Synthese - Tests d'amélioration des performances

Objectif

Trouver des solutions pour supprimer la limite des 100 000 données affichables et exportables.

Principe général

Afin d'améliorer les performances du module Synthèse de GeoNature, nous pouvons remplacer le web service geojson actuel servant à la fois au rendu sur la carte et à l'affichage des données en liste par plusieurs web services spécialisés. Ces web services devront retourner des données relativement constantes quelque soit le nombre d'observation à afficher résultant de la recherche effectuée dans la Synthese.

Nous pouvons distinguer 3 principaux type de web services dans le module Synthese qui fournissent des données à :

- le tableau d'informations paginée
- l'export des données
- la carte

Nous proposerons des solutions pour chacun d'entre eux.

Solutions pour le tableau d'information paginée

Dans le cas des observations présentées sous forme de tableau paginées, la solution la plus adéquates consiste à **créer un web service paginée côté serveur**. L'avantage de ce type de web service c'est que le nombre d'informations retournées est constant quelque soit les résultats de la recherche. Son utilisation dans certains modules de GeoNature a montré son efficacité et sa facilité d'utilisation avec le composant Datatable d'Angular Material.

Côté serveur, il sera nécessaire de s'assurer que les requêtes exécuter en base de données soient toujours les plus performantes possibles. Si nous souhaitons garder un tri similaire des observations, il faudra rassembler les observations sélectionnée via la carte en début de liste.

Côté navigateur client, ce mécanisme permet de s'assurer que les performances seront toujours les mêmes. Il faudra toutefois peut être retravailler le mécanisme d'interaction entre les observations de la carte et du tableau. Il semble toutefois possible de s'appuyer sur la valeur du champ `id_synthese` pour le maintenir.

Solutions pour l'export des données

Le travail effectué en 2023 sur le module Export a démontré qu'il était possible d'exporter un grand volume de données sans surcharger la mémoire et l'espace disque du serveur. Mais pour cela, il est nécessaire de générer l'export en arrière plan et alerter l'utilisateur lorsque l'export est prêt à être téléchargé.

Pour cela, les nouveaux mécanismes tels que l'utilisation de tâches Celery et les notifications ajoutés dans les dernières version de GeoNature nous permettrons de mettre en place un export performant avec des limites très largement augmentées.

Toutefois, la mise en place de cette solution nécessitera de revenir sur la branche principale de GeoNature dans laquelle les tâches Celery et les notifications ont été implémentées.

Solutions pour la carte

L'amélioration des performances d'affichage sur la carte et le point qui nécessite le plus d'expérimentation car aucune solution satisfaisante n'a encore été expérimenté dans GeoNature. Nous réaliserons donc la majorité des nos tests sur ce sujet particulier.

Toutefois, le web service GeoJson actuel a vu ces performances largement améliorées lors de travaux effectués en 2022 avec le regroupement des observations par géométries identiques. Une optimisation de la création du GeoJson a également été effectuée lors des travaux visant à permettre le regroupement et l'affichage des observations par mailles sur la carte de la Synthese.

Afin de poursuivre l'amélioration de l'affichage d'une grand nombre d'observations sur la carte, nous envisageons donc d'expérimenter les éléments suivant :

1. Tester la différence entre l'utilisation de la geom avec un SRID 2154 et 4326 pour les intersections
2. Comparer la rapidité des intersections de géométrie à l'aide de : `st_intersects()` ou de &&
3. Comparer différents types d'index (GIST, BRIN, SP-GIST) sur le champ geométrie de la table Synthese
4. Comparer l'utilisation d'une intersection spatiale vis à vis de l'utilisation d'une table relationnel (`cor_area_synthese`)
5. Utilisation de tuiles vecteurs ou geojson permettant de paralléliser les requêtes en base de données
6. Web service spécifique carto geojson/mvt filtrer sur la bbox de la carte actuellement visualisé
7. Utiliser un affichage différent en fonction du zoom : petit zoom avec mailles, moyen avec cluster/polygones, grand avec points précis.
8. Tenter d'améliorer les performances de `cor_area_synthese` :
    1. Ajouter une colonne `area_type_code` contenant le code du type de zone géo correspondant à l'`id_area`.
    2. Mettre en place un partitionnement de la table basé sur `area_type_code`
    3. Créer une vue matérialisée mettant en cache les données agrégées par maille pour l'affichage par défaut (nombre d'observation par maille sans filtre)
9. Tester l'utilisation d'une table des géométries de `l_areas` subdivisées (`st_subdivide`) pour essayer l'agrégation par communes.
10. Tester l'utilisation d'une vue matérialisée correctement indexé et agrégeant l'ensemble des informations nécessaires aux requêtes de la Synthese

Tests d'amélioration de l'affichage sur la carte

Principes d'amélioration

Pour l'amélioration de l'affichage cartographique, nous envisageons de :

- paralléliser l'accès aux données
- diminuer le volume de données restitué par chaque appel de web service
- agréger les données à petite échelle

Afin de paralléliser l'accès aux données, il est nécessaire de remplacer le "mono" webservice REST GeoJson actuel par un web service de type Tile Map Service (TMS) fournissant des données vectorisés sous forme de tuiles.

Le format de web service TMS, dont le standard est décrit par osgeo.org dans le doc Tile Map Service Specification, est utilisé par Google Map, mais également par OpenSteetMap dans une version simplifiée.
Ce type de web service doit pouvoir être appelé plusieurs fois en parallèle si l'on souhaite accélérer le rendu sur une carte web. Pour cela l'utilisation d'URLs HTTPS supportant HTTP2, avec des sous-domaines différents (a., b., c., etc) doit permettre aux navigateurs de réaliser un maximum de requêtes simultanées.
La possibilité de générer parallèlement plusieurs tuiles vecteurs, de dimension réduite et dont le contenu varie en fonction du niveau de zoom devrait accélérer le rendu.

L'utilisation de tuiles vecteurs permet de maintenir l'interaction avec les objets retournés comme c'est le cas actuellement avec le GeoJson. Il existe deux format de tuiles vecteurs envisageable : Mapbox Vector Tile (MVT) ou GeoJson. Mais la quasi totalité des exemples et articles sur le web concernent les tuiles MVT. Les tuiles geojson sont-elles moins intéressantes ?

Pour les petits niveau de zoom, nous utiliserons une technique d'agrégation permettant d'afficher une grande quantité d'information. Nous essaierons de retourner des données agrégées par maille avec une coloration des mailles en fonction du nombre d'observation contenues. Cela permettra de garantir la lisibilité des données affichés.

L'utilisation de maille est un facteur important car c'est un objet géographique simple qui ne contient que 5 points. Nous disposons également de mailles de différentes tailles 1, 5 et 10 km par défaut dans GeoNature, ce qui permet de sélectionner la taille la mieux adaptée à son territoire et à la quantité de données hébergée. Enfin, l'intersection de données avec des mailles en base de données peut se faire à l'aide de la fonction Postgis `ST_Intersects()` mais également à l'aide de l'opérateur bien plus performant && qui pour les géométries signifie "l'étendue recouvre ou touche".

Enfin, afin de réduire le volume de données à renvoyer par tuile, il est possible de simplifier la géométrie des objets renvoyés. Dans le cadre du module Synthèse de GeoNature, cette possibilité est envisageable mais n'aura pas forcément beaucoup d'intérêt. En effet, les mailles agrégeant les données à petite échelle ne peuvent pas être plus simplifiées. Cette technique peut éventuellement être utilisé pour simplifier les géométries d'observation de type polygone renvoyées à grande échelle. Mais dans ce cas là, nous cherchons souvent à garder le maximum de précision.
Par contre, il peut être intéressant de simplifier le nombre de décimale des coordonnées des géométries des observation. Avec 5 chiffres après la virgule, une coordonnées est précise au mètre. Il semble donc intéressant de garder 5 (ou 6) chiffres maximum après la virgule.

La parallélisation de l'accès aux données, leur vectorisation et leur agrégation par mailles devrait rendre possible l'affichage d'une grande quantité de données à petite échelle.

Tuiles GeoJson

Il serait intéressant de maintenir le format GeoJson en proposant des tuiles vecteurs au format "geojson". Les principaux avantages sont :

- bon support du GeoJson par Leaflet
- existence d'un plugin Leaflet Tilelayer GeoJson
- le module Synthese de GeoNature exploite déjà ce format

Cependant, les défauts des tuiles GeoJson comparés aux tuiles MVT sont :

- le poids des tuiles GeoJson (en texte) générées comparé à celui des tuiles PBF qui utilisent un format binaire est plus important.
- la quasi absence de ressource sur le web concernant la création de tuiles GeoJson.
- le support des tuiles GeoJson n'est pas offert nativement par les principaux framework carto web : Maplibre (pas de support), LeafLet (via le plugin Leaflet Tilelayer GeoJson), OpenLayers (?).

