-srv*" :

- Exporter le nom de la branche dans une variable d'env locale à la session. C'est le nom utilisé pour les fichiers il faut donc y remplacer les éventuels "/" par "-" : `export GNAB="<branche>"`
- Indiquer la version de l'Atlas sur laquelle se base les développements
    - Pour voir la version, en local dans le dossier du code de l'Atlas contenant les développements : `cat VERSION`
    - Récupérer aussi les 7 derniers caractères du SHA1 du dernier commit de la branche à récupérer

    - Exporter la version de l'Atlas obtenu avec *cat* en la concaténant avec les 7 premiers caractères du commit séparé par "-" . Ex. : `export GNAV="1.5.2-dev0-04dbbcb"`
- Vérifier les versions en question : '' echo "Branche: ${GNAB} ; TaxHub : ${GNAV}" ''
- Se placer dans le dossier *dwl* de l'utilisateur *geonat* : `cd ~/dwl/`
- Télécharger l'archive : `wget https://github.com/PnX-SI/GeoNature-atlas/archive/<branche>.zip -O "atlas_${GNAB}_v${GNAV}.zip"`
    - Il est aussi possible d'indiquer un SHA1 de commit pour l'URL : `https://github.com/PnX-SI/GeoNature-atlas/archive/<sha1>.zip`
- Décompresser l'archive dans le dossier *dwl* de l'utilisateur *geonat* : `unzip atlas_${GNAB}_v${GNAV}.zip -d ~/dwl/`
- Renommer le dossier au format GeoNature (⇒ uniformité), dans le nom de la branche les "/" sont remplacés par des "-" : `mv GeoNature-atlas-${GNAB} atlas_${GNAB}_v${GNAV}`
- Déplacer le dossier au format GeoNature dans *www/* : '' mv atlas_${GNAB}_v${GNAV} ~/www/ ''
- Se rendre dans le dossier *www* : `cd ~/www/`
- Copier les fichiers de config de l'ancien dossier vers le nouveau :
    - le fichier de paramètres pour les scripts d'installation : `cp ~/www/atlas/atlas/configuration/settings.ini ~/www/atlas_${GNAB}_v${GNAV}/atlas/configuration/settings.ini`
    - le fichier de config du backend Python : `cp ~/www/atlas/atlas/configuration/config.py ~/www/atlas_${GNAB}_v${GNAV}/atlas/configuration/config.py`
- Vérifier l'existence de nouveaux paramètres de config :
    - Dans *config.py* : '' diff ~/www/atlas/atlas/configuration/config.py.sample ~/www/atlas_${GNAB}_v${GNAV}/atlas/configuration/config.py.sample ''
    - Dans *settings.ini* : '' diff ~/www/atlas/atlas/configuration/settings.ini.sample ~/www/atlas_${GNAB}_v${GNAV}/atlas/configuration/settings.ini.sample ''
- Compléter/modifier les paramètres de config en fonction des besoins de la branches dans les fichiers :
    - *config.py* : '' vi ~/www/atlas_${GNAB}_v${GNAV}/atlas/configuration/config.py ''
    - *settings.ini* : '' vi ~/www/atlas_${GNAB}_v${GNAV}/atlas/configuration/settings.ini ''
- Copier le contenu du dossier "*static*" : '' cp -aR ~/www/atlas/static/custom/ ~/www/atlas_${GNAB}_v${GNAV}/static ''
- Modifier les liens symboliques :
    - pour la nouvelle version : `cd ~/www; rm -f atlas; ln -s atlas_${GNAB}_v${GNAV} atlas`
    - pour l'ancienne version :
        - `cd ~/www; rm -f atlas_old; ln -s atlas_v${GNAV} atlas_old`
        - ou si l'ancienne version était déjà sur une branche de test : `cd ~/www; rm -f atlas_old; ln -s atlas_${GNAB}_v<ancienne-version> atlas_old`
- Synchroniser l'ensemble de "www" de "web-srv" vers "db-srv" : `rsync -av -e "ssh -p <port-ssh-db>" /home/geonat/www/ geonat@db-<region>-sinp:/home/geonat/www/`
- Consulter les éventuelles notes de version spécifiques décrites au niveau de chaque version : https://github.com/PnX-SI/GeoNature-atlas/releases
    - Sur l'instance "*web-srv*" exécuter tous les scripts en lien avec l'interface.
    - Sur l'instance "*db-srv*" exécuter tous les scripts en lien avec la base de données.
- Sur "*db-srv*" :
    - Se connecter au serveur : `ssh geonat@<sinp-db>`