Au niveau pratique, concernant la création de tuile GeoJson, Postgis semble offrir la possibilité de découper des géométries en fonction d'un polygone donnée ("clip") à l'aide de la fonction `ST_Intersection()`. Il semble également possible d'utiliser l'utilitaire ogr2ogr qui permet de générer du GeoJson en redécoupant les géométrie données en fonction du contour d'une bbox.

Exemples de code pour créer des tuiles GeoJson :

- Dirt-simple-postgis-http-api - GeoJson ⇒ en réalité, cela ne créé pas un geojson dont les géométrie sont restreinte à la tuile mais renvoie toutes les géométrie intersectant la bbox de la tuile demandée…

Tuiles Mapbox Vector Tile

Le principal format de tuiles vecteurs est le Mapbox Vector Tiles (MVT). Les URLs permettant de récupérer ces tuiles se terminent souvent par l'extension ".mvt" mais l'extension ".pbf" est également utilisée car comme expliqué dans le guide du standard MVT, les tuiles MVT sont encodées en s'appuyant sur Google Protobufs (PBF).

Par contre, le format de fichier PBF d'OpenStreetMap utilise également Google Protobufs mais n'a rien à voir au niveau de son implémentation avec le format utilisé pour les tuiles MVT.

Concernant la création de tuile MVT, Postgis supporte très bien ce format à l'aide des fonctions `ST_TileEnvelope()`, `ST_AsMVTGeom()` et `ST_AsMVT()`.

Simplification des géométries à l'aide de Postgis

Comme indiqué précédemment, il peut être utile de simplifier les géométries retournée en réduisant le nombre de chiffre après la virgule des coordonnées en utilisant la fonction Postgis `ST_SnapToGrid()`. Pour garder une précision métrique, 5 chiffres après la virgule suffisent : `ST_SnapToGrid(geom, 0.00001)`.

S'il s'avèrent nécessaire de simplifier des géométries complexes (Communes, Départements, zones de protection...), il faut utiliser des fonctions qui préservent la topologie des géométries pour éviter de créer des géométries invalides. C'est le cas de la fonctions Posgis `ST_SimplyPreserveTopology`.

Le web service retournant ces géométrie doit pouvoir activer et augmenter la simplification des géométrie en fonction du niveau zoom. L'utilisation de la formule suivante dans les requêtes est très intéressante car cela permet de simplifier automatiquement les géométries en fonction du niveau de zoome de façon que les écarts apparaissant entre 2 géométries contiguës ne soient pas visibles : `ST_SimplyPreserveTopology(geom, 0.7 / (2 ^ <zoom-level>))`.

Framework carto web et tuiles vecteurs

Actuellement, nous utilisons **Leaflet** comme framework carto web. Il est simple, peu verbeux et fournie jusqu'à présent toutes les fonctionnalités carto dont GeoNature à besoin. Malheureusement, il ne supporte pas nativement les tuiles vecteurs. Il est possible d'utiliser un plugin pour cela mais il en existe de nombreux qui ne sont pas tous bien maintenu. Nous listerons les plugins Leaflet existant permettant l'utilisation de tuiles vecteurs, détermineront s'ils sont toujours actifs et évaluerons s'ils sont bien maintenus.

Enfin, nous testerons la solution dont le rendu des tuiles vecteur est le plus performant dans un navigateur web. Même si cela impliquerait une modification importante du code de GeoNature, les possibilités offertes par ces récents frameworks supportant de nouvelle techniques d'affichage web pourraient s'avérer valoir l'investissement. Nous testerons l'utilisation de tuiles vecteur avec le plus intéressant et évaluerons sa facilité d'intégration avec Angular.

Besoins

Dans le cas d'un plugin pour Leaflet, les besoins sont :

- supporter l'affichage de tuiles "vecteur" au format Mapbox Vector Tiles (MVT)
- être activement maintenu
- être le plus performant possible pour le rendu de tuiles vecteur dans un navigateur web

Dans le cas d'un framework carto différent de Leaflet, en plus des besoins listés précédemment, il doit :

- supporter l'affichage de tuiles "raster" au format TMS fournie par OpenStreetMap
- supporter l'affichage de données issues de web service WMS
- supporter l'affichage de GeoJson
- permettre l'édition en ligne (point, polygone) directement ou via un plugin

Plugins Leaflet ajoutant le support des tuiles vecteur

Leaflet consacre une catégorie pour les plugins apportant le support des tuiles vecteurs. Les plugins non libres ou s'appuyant sur une API propriétaire ne seront pas pris en compte. Liste des plugins apportant le support des tuiles MVT ou GeoJson à Leaflet classé dans l'ordre décroissant d'activité/maintenance :

Nom	Dépôt	Formats supportés	Dernière version	Date dernière release	Date dernier commit	Licence
Maplibre GL Leaflet	maplibre/maplibre-gl-leaflet	MVT	v0.0.20	19 septembre 2023	6 juin 2024	ISC
Leaflet.VectorTileLayer	jkuebart/Leaflet.VectorTileLayer	MVT	v0.16.0	10 octobre 2023	20 avril 2024	BSD-3-Clause license
Protomaps Leaflet	protomaps/protomaps-leaflet	MVT, PMTiles	-	-	9 mai 2024	BSD-3-Clause license
Vector Grid	Leaflet/Leaflet.VectorGrid	MVT	v1.3.0	28 août 2017	1er septembre 2021	THE BEER-WARE LICENSE
Leaflet Tilelayer GeoJson	glenrobertson/leaflet-tilelayer-geojson	GeoJson	v1.0.2	11 octobre 2016	21 octobre 2016	THE BEER-WARE LICENSE
Hoverboard	summer4096/hoverboard	MVT, GeoJson	v1.1.3	27 mars 2015	13 mai 2015	?

L'utilisation de Leafleft avec le plugin Maplibre-Gl-Leaflet est la solution qu'il faudrait retenir car la communauté autour de MapLibre (fork de Mapbox GL JS) semble être bien active. Ceci dit les plugins jkuebart/Leaflet.VectorTileLayer et protomaps/protomaps-leaflet semblent être des solutions envisageables.

Framework carto web avec support natif des tuiles vecteur

Les frameworks carto web suivant possèdent un support des tuiles vecteurs sans l'ajout d'un plugin sont :

1. MapLibre GL
2. OpenLayers

Conclusion

Nous testerons l'utilisation du framework MapLibre GL car son développement a été basé dès l'origine sur l'utilisation de tuiles vecteur. C'est celui qui est sensé nous offrir les meilleures performances lors de l'utilisation de tuiles vecteur.

Ce test sera réalisé en développant un module GeoNature spécifique "Syntests". Cela permettra d'évaluer la facilité d'intégration du framework à Angular. Nous pourrons aussi concentrer nos sur l'optimisation des performances du rendu des observations issues de la table `synthese` de GeoNature sans être contraint par l'implémentation actuelle du module "Synthese".

Tests de serveurs de tuiles vecteurs

Actuellement, il n'y a pas d'API fournissant de tuiles vecteurs de manière équivalente aux tuiles raster fournies par OpenStreetMap. Cela nécessite donc de **maintenir l'affichage du fond cartographique de base via des tuiles raster**.

Il existe deux grands types de serveurs de tuiles vecteurs :

- les serveurs de fonds cartographiques : une tuile contient un très grand nombre de données différentes afin de générer un fond cartographique.
- les serveurs de couches : une tuile contient uniquement les données de la couche demandée/configurée.

Serveurs de tuiles pour fonds cartographiques

Il existe des serveurs de tuiles vecteurs open source et/ou gratuit qui sont auto-hébergeables. Par exemple :

- Mbtileserver : en Go. Utilise des tuiles issues de fichiers `mbtiles`.
- Tileserver-GL : open source et gratuit. Utilise des tuiles issues de fichiers `mbtiles`.
- Maptiler Server : très simple à installer, à configurer et utiliser. Il est compatible Linux mais il nécessite une licence payante pour une utilisation en production.
- Voir la liste de serveurs de tuiles vecteurs maintenue par Mapbox

L'utilisation de ces serveurs de tuiles vecteurs nécessite de les héberger sur son propre serveur. Cela implique :

- l'installation, la configuration et la maintenance du serveur
- de disposer de suffisamment d'espace disque : de quelques dizaines à plusieurs centaines de Go suivant les niveaux de zoom supportés et la taille de la carte à rendre (pays, continent, planète).

Dans le cadre de GeoNature, l'utilisation de ce type de serveurs peut être intéressant si le framework carto web utilisé pour gérer les cartes supporte bien les tuiles vecteurs. Cela ouvre la voie a de nouveaux usages : vue 3D (intéressant pour mieux visualiser les pentes, vallée…), changement de langue des textes de la carte, changement de style instantané du fond carto en fonction des usages (nuit, nature, …). Dans le cas contraire, il vaut mieux privilégier l'utilisation de tuiles raster.

Pour les cartes de GeoNature, il pourrait être intéressant de **fournir des tuiles vecteurs pour le fond cartographique spécialement conçu pour les zones non urbanisés**. Ce style d'affichage est nommé "terrain" ou "outdoor". Il existe des fichiers de styles open source pour ce type d'affichage. Ce style nécessite l'utilisation de plusieurs types de tuiles :

- raster-dem : pour les ombres portés du relief
- vecteurs : pour les contours et les autres éléments

Pour les contours (lignes de niveaux) et le relief, il est possible d'utiliser les fichiers fournis par Makina Corpus et qui concernent la France Métropolitaine. Voir également l'article concernant la génération des ces tuiles.

Serveurs de tuiles pour couches de données spécifiques

Ces serveurs s'appuie sur une base de données Postgis existante pour générer les tuiles vecteurs. La configuration de ces serveurs prévoie de pouvoir associer une requête SQL à chaque couche de tuiles vecteurs que l'on souhaite générer.

Serveurs de couches intéressant :

- Tegola : en Go, open source et gratuit. Génère des tuiles depuis Postgresql ou des fichiers GPKG.
- Martin
- pg_tileserver

Dans le cadre de GeoNature, ce type de serveur peut être intéressant à installer si l'on souhaite générer des couches de tuiles vecteurs spécifiques à des données hébergées dans la base. Par exemple, les différents types de zones géo présentent dans la table `ref_geo.l_areas` pourraient être proposé sous forme de web services de tuiles vecteurs activable ou pas en fonction des besoins. Il est possible d'imaginer aussi des web services d'agrégation de données basés sur des vues : intensité de prospection, nombre d'observations, diversité spécifiques...

Par contre, les services de tuiles en question ne doivent pas être assujetti à l'utilisation de filtres dépendant du choix de l'utilisateur via l'interface. Il faut pouvoir fournir les tuiles de la même façon quelque soit la personne en faisant la demande...

Web service de tuiles vecteurs sur mesure

Dans notre cas, il semble **plus intéressant de construire des web services de tuiles vecteurs en s'appuyant sur les fonctionnalités offertes par Postgis**. Les données renvoyées dans les tuiles de la Synthese" seront dépendantes :

- de l'utilisateur en faisant la demande
- de nombreux filtres au choix de l'utilisateur

Posgis fournit des fonctions permettant de faciliter la création de tuile MVT :

- `ST_TileEnvelope(z, x, y)` (doc) : permet de crée un polygone rectangulaire (bbox) dans le SRID 3857 en fonction du niveau de zoom, du x et du y d'une URL d'un web service TMS.
- `ST_AsMVTGeom()` (doc) : transforme une géométrie dans l'espace de coordonnées d'une tuile MVT (Mapbox Vector Tile), en la coupant aux limites de la tuile si nécessaire.
- `UNION` et `ST_AsMVTGeom()` : permettent de stocker plusieurs couches (*layers*) de géométries "MVT" dans la même tuile.
- `ST_AsMVT()` (doc) : fonction d'agrégation qui renvoie une représentation binaire Mapbox Vector Tile d'un ensemble de lignes correspondant à une couche de tuiles.

Exemple de requêtes créant une tuile vecteur basée sur la table `ref_geo.l_areas` de GeoNature pour les valeurs zoom = 16, x = 33877 et y = 23672 :

```
WITH bounds AS (
    SELECT ST_TileEnvelope(16, 33877, 23672) AS envelope
```

```
), mvtgeom AS (
    SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), b.envelope) AS geom,
           t.id_area,
           t.area_code,
           t.area_name
    FROM ref_geo.l_areas AS t, bounds AS b
    WHERE ST_Intersects(t.geom, ST_Transform(b.envelope, 2154))
        AND t.id_type = ref_geo.get_id_area_type('COM')
)
SELECT ST_AsMVT(mvtgeom.*) FROM mvtgeom ;
```

Exemples de création de tuiles vecteurs en Python à l'aide de Postgis :

- Github :
  - pramsey/minimal-mvt
  - jbdesbas/vectipy
  - Oslandia/postile
  - Fast Vector - Python FastApi, Postgis
- Articles :
  - Vector tiles, Postgis et OpenLayers
  - Serving Mapbox Vector Tiles with Postgis, Nginx and Python backend
  - Restricted Vector Tile access with FastAPI & PostGIS
  - Composite MVT tiles with Postgis

Principe des tests de performance

Nous avons réalisé plusieurs tests à l'aide du module un module GeoNature spécifique "Syntests". Il nous a permis d'encapsuler rapidement l'exécution de requêtes SQL afin de comparer leur rendu sous forme de tuiles vecteurs affichées à l'aide du framework carto web Maplibre GL.

Nous avons également utilisé les données de la base du SINP AURA qui comprend 23,5 millions d'observations dans la table `synthese` et 378 millions de lignes dans la table `cor_area_synthese`. Nous avons pu ainsi tester l'efficacité des tuiles vecteurs sur une base de données comprenant un nombre conséquent d'observations.

Comparaison SRID 2154 et 4326

Les résultats de la comparaison du nombre de données récupérés et des temps d'obtention entre l'utilisation du SRID 4326 et 2154 ne montre pas de différences majeures. Mais l'utilisation du SRID 4326 semble légèrement plus rapide et en outre demande moins de traitement pour son utilisation avec le format GeoJson.

Conclusion : **privilégier le SRID 4326** pour les champs de type géométrie.

Comparaison opérateur && et st_intersects avec index GIST

Le test de comparaison des opérateurs ''&&'' et de la fonction ''ST_Intersects'' avec un index GIST montre que l'opérateur && est plus rapide. Cependant il ne peut être utilisé qu'avec des géométries

de type "bounding box" (des rectangles).

Conclusion : **privilégier l'opérateur &&** quand c'est possible sinon utiliser `st_intersects`.

Comparaison opérateur && et st_intersects avec index BRIN

Le test d'utilisation d'un index de type BRIN sans trier les lignes sur la colonne géométrique utilisée montre des temps d'exécution plus long qu'avec l'index GIST. Ces temps sont plus longs pour l'opérateur && comme pour la fonction `ST_Intersects`.

Conclusion : **utiliser un index GIST** sur le champ contenant les géométries des observations de la Synthese.

Comparaison index GIST et SP-GIST

En suivant les informations fournies par ce document " Dalibo - Indexation avancée", nous avons mis en place un index de type SP-GIST sur le champ géométrie de la Synthese. Les résultats de la requête utilisée montre un léger avantage pour l'index de type SP-GIST.

Conclusion : **utiliser un index SP-GIST** pour améliorer les performances sur la colonne géométrie de la Synthese.

Comparaison agrégation via table relation et via intersection

La comparaison de l'agrégation des observations via l'utilisation de la table `cor_area_synthese` ou via l'opération spatiale && et des index SP-GIST mais bien en évidence qu'il est plus performant d'utiliser une table relationnelle pour pré-stocker les intersections.

L'agrégation par maille de 10km à partir du zoom 8 met 6 secondes pour la table `cor_area_synthese` et 52 secondes avec l'intersection spatiale ! Et cela malgré les 378 millions de lignes de la table `cor_area_synthese`.

Il faut donc maintenir l'utilisation de la table `cor_area_synthese` et chercher des solutions visant à optimiser ce mécanisme :

- Ajout d'une colonne indiquant le code du type de zone geo dans la table cor_area_synthese : `area_type_code`
    - Partitionnement de la table basé sur le type de zone geo
- Création d'une table de relation spécialisée pour les mailles M10 : `id_synthese`, `id_area`
- Mise en cache du nombre d'observation par maille M10 : `id_area`, `observation_nbr`

Conclusion : **pré-calculer et stocker dans une table les intersections de géométries**.