    - Exécuter les scripts de mise à jour de la base de données. Vous pouvez utiliser une commande du type : `psql -h localhost -U geonatadmin -d gnatlas -f ~/www/atlas/data/<mon-fichier-de-mise-a-jour>.sql`
        - **Note** : la requête doit être exécuter avec l'utilisateur *geonatadmin* et non *geonatatlas* car les vues matérialisées de la base *gnatlas* appartiennent à *geonatadmin*.
- Sur "*web-srv*" :
    - Relancez l'installation automatique de l'application : '' cd ~/www/atlas/ ; ./install_app.sh ''
    - Consulter les logs de l'installation dans `~/www/atlas/log/errors_atlas.log` avec : `tail -f ~/www/atlas/log/errors_atlas.log '' * Réactiver le service de *Systemd* : ''sudo systemctl start atlas`
    - Désactiver le mode maintenance de l'Atlas : `sudo nginx_dissite atlas_maintenance.conf ; sudo nginx_ensite atlas.conf ; sudo nginx-reload`

Import des images de l'INPN

Si vous souhaitez utiliser dans l'Atlas les images issues de l'INPN vous pouvez suivre la procédure suivante :

- Se connecter sur le serveur : `ssh geonat@sinp-<region>-web`
- Lancer une session avec Screen car l'exécution du script peut prendre du temps : '' screen -S "images-inpn-import" ''
    - Voir la documentation générale concernant les commandes à utiliser avec Screen pour quitter puis se reconnecter à une session.
- Se placer dans le dossier du script dans TaxHub : `cd ~/www/taxhub/data/scripts/import_inpn_media '' * Créer un environnement virtuel : '' python3 -m venv venv '' * Activer l'environnement virtuel : '' source venv/bin/activate '' * Redonner les droits d'exécution à GCC pour tout le monde si l'on veut pouvoir installer correctement les paquets Python dans le venv : '' sudo chmod o+x /usr/bin/gcc '' * Installer les paquets suivant : '' pip install psycopg2 requests '' * Retirer les droits d'exécution à GCC pour tout le monde : '' sudo chmod o-x /usr/bin/gcc '' * Créer le fichier de configuration : '' cp config.py.sample config.py '' * Modifier les paramètres : * '' SQLALCHEMY_DATABASE_URI = "postgresql://geonatadmin:<mot-de-passe>@10.0.1.20:5432/geonature2db"`
    - Pour l'intégration intiale :

```
 QUERY_SELECT_CDREF = """SELECT DISTINCT cd_ref FROM taxonomie.bib_noms ORDER BY cd_ref LIMIT 100"""
```

        - Supprimer le '' LIMIT 100 '' une fois un premier test effectué
    - Pour les mise à jour :

```
QUERY_SELECT_CDREF = """
SELECT DISTINCT cd_ref
FROM taxonomie.bib_noms AS bn
WHERE NOT EXISTS (
    SELECT 'X'
```

```
    FROM taxonomie.t_medias tm
    WHERE tm.cd_ref = bn.cd_ref
)
ORDER BY cd_ref
"""
```

- Si la table `taxonomie.bib_noms` est vide. Cela peut arrivé si aucun module de saisie n'est actif (cas du SINP). Il est possible de la remplir à partir des données de `gn_synthese.synthese` avec la requête suivante :

```
INSERT INTO taxonomie.bib_noms (cd_nom, cd_ref)
SELECT DISTINCT s.cd_nom, t.cd_ref
FROM gn_synthese.synthese AS s
  JOIN taxonomie.taxref AS t
    ON s.cd_nom = t.cd_nom
WHERE NOT s.cd_nom IN (SELECT DISTINCT cd_nom FROM
taxonomie.bib_noms);
```

- Utiliser *Psql* avec la requête suivante si `bib_noms` est vide (Temps d'exécution 20s pour 9,3 millions de lignes dans `synthese`): '' psql -h localhost -U geonatadmin -d geonature2db -c "INSERT INTO taxonomie.bib_noms (cd_nom, cd_ref) SELECT DISTINCT s.cd_nom, t.cd_ref FROM gn_synthese.synthese AS s JOIN taxonomie.taxref AS t ON s.cd_nom = t.cd_nom ;" ''