Test de l'ajout d'un champ area_type_code à la table cor_area_synthese

La table `cor_area_synthese` comprenant un très grand nombre de lignes (378 millions dans notre cas), il pourrait s'avérer plus performant de permettre à Postgresql d'éliminer un grand nombre d'entre elles en spécifiant le type de la géométrie liée. Pour cela, nous proposons d'ajouter un champ ''area_type_code''. Nous avons également tester l'utilisation de différents index sur ce champ.

Conclusion : **l'utilisation du champ `area_type_code` améliore les performances lorsqu'il est utilisé avec un index `'btree(area_type_code, id_area)`**.

Test vue matérialisée pour relations entre observations ''synthese'' et mailles M10

La création d'une vue matérialisée spécialisée pour stocker les relations entre observations ''synthese'' et mailles M10 a permis de tester son utilisation. Nous avons constaté des résultats très proches de ceux obtenus avec le simple ajout du champ `area_type_code` sur la table `cor_area_synthese`.

Conclusion : **continuer d'utiliser `cor_area_synthese`** plutôt que des vues matérialisées par type de zones géoégraphiques.

Test de vues matérialisées pour mettre en cache le nombre d'observations ''synthese'' par mailles

La création d'une première vue matérialisée `m10_observation_nbr` a permis de en cache le nombre d'observations `synthese` par maille M10. La création d'une seconde vue matérialisée `observation_nbr` a permis de mettre en cache le nombre d'observations `synthese` pour les différents types de mailles (M10, M5 et M1).

Cela nous a permis de tester l'utilisation d'une vue contenant un nombre réduit de données, uniquement pour les mailles 10km (table `m10_observation_nbr`), vis à vis de la vue contenant la mise en cache du nombre d'observation pour l'ensemble des mailles (table `observation_nbr`).

Nous constatons bien que les requêtes effectuées sur a vue contenant la mise en cache de calcul seulement pour les mailles M10 est plus rapide que celle contenant l'ensemble des mailles. Mais dans les 2 cas et pour l'ensemble des requêtes, nous obtenons de très bon temps de réponse (moins de 0,1s). Même si l'utilisation d'une vue matérialisée spécifique a un type de maille est plus rapide, il semble plus intéressant d'utiliser la vue matérialise prenant en compte l'ensemble des mailles car le rafraîchissement des données d'une seule vue matérialisée est plus simple à gérer.

Par ailleurs, la mise en cache de calcul sur de très grande quantité de données et donc une solution très satisfaisante pour générer des tuiles vecteurs. Leur affichage dans l'interface est très fluide. Malheureusement, cette méthode peut être difficilement appliquée sur le module Synthese qui utilise de nombreux filtres spécifiques et dont les résultats sont également dépendant des droits de l'utilisateur.

Conclusion :

- **mettre les calculs (ex. `COUNT`) en "cache" dès que possible**
- trouver le juste milieu entre performance et facilité de maintenance

Test de la création d'une table "cor_area_synthese" partitionnée

Ressources :

- PostgreSQL : évolution du partitionnement de 9.6 à 12 (1/2) : Capdata team, 14 novembre 2019.
- PostgreSQL : évolution du partitionnement de 9.6 à 12 (2/2) : Capdata team, 22 novembre 2019.

Nous avons créée une table partitionnée ''cor_area_synthese''. Cela nous a permis d'obtenir des informations sur sa composition, effectuer des tests dessus et mieux appréhender le principe du partitionnement.

Les résultats sur la mise en cache des données pour l'ensemble des mailles vis à vis d'un seul type (M10) a montré que la requête s'exécutant sur un seul type était pls rapide. Le partitionnement subdivisant la table initiale en une multitude de table par type, nous pouvons penser que cela pourra améliorer les résultats.

Notes :

- Il n'est pas possible de réutiliser une table standard pour la partitionnée, il faut recréer une nouvelle table partitionnée dés sa création.
- Il est nécessaire d'ajouter une colonne `area_type_code` pour l'utiliser comme origine du partitionnement.
- Toutes les colonnes présentes dans les partitions doivent faire partie de la clé primaire.
- Il peut être intéressant d'activer le partitionnement d'une table lorsque la taille de celle-ci dépasse la quantité de mémoire vive disponible...

Conclusions :

- l'utilisation du partitionnement **ne semble pas plus performant que l'ajout d'un champ `area_type_code`** sur la table `cor_area_synthese`. Par contre, il **facilite la gestion des données de la table `cor_area_synthese`**. Avec l'augmentation du nombre d'observations dans la `synthese`, le partitionnement pourrait s'avérer plus performant à terme.
- Il est **envisageable de gérer le partitionnement de la table `cor_area_synthese` sans toucher au code de GeoNature** à partir du moment où la table `cor_area_synthese` possède un champ `area_type_code`.

Test de la fonction st_subdivide()

Dans le cadre de la création d'une table `cor_area_synthese` partitionnée, nous avons utilisé la fonction Postgis st_subdivide() afin de créer une table contenant une ensemble de géométries simplifiées pour chaque géométrie complexe (toutes sauf mailles) présentent dans la table `l_areas`.

Les intersections effectuées à l'aide de la fonction st_intersects() se sont avérées largement plus rapide lorsqu'elles sont exécutées sur les géométries simplifiées de cette table et cela même si elles en contient un plus grand nombre. Voir cette article de Paul Ramsey sur le sujet.

Conclusion : **utiliser st_intersects() sur des géométries complexes simplifiées via**

st_subdivide()

Test utilisation table "cor_area_synthese" sans lien vers la "synthese"

Nous avons voulu tester une requête s'exécutant seulement sur une seule table pour évaluer l'impact des jointures. La requête effectuée seulement sur la table `cor_area_synthese` sans relation vers la table `synthese` divise le temps d'exécution par 5 !

Par ailleurs, nous avons confirmé dans le cadre de l'utilisation d'une autre base de données (Simethis) que des requêtes effectuées sur une seule table contenant des millions d'observations sans aucune jointure permet de générer un GeoJson agrégeant plusieurs millions d'observations (~8 millions) sous forme de mailles sur une surface correspondant à une région pour temps d'exécution de 30 secondes !

Conclusion : **réaliser des requêtes sur une seule table bien indexée** !

Test requêtes Synthese sur VM indexée

Le test précédent a montré une très grande efficacité et nous avons pu la confirmé dans le cadre de l'utilisation d'une autre base de données (Simethis). Les requêtes effectuées sur une seule table contenant des millions d'observations sans aucune jointure permet de générer un GeoJson agrégeant plusieurs millions d'observations (~8 millions) sous forme de mailles sur une surface correspondant à une région pour temps d'exécution de 30 secondes !

Nous avons commencer à tester cela en générant une vue matérialisé v_synthese_for_web_app bien indexée. Ensuite, nous avons commencé à modifié le fichier query_select_sqla.py dans lequel la majorité des requêtes effectuées sur la Synthese sont générées.

Nous avons lancer des interrogations dans la Synthese en utilisant les filtres par taxon ou par commune. Une fois les requêtes modifiées pour n'utiliser que les champs de nouvelle VM et après l'utilisation de EXPLAIN pour vérifier leur comportement, nous avons pu constaté une amélioration importante des performances !

Par ailleurs, cette approche permet de réutiliser un mécanisme déjà en place dans le module Synthese, l'utilisation de la vue `v_synthese_for_web_app`. L'idée ici est donc de limiter toutes les requêtes à cette vue. *Vue* qui peut devenir *matérialisée* pour exploiter des index sur bases de données GeoNature à forte volumétrie. Cette solution demande finalement qu'un minimum de modification de code.

Nous pouvons continuer à utiliser du GeoJson pour le rendu à partir du moment où les observations sont agrégées par mailles dès que leur nombre devient trop important à afficher sur la carte. Une bascule automatique vers ce type d'affichage doit être mise en place également.

Conclusion : **exécuter des requêtes pour le module Synthese uniquement sur une vue matérialisée `v_synthese_for_web_app` bien indexée.**

Liste des améliorations

L'ensemble de ces tests nous permettent lister les éléments d'amélioration des performances suivant, utiliser :

- le SRID 4326 pour les intersections
- l'opérateur && plutôt que `st_intersects` si possible
- un index `SP-GIST` si les géométrie ne se chevauchant pas, sinon `GIST` plutôt que `BRIN`.
    - Pour `synthese` ⇒ SP-GIST, pour `l_areas` ⇒ GIST.
- une table de relation (`cor_area_synthese`) plutôt qu'une recherche spatiale.
    - ajout d'une colonne `area_type_code` sur `cor_area_synthese`.
- `st_intersects()` sur des géométries complexes simplifiées via `st_subdivide()`.
- une table/VM de cache stockant le nombre d'observation par maille pour gérer l'affichage par défaut sans filtre.
    - sinon utiliser seulement la table `cor_area_synthese` (sans relation avec la table `synthese`) pour les requêtes avec des tailles de mailles appropriées au niveau de zoom pour gérer l'affichage par défaut sans filtre.
- des requêtes s'exécutant uniquement sur une seule table bien indexée sans jointure.
- des requêtes pour le module Synthese s'exécutant uniquement sur une vue matérialisée `v_synthese_for_web_app` bien indexée.

Conclusion sur l'utilisation des tuiles vecteurs

Concernant l'amélioration des performances du module Synthese, les différents tests effectués nous ont permis de nous rendre compte que **l'utilisation des tuiles vecteurs n'était pas à elle seule une solution concluante**. A certain niveau de zoom, lorsque les bounding box des tuiles vecteurs se répartissent bien les mailles d'agrégation ou les observations affichées la parallélisation des requêtes effectuées en base est intéressante.
Malheureusement, cette parallélisation des requêtes n'est pas valable à tous les niveaux de zoom. Avec les petits zoom, nous avons peu de requêtes parallélisées et elles s'exécutent sur de très grande surface et quantité de données. Les performances sont dégradées.

En outre, pour que l'affichage sur la carte soit fluide et garantisse de bonnes conditions d'utilisation à l'utilisateur final, il est nécessaire que la création des tuiles s'effectue très rapidement. À l'exception des requêtes exécutées sur une table mettant en cache le nombre d'observation par mailles, il n'a pas été possible de générer les tuiles assez vite pour garantir une impression de fluidité lors de l'affichage des tuiles sur la carte.

Conclusion sur le framework MapLibre GL

Le framework carto MapLibre GL a été facilement intégré à Angular lors de son utilisation dans le module Syntests. Une modification minime du cœur de GeoNature a toutefois été nécessaire. Nous avons pu constaté qu'il répondait bien à tout les besoins attendus d'une nouveau framework carto utilisable dans le cadre de GeoNature. Il apporte de nouvelles fonctionnalités très intéressantes : affichage du relief, rotation de la carte, traduction instantanée des libellées des cartes, changements de thèmes du rendu carto… C'est donc un choix très intéressant pour le remplacement des Leaflet.

Ceci dit, ce framework étant assez récent, il ne possède pas tous les plugins offerts par Leaflet. L'ajout des fonctionnalités d'édition de la carte s'est avéré ainsi plus complexe qu'avec Leaflet.

Au vue de la forte intégration de Leaflet aux composant générique du frontend de GeoNature et au fait que l'utilisation des tuiles vecteurs ne résolvent pas les problèmes de performance, la bascule vers ce nouveau framework ne semble pas nécessaire. Il semble donc plus pertinent de continuer à utiliser Lealflet avec un rendu des observations à l'aide de GeoJson.

Conclusion générale

Ressources Web

- Module externe de test : Syntests
- Listes des tickets GeoNature traitant du sujet :
  - #1093 - Synthèse : affichage de mailles et observations géo-confidentielles
  - #560 - Synthèse - Travail sur les performances
  - #559 - Synthese: ajout plugin Leaflet clusterMarker
  - #728 - CARTO - permettre l'affichage de plusieurs couches simultanément
- Sources de données :
  - Ressources proposées par OpenMapTiles
  - Présentation des tuiles vecteurs proposée par Etalab
  - Tuiles et styles proposés par IGN
- Amélioration des index spatiaux :
  - (The Many) Spatial Indexes of PostGIS : Paul Ramsey, 5 mai 2021.
  - Quelles sont les meilleures pratiques pour simplifier les géométries et réduire les attributs dans PostGIS pour les tuiles vectorielles ? : IA et communauté Linkedin.
  - PostgreSQL Best Practices: Selection and Optimization of PostGIS Spatial Indexes (GiST, BRIN, and R-tree) : Digoal, 18 décembre 2020.
  - Boosting PostGIS Performance : Symphony, 7 mars 2022.
  - Spatial Indexes and Bad Queries : Paul Ramsey, 4 mai 2021.
  - 5x Faster Spatial Join with this One Weird Trick : Paul Ramsey, 28 septembre 2018.

Analyse du nombres d'observation max par communes et taxons

Requêtes Communes :

```
SELECT
    a.area_name,
    a.area_code,
    COUNT(s.id_synthese) AS obs_nbr
FROM gn_synthese.synthese AS s
    JOIN gn_synthese.cor_area_synthese AS cas
        ON (cas.id_synthese = s.id_synthese AND cas.area_type_code = 'COM')
    JOIN ref_geo.l_areas AS a
        ON cas.id_area = a.id_area
GROUP BY a.area_name, a.area_code
ORDER BY obs_nbr DESC ;

SELECT
    a.area_name,
```

```
    a.area_code,
    COUNT(s.id_synthese) AS obs_nbr
FROM gn_synthese.synthese AS s
    JOIN gn_synthese.cor_area_synthese AS cas
        ON cas.id_synthese = s.id_synthese
    JOIN ref_geo.l_areas AS a
        ON (cas.id_area = a.id_area AND a.id_type =
ref_geo.get_id_area_type_by_code('COM'))
GROUP BY a.area_name, a.area_code
ORDER BY obs_nbr DESC ;
```

Résultats pour les Communes sur la base SINP AURA de 23 millions de données du 2024-05-20 :

- max d'observation pour une commune : 245 043

Requêtes Taxons :

```
SELECT
    t2.cd_nom AS sciname_code,
    t2.lb_nom AS sciname,
    t2.nom_vern AS vernaname,
    COUNT(s.id_synthese) AS obs_nbr
FROM gn_synthese.synthese AS s
    JOIN taxonomie.taxref AS t1
        ON t1.cd_nom = s.cd_nom
    JOIN taxonomie.taxref AS t2
        ON t2.cd_nom = t1.cd_ref
GROUP BY t2.lb_nom, t2.cd_nom
ORDER BY obs_nbr DESC
LIMIT 2000 ;
```

Résultats pour les Taxons sur la base SINP AURA de 23 millions de données du 2024-05-20 :

- taxon avec le max d'observation : 613 465 obs
- taxons avec plus de 150 000 obs : 25 / 32 465 taxons

Utilisation de mailles 20km et 50km

- Source des mailles : https://inpn.mnhn.fr/telechargement/cartes-et-information-geographique
- Procédure :
    - Télécharger manuellement les archives Zip
    - Extraire manuellement les archives
    - Utiliser Shp2pgsql :

Note : le paquet postgis doit être isntallé pour pouvoir utiliser Shp2pgsql.

- M20 : `shp2pgsql METROP_L9320X20.shp ref_geo.m20 | psql -h localhost -p 5432 -U geonatadmin -d gn2_default`
- M50 : `shp2pgsql METROP_L9350X50.shp ref_geo.m50 | psql -h localhost -p 5432 -U geonatadmin -d gn2_default`

- Insérer les mailles M20 :

```
INSERT INTO ref_geo.bib_areas_types (
    type_name,
    type_code,
    type_desc,
    ref_name,
    ref_version,
    num_version,
    size_hierarchy
) VALUES (
    'Mailles20*20',
    'M20',
    'Type maille INPN 20*20km',
    NULL,
    16082021,
    NULL,
    20000
);

INSERT INTO ref_geo.l_areas (
    id_type,
    area_name,
    area_code,
    geom,
    geom_4326,
    centroid,
    "source",
    "comment",
    "enable",
    additional_data
)
    SELECT DISTINCT
        ref_geo.get_id_area_type('M20'),
        m.cd_sig,
        m.code_20km,
        m.geom,
        st_transform(ST_SetSRID(m.geom, 2154), 4326),
        st_centroid(m.geom),
        'INPN',
        NULL,
        TRUE,
        NULL
    FROM ref_geo.m20 AS m
        JOIN ref_geo.tmp_subdivided_areas AS a
            ON (m.geom && a.geom AND a.type_code = 'SINP') ;

CREATE TABLE gn_synthese.cor_area_synthese_m20 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('M20');
```