- Lancer le script : '' python import_inpn_media.py ''
- Vérifier la présence des médias dans la table `taxonomie.t_medias`.
- Si tout c'est bien passé, désactiver l'environnement virtuel : '' deactivate ''
- Le script ajoute les photos en tant que "secondaire" pour désigner celle avec le plus petit id_media (=~ au hasard) comme "principale", utiliser la requête :
  - lors de l'intégration initiale :

```
WITH first_media AS (
    SELECT MIN(id_media) AS first_id_media_founded, cd_ref
    FROM taxonomie.t_medias
    GROUP BY cd_ref
)
UPDATE taxonomie.t_medias AS tm
    SET id_type = 1
    FROM first_media AS fm
    WHERE tm.id_media = fm.first_id_media_founded
        AND tm.cd_ref = fm.cd_ref ;
```

  - lors des mises à jours :

```
WITH first_media AS (
    SELECT MIN(id_media) AS first_id_media_founded, cd_ref
    FROM taxonomie.t_medias
    WHERE cd_ref NOT IN (
    SELECT cd_ref
    FROM taxonomie.t_medias stm
    WHERE id_type = 1
    )
```

```
    GROUP BY cd_ref
)
UPDATE taxonomie.t_medias AS tm
    SET id_type = 1
    FROM first_media AS fm
    WHERE tm.id_media = fm.first_id_media_founded
        AND tm.cd_ref = fm.cd_ref ;
```

    - Soit : '' psql -h localhost -U geonatadmin -d geonature2db -c "WITH first_media AS (SELECT MIN(id_media) AS first_id_media_founded, cd_ref FROM taxonomie.t_medias GROUP BY cd_ref) UPDATE taxonomie.t_medias AS tm SET id_type = 1 FROM first_media AS fm WHERE tm.id_media = fm.first_id_media_founded AND tm.cd_ref = fm.cd_ref ;" ''
- Pour afficher les images sur l'Atlas, il est nécessaire de rafraichir les données des vues matérialisées *atlas.vm_medias* et *atlas.vm_taxons_plus_observes* :

```
REFRESH MATERIALIZED VIEW atlas.vm_medias WITH DATA ;
REFRESH MATERIALIZED VIEW atlas.vm_taxons_plus_observes WITH DATA ;
REFRESH MATERIALIZED VIEW atlas.vm_stats WITH DATA ;
TRUNCATE atlas.t_cache ;
```

Mise à jour des données

- **Ressources** : https://github.com/PnX-SI/GeoNature-atlas/blob/master/docs/vues_materialisees_maj.rst
- Se connecter au serveur : `ssh geonat@<sinp-db>`
- Lancer une nouvelle session avec *Screen* : `screen -S "update-atlas"`
    - Voir la documentation générale concernant les commandes à utiliser avec Screen pour quitter puis se reconnecter à une session.
- **NOTES** : si le *refresh* est réalisé `CONCURRENTLY`, il est possible de lancer le rafraîchissement des VMs sur l'installation de production. L'Atlas restera fonctionnel même pendant le rafraîchissement.
- Les actions disponibles :
    - Pour mettre à jour uniquement les **observations** (table *synthese*) utiliser la requête : ''psql -h localhost -U geonatadmin -d gnatlas -c "SELECT atlas.refresh_materialized_view_data() ;" ''
    - Pour mettre à jour uniquement les **zones géographiques** (à faire uniquement si le *territoire* a été modifié) : '' psql -h localhost -U geonatadmin -d gnatlas -c "SELECT atlas.refresh_materialized_view_ref_geo() ;" ''
    - Pour tout mettre à jour sans distinction (la mise à jour des données géographiques peut être longue) : '' psql -h localhost -U geonatadmin -d gnatlas -c "SELECT RefreshAllMaterializedViews('atlas') ;" ''
- Il est possible d'automatiser cette tâche via un *cron* ⇒ Dans notre cas, ce n'est pas utile...

From:
https://sinp-wiki.cbn-alpin.fr/ - **CBNA SINP**

Permanent link:
https://sinp-wiki.cbn-alpin.fr/serveurs/installation/web-srv/geonature-atlas-mise-a-jour?rev=1648241904

Last update: **2022/03/25 20:58**