- Insérer les mailles M50 :

```
INSERT INTO ref_geo.bib_areas_types (
    type_name,
    type_code,
    type_desc,
    ref_name,
    ref_version,
    num_version,
    size_hierarchy
) VALUES (
    'Mailles50*50',
    'M50',
    'Type maille INPN 50*50km',
    NULL,
    16082021,
    NULL,
    50000
);


INSERT INTO ref_geo.l_areas (
    id_type,
    area_name,
    area_code,
    geom,
    geom_4326,
    centroid,
    "source",
    "comment",
    "enable",
    additional_data
)
    SELECT DISTINCT
        ref_geo.get_id_area_type('M50'),
        m.cd_sig,
        m.code_50km,
        m.geom,
        st_transform(ST_SetSRID(m.geom, 2154), 4326),
        st_centroid(m.geom),
        'INPN',
        NULL,
        TRUE,
        NULL
    FROM ref_geo.m50 AS m
            JOIN ref_geo.tmp_subdivided_areas AS a
                ON (m.geom && a.geom AND a.type_code = 'SINP') ;

CREATE TABLE gn_synthese.cor_area_synthese_m50 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
```

```
        FOR VALUES IN ('M50');
```

- Si besoins mettre à jour le champ `area_type_code` :

```
WITH m20 AS (
    SELECT id_area
    FROM ref_geo.l_areas
    WHERE id_type = ref_geo.get_id_area_type('M20')
)
UPDATE gn_synthese.cor_area_synthese AS cas SET
    area_type_code = 'M20'
FROM m20
WHERE cas.id_area = m20.id_area;

WITH m50 AS (
    SELECT id_area
    FROM ref_geo.l_areas
    WHERE id_type = ref_geo.get_id_area_type('M50')
)
UPDATE gn_synthese.cor_area_synthese AS cas SET
    area_type_code = 'M50'
FROM m50
WHERE cas.id_area = m50.id_area;
```

- Activer seulement les mailles M20 de la région SINP :

```
UPDATE ref_geo.l_areas AS a
    SET "enable" = FALSE
WHERE a.id_type = ref_geo.get_id_area_type('M20');

UPDATE ref_geo.l_areas AS a
    SET "enable" = TRUE
FROM ref_geo.tmp_subdivided_areas AS sa
WHERE a.id_type = ref_geo.get_id_area_type('M20')
    AND a.geom && sa.geom
    AND sa.type_code = 'SINP';
```

- Activer seulement les mailles M50 de la région SINP :

```
UPDATE ref_geo.l_areas AS a
    SET "enable" = FALSE
WHERE a.id_type = ref_geo.get_id_area_type('M50');

UPDATE ref_geo.l_areas AS a
    SET "enable" = TRUE
FROM ref_geo.tmp_subdivided_areas AS sa
WHERE a.id_type = ref_geo.get_id_area_type('M50')
    AND a.geom && sa.geom
    AND sa.type_code = 'SINP';
```

Tests d'outils

Utilisation de pg_stat_statements

- Ressource : Doc Postgresql 15 "pg_stat_statements", en français
- `pg_stat_statements` est une extension Postgresql qui permet d'avoir un historique des requêtes lancés, leur fréquence, durée...
    - Étant données que les index spécialisées doivent être créé en fonction des requêtes les plus fréquentes, cet outil va nous servir à les identifiés et suivre leurs évolutions.
- Procédure de mise en place :
    - Définir les paramètres de config Postgresql suivant :

```
shared_preload_libraries = 'pg_stat_statements'
compute_query_id = on
pg_stat_statements.max = 10000
pg_stat_statements.track = all
```

    - Ajouter l'extension suivante à la base de données à surveiller :

```
CREATE EXTENSION pg_stat_statements ;
```

    - Vous pourrez ensuite consultez les requêtes exécutés avec :

```
SELECT pd.datname, pr.rolname, pss.query, pss.calls,
(pss.mean_exec_time/1000) AS sec_mean_exec_time
FROM pg_stat_statements AS pss
    JOIN pg_roles AS pr ON (pr."oid" = pss.userid)
    JOIN pg_catalog.pg_database AS pd ON (pd."oid" = pss.dbid)
WHERE pr.rolname = 'geonatadmin'
    AND pd.datname = 'gn2_default'
    AND pss.mean_exec_time > 500
ORDER BY pss.calls DESC, pss.mean_exec_time DESC;
```

- `pg_stat_statements` permet de connaître les requêtes les plus fréquentes, ce qui permet ensuite de les analyser avec `EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON)` et https://explain.dalibo.com/.
- Voir aussi cet article pour identifiez les index inutilisés.

Test du serveur de fond carto : Tileserver-GL

- Utiliser Docker pour installer le TileServer GL. Debian ne possède pas nativement des paquets pour toutes les dépendances.

Debugger de tuiles vecteurs

- Extension Chrome Mapbox Vector Tile : ajoute un onglet dans les outils développeur]]
- Vector Inspector

- Affichage z,x,y des tuiles - OpenLayers
- Affichage z,x,y des tuiles - Maplibre Planetiler

Test framework carto web Maplibre

- Création des outils de base de visualisation de la carte très facile : zoom, passage en 3D, geolocalisation,
- Liste des plugins Maplibre GL : https://maplibre.org/maplibre-gl-js/docs/plugins/
  - Plugin d'outils d'édition *Mapbox-Gl-Draw* : https://github.com/mapbox/mapbox-gl-draw
    - Comment ajouter un mode (fonctionnalité d'édition) :
    - Commenta ajouter une icone d'outil :
    - Mode édition de rectangle :
    - Mode édition de cercle :
  - Plugin d'ajout de légende : https://github.com/watergis/mapbox-gl-legend
  - Plugin de gestion de couches et de leur opacité : https://github.com/mug-jp/maplibre-gl-opacity
  - Plugin tuiles GeoJson : https://github.com/mkeller3/mapbox-gl-ogc-feature-collection
  - Plugin d'inspection des features (debug) : https://github.com/maplibre/maplibre-gl-inspect
- Intégration de Mapblibre avec Angular "@maplibre/ngx-maplibre-gl" : https://maplibre.org/ngx-maplibre-gl/demo/display-map
  - Documentation API Angular "@maplibre/ngx-maplibre-gl" : https://maplibre.org/ngx-maplibre-gl/doc
  - Code source des exemples Maplibre et Angular : https://github.com/maplibre/ngx-maplibre-gl/tree/c04efff199c1d21016c9bd2d6ba774286c5c3e77/projects/showcase/src/app/demo/examples

Test d'utilisation de mailles hexagonales

- Ressources :
  - Tile Serving with Dynamic Geometry : Paul Ramsey, 24 mars 2020.
  - Fishnets, Honeycombs and Footballs; Better spatial models with hexagonal grids : Dennis Bauszus, 3 mars 2017.
  - Hex grid algorithm for PostGIS : Dennis Bauszus, 3 novembre 2017.

Code utilisé :

```
CREATE OR REPLACE
FUNCTION
public.hexagon(i INTEGER, j INTEGER, edge float8)
RETURNS geometry
AS $$
DECLARE
h float8 := edge*cos(pi()/6.0);
cx float8 := 1.5*i*edge;
cy float8 := h*(2*j+abs(i%2));
BEGIN
RETURN ST_MakePolygon(ST_MakeLine(ARRAY[
            ST_MakePoint(cx - 1.0*edge, cy + 0),
            ST_MakePoint(cx - 0.5*edge, cy + -1*h),
```

```
            ST_MakePoint(cx + 0.5*edge, cy + -1*h),
            ST_MakePoint(cx + 1.0*edge, cy + 0),
            ST_MakePoint(cx + 0.5*edge, cy + h),
            ST_MakePoint(cx - 0.5*edge, cy + h),
            ST_MakePoint(cx - 1.0*edge, cy + 0)
         ]));
END;
$$
LANGUAGE 'plpgsql'
IMMUTABLE
STRICT
PARALLEL SAFE;

CREATE OR REPLACE
FUNCTION public.hexagonCoordinates(bounds geometry, edge float8,
                              OUT i INTEGER, OUT j INTEGER)
RETURNS SETOF record
AS $$
    DECLARE
        h float8 := edge*cos(pi()/6);
        mini INTEGER := FLOOR(st_xmin(bounds) / (1.5*edge));
        minj INTEGER := FLOOR(st_ymin(bounds) / (2*h));
        maxi INTEGER := CEIL(st_xmax(bounds) / (1.5*edge));
        maxj INTEGER := CEIL(st_ymax(bounds) / (2*h));
    BEGIN
    FOR i, j IN
    SELECT a, b
    FROM generate_series(mini, maxi) a,
         generate_series(minj, maxj) b
    LOOP
       RETURN NEXT;
    END LOOP;
    END;
$$
LANGUAGE 'plpgsql'
IMMUTABLE
STRICT
PARALLEL SAFE;

CREATE OR REPLACE
FUNCTION public.tileHexagons(z INTEGER, x INTEGER, y INTEGER, step INTEGER,
                       OUT geom geometry(Polygon, 3857), OUT i INTEGER, OUT j
INTEGER)
RETURNS SETOF record
AS $$
    DECLARE
        bounds geometry;
        maxbounds geometry := ST_TileEnvelope(0, 0, 0);
        edge float8;
    BEGIN
```

```
    bounds := ST_TileEnvelope(z, x, y);
    edge := (ST_XMax(bounds) - ST_XMin(bounds)) / pow(2, step);
    FOR geom, i, j IN
    SELECT ST_SetSRID(hexagon(h.i, h.j, edge), 3857), h.i, h.j
    FROM hexagoncoordinates(bounds, edge) h
    LOOP
       IF maxbounds ~ geom AND bounds && geom THEN
            RETURN NEXT;
         END IF;
     END LOOP;
     END;
$$
LANGUAGE 'plpgsql'
IMMUTABLE
STRICT
PARALLEL SAFE;

CREATE OR REPLACE
FUNCTION public.hexagons(z INTEGER, x INTEGER, y INTEGER, step INTEGER
DEFAULT 4)
RETURNS bytea
AS $$
WITH
bounds AS (
    -- Convert tile coordinates to web mercator tile bounds
    SELECT ST_TileEnvelope(z, x, y) AS geom
 ),
 ROWS AS (
    -- All the hexes that interact with this tile
    SELECT h.i, h.j, h.geom
    FROM TileHexagons(z, x, y, step) h
 ),
 mvt AS (
     -- Usual tile processing, ST_AsMVTGeom simplifies, quantizes,
     -- and clips to tile boundary
    SELECT ST_AsMVTGeom(ROWS.geom, bounds.geom) AS geom,
           ROWS.i, ROWS.j
    FROM ROWS, bounds
)
-- Generate MVT encoding of final input record
SELECT ST_AsMVT(mvt, 'hexagons') FROM mvt
$$
LANGUAGE 'sql'
STABLE
STRICT
PARALLEL SAFE;

COMMENT ON FUNCTION public.hexagons IS 'Hex coverage dynamically generated.
Step parameter determines how approximately many hexes (2^step) to generate
per tile.';
```

```
-- Test de requête
-- 10, 523, 369
WITH bounds AS (
    -- Convert tile coordinates to web mercator tile bounds
    SELECT ST_TileEnvelope(11, 1046, 739) AS geom
),
ROWS AS (
    -- All the hexes that interact with this tile
    SELECT
        h.i,
        h.j,
        h.geom,
        st_transform(h.geom, 4326) AS geom_4326
    FROM tileHexagons(9, 261, 184, 4) AS h
 ),
observations AS (
    SELECT
        r.geom,
        r.i,
        r.j,
        COUNT(s.id_synthese) AS nbr
    FROM gn_synthese.synthese AS s
        JOIN "rows" AS r
            ON st_intersects(s.the_geom_4326, r.geom_4326)
    GROUP BY r.geom, r.i, r.j
)
-- Usual tile processing, ST_AsMVTGeom simplifies, quantizes,
 -- and clips to tile boundary
SELECT
    ST_AsMVTGeom(o.geom, b.geom) AS geom,
    o.i,
    o.j,
    o.nbr
FROM observations AS o, bounds AS b
WHERE ST_AsMVTGeom(o.geom, b.geom) IS NOT NULL;
```

Résultats : l'agrégation d'observations sur des mailles hexagonales générées dynamiquement est assez rapide aux niveaux de zoom 10 et plus.

Tests de requête SQL d'aggrégation

```
SELECT
   t.id_area,
   t.area_code,
   t.area_name,
   t.geom
FROM ref_geo.l_areas AS t
```

```
WHERE ST_Intersects(t.geom, ST_Transform(ST_TileEnvelope(12, 2117, 1479),
2154))
    AND t.id_type = ref_geo.get_id_area_type('COM');

SELECT public.squares(12, 2117, 1479);

-- Communes
SELECT ST_TileEnvelope(12, 2117, 1479) AS envelope; -- 9 row(s) fetched -
0,010s (0,001s fetch), on 2024-03-19 at 16:32:36
SELECT ST_TileEnvelope(9, 264, 184) AS envelope; -- 155 row(s) fetched -
0,073s (0,019s fetch), on 2024-03-19 at 16:32:17
SELECT ST_TileEnvelope(8, 132, 92) AS envelope; -- 408 row(s) fetched -
0,188s (0,071s fetch), on 2024-03-19 at 16:36:21
SELECT ST_TileEnvelope(7, 66, 46) AS envelope; -- 873 row(s) fetched -
0,366s (0,097s fetch), on 2024-03-19 at 16:35:29
SELECT ST_TileEnvelope(6, 33, 23) AS envelope; -- 1311 row(s) fetched -
0,490s (0,113s fetch), on 2024-03-19 at 16:37:22
SELECT ST_TileEnvelope(6, 32, 23) AS envelope; -- 6763 row(s) fetched - 1s
(0,439s fetch), on 2024-03-19 at 16:37:55
SELECT ST_TileEnvelope(5, 16, 11) AS envelope; -- 24849 row(s) fetched - 4s
(0,924s fetch), on 2024-03-19 at 16:31:44
SELECT ST_TileEnvelope(4, 8, 5) AS envelope; -- 30521 row(s) fetched - 4s
(0,690s fetch), on 2024-03-19 at 16:34:23

WITH bounds AS (
    SELECT ST_TileEnvelope(8, 132, 92) AS envelope
)
SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), b.envelope) AS geom,
       t.id_area,
       t.area_code,
       t.area_name
FROM ref_geo.l_areas AS t, bounds AS b
WHERE t.geom && ST_Transform(b.envelope, 2154)
    AND t.id_type = ref_geo.get_id_area_type('COM') ;

-- Maille M1
SELECT ST_TileEnvelope(12, 2117, 1479) AS envelope; -- 72 row(s) fetched -
0,003s (0,001s fetch), on 2024-03-19 at 16:57:41
SELECT ST_TileEnvelope(8, 132, 92) AS envelope; -- 13161 row(s) fetched -
0,211s (0,079s fetch), on 2024-03-19 at 16:59:17
SELECT ST_TileEnvelope(4, 8, 5) AS envelope; -- 481502 row(s) fetched - 6s
(3s fetch), on 2024-03-19 at 17:00:30

WITH bounds AS (
    SELECT ST_TileEnvelope(4, 8, 5) AS envelope
)
SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), b.envelope) AS geom,
       t.id_area,
       t.area_code,
       t.area_name
FROM ref_geo.l_areas AS t, bounds AS b
```

```
WHERE t.geom && ST_Transform(b.envelope, 2154)
    AND t.id_type = ref_geo.get_id_area_type('M1') ;


-- Maille M5
SELECT ST_TileEnvelope(12, 2117, 1479) AS envelope; -- 6 row(s) fetched -
0,005s (0,001s fetch), on 2024-03-19 at 16:58:06
SELECT ST_TileEnvelope(8, 132, 92) AS envelope; --  533 row(s) fetched -
0,021s (0,006s fetch), on 2024-03-19 at 16:58:32
SELECT ST_TileEnvelope(4, 8, 5) AS envelope; -- 18856 row(s) fetched -
0,329s (0,119s fetch), on 2024-03-19 at 17:01:26

WITH bounds AS (
    SELECT ST_TileEnvelope(4, 8, 5) AS envelope
)
SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), b.envelope) AS geom,
       t.id_area,
       t.area_code,
       t.area_name
FROM ref_geo.l_areas AS t, bounds AS b
WHERE t.geom && ST_Transform(b.envelope, 2154)
    AND t.id_type = ref_geo.get_id_area_type('M5') ;

-- Maille M10
SELECT ST_TileEnvelope(12, 2117, 1479) AS envelope; -- 2 row(s) fetched -
0,002s, on 2024-03-19 at 17:03:21
SELECT ST_TileEnvelope(8, 132, 92) AS envelope; -- 156 row(s) fetched -
0,005s (0,001s fetch), on 2024-03-19 at 17:02:58
SELECT ST_TileEnvelope(4, 8, 5) AS envelope; -- 6013 row(s) fetched - 0,088s
(0,036s fetch), on 2024-03-19 at 17:02:07

WITH bounds AS (
    SELECT ST_TileEnvelope(12, 2117, 1479) AS envelope
)
SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), b.envelope) AS geom,
       t.id_area,
       t.area_code,
       t.area_name
FROM ref_geo.l_areas AS t, bounds AS b
WHERE t.geom && ST_Transform(b.envelope, 2154)
    AND t.id_type = ref_geo.get_id_area_type('M10') ;

WITH bounds AS (
    SELECT ST_TileEnvelope(12, 2117, 1479) AS envelope
),
observations AS (
    SELECT
        l.area_code AS code,
        ST_Transform(l.geom, 3857) AS geom,
        COUNT(s.id_synthese) AS nbr
```

```
    FROM gn_synthese.synthese AS s
        JOIN gn_synthese.cor_area_synthese AS cas
            ON s.id_synthese = cas.id_synthese
        JOIN ref_geo.l_areas AS l
            ON l.id_area = cas.id_area
        , bounds AS b
    WHERE l.geom && ST_Transform(b.envelope, 2154)
        AND l.id_type = ref_geo.get_id_area_type('M10')
    GROUP BY l.area_code, l.geom
)
SELECT ST_AsMVTGeom(o.geom, b.envelope) AS geom,
       o.nbr,
       o.code
FROM observations AS o, bounds AS b ;

WITH bounds AS (
    SELECT ST_TileEnvelope(9, 264, 184) AS envelope
)
SELECT
    l.area_code AS code,
    ST_Transform(l.geom, 3857) AS geom,
    COUNT(s.id_synthese) AS nbr
FROM gn_synthese.synthese AS s
    JOIN gn_synthese.cor_area_synthese AS cas
        ON s.id_synthese = cas.id_synthese
    JOIN ref_geo.l_areas AS l
        ON l.id_area = cas.id_area
    , bounds AS b
WHERE l.geom && ST_Transform(b.envelope, 2154)
    AND l.id_type = ref_geo.get_id_area_type('M10')
GROUP BY l.area_code, l.geom ; -- 32 row(s) fetched - 7s (0,001s fetch), on
2024-03-19 at 17:20:06

WITH bounds AS (
    SELECT ST_TileEnvelope(9, 264, 184) AS envelope
)
SELECT l.id_area, l.geom, l.area_code
FROM ref_geo.l_areas AS l, bounds AS b
WHERE l.geom && ST_Transform(b.envelope, 2154)
    AND l.id_type = ref_geo.get_id_area_type('M10') ; -- 32 row(s) fetched -
0,007s (0,001s fetch), on 2024-03-19 at 17:29:37

WITH bounds AS (
    SELECT ST_TileEnvelope(9, 264, 184) AS envelope
),
m10 AS (
    SELECT
        l.id_area,
        l.geom,
        l.area_code AS code
    FROM ref_geo.l_areas AS l, bounds AS b
```

```
    WHERE l.geom && ST_Transform(b.envelope, 2154)
        AND l.id_type = ref_geo.get_id_area_type('M10')
)
SELECT
    m10.code,
    ST_Transform(m10.geom, 3857) AS geom,
    COUNT(s.id_synthese) AS nbr
FROM gn_synthese.synthese AS s
    JOIN gn_synthese.cor_area_synthese AS cas
        ON s.id_synthese = cas.id_synthese
    JOIN m10
        ON cas.id_area = m10.id_area
GROUP BY m10.code, m10.geom ; -- 32 row(s) fetched - 7s, on 2024-03-19 at
17:39:22

WITH bounds AS (
    SELECT ST_TileEnvelope(9, 264, 184) AS envelope
),
m10 AS (
    SELECT l.id_area, l.geom, l.area_code AS code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom && ST_Transform(b.envelope, 2154)
        AND l.id_type = ref_geo.get_id_area_type('M10')
)
SELECT
    m10.code,
    ST_Transform(m10.geom, 3857) AS geom,
    COUNT(s.id_synthese) AS nbr
FROM gn_synthese.synthese AS s
    JOIN m10
        ON s.the_geom_local && m10.geom
GROUP BY m10.code, m10.geom ; -- 32 row(s) fetched - 27s (0,001s fetch), on
2024-03-19 at 17:35:57


WITH bounds AS (
    SELECT ST_TileEnvelope(8, 132, 92) AS envelope
), mvtgeom AS (
    SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), b.envelope) AS geom,
           t.id_area,
           t.area_code,
           t.area_name
    FROM ref_geo.l_areas AS t, bounds AS b
    WHERE ST_Intersects(t.geom, ST_Transform(b.envelope, 2154))
        AND t.id_type = ref_geo.get_id_area_type('COM')
)
SELECT ST_AsMVT(mvtgeom.*) FROM mvtgeom ;
```

Annexes

Annexe 1 - Comparaison SRID 4326 et 2154

```
-- 14, 8372, 5916  -- 10 row(s) fetched - 0,007s, on 2024-03-22 at 17:52:27
-- 13, 4185, 2957  -- 45 row(s) fetched - 0,042s (0,001s fetch), on
2024-03-22 at 17:46:28
-- 12, 2092, 1478  -- 545 row(s) fetched - 0,051s (0,014s fetch), on
2024-03-22 at 17:55:10
-- 11, 1046, 739   -- 2223 row(s) fetched - 0,152s (0,044s fetch), on
2024-03-22 at 17:54:35
-- 10, 523, 369    -- 18639 row(s) fetched - 1s (0,283s fetch), on
2024-03-22 at 17:54:54
WITH tile AS (
    SELECT ST_TileEnvelope(12, 2092, 1478) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
observations AS (
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON ST_Intersects(b.envelope_4326, s.the_geom_4326)
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;


-- 14, 8372, 5916  -- 13 row(s) fetched - 0,012s (0,001s fetch), on
2024-03-22 at 17:51:53
-- 13, 4185, 2957  -- 47 row(s) fetched - 0,045s (0,001s fetch), on
2024-03-22 at 17:52:51
-- 12, 2092, 1478  -- 564 row(s) fetched - 0,076s (0,015s fetch), on
```

```
2024-03-22 at 17:53:16
-- 11, 1046, 739   -- 2272 row(s) fetched - 0,229s (0,048s fetch), on
2024-03-22 at 17:53:33
-- 10, 523, 369    -- 18886 row(s) fetched - 1s (0,321s fetch), on
2024-03-22 at 17:53:50
WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 2154) AS envelope_2154
     FROM tile AS t
),
observations AS (
    SELECT
        s.the_geom_local AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON ST_Intersects(b.envelope_2154, s.the_geom_local)
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;
```

Annexe 2 - Comparaison opérateur && et st_intersects avec index GIST

```
DROP INDEX IF EXISTS gn_synthese.idx_synthese_the_geom_4326_brin;

CREATE INDEX i_synthese_the_geom_4326 ON gn_synthese.synthese USING GIST
(the_geom_4326) ;

-- 14, 8372, 5916  -- 10 row(s) fetched - 0,022s (0,001s fetch), on
2024-03-22 at 18:13:39
-- 13, 4185, 2957  -- 45 row(s) fetched - 0,046s (0,001s fetch), on
2024-03-22 at 18:14:47
-- 12, 2092, 1478  -- 545 row(s) fetched - 0,058s (0,013s fetch), on
2024-03-22 at 18:14:30
-- 11, 1046, 739   -- 2223 row(s) fetched - 0,179s (0,044s fetch), on
```

```
2024-03-22 at 18:15:06
-- 10, 523, 369    -- 18639 row(s) fetched - 1s (0,288s fetch), on
2024-03-22 at 18:15:21
WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
observations AS (
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON ST_Intersects(b.envelope_4326, s.the_geom_4326)
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;


-- 14, 8372, 5916  -- 10 row(s) fetched - 0,004s (0,001s fetch), on
2024-03-22 at 18:15:52
-- 13, 4185, 2957  -- 46 row(s) fetched - 0,010s (0,002s fetch), on
2024-03-22 at 18:16:04
-- 12, 2092, 1478  -- 545 row(s) fetched - 0,043s (0,013s fetch), on
2024-03-22 at 18:16:17
-- 11, 1046, 739   -- 2223 row(s) fetched - 0,162s (0,044s fetch), on
2024-03-22 at 18:16:43
-- 10, 523, 369    -- 18640 row(s) fetched - 0,954s (0,419s fetch), on
2024-03-22 at 18:16:57
WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
```

```
     FROM tile AS t
),
observations AS (
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON b.envelope_4326 && s.the_geom_4326
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;
```

Annexe 3 - Comparaison opérateur && et st_intersects avec index BRIN

```
DROP INDEX gn_synthese.i_synthese_the_geom_4326;
CREATE INDEX idx_synthese_the_geom_4326_brin ON gn_synthese.synthese USING
BRIN(the_geom_4326) WITH (pages_per_range = 1);

-- 14, 8372, 5916  -- 10 row(s) fetched - 2s (0,001s fetch), on 2024-03-22
at 18:02:09
-- 13, 4185, 2957  -- 45 row(s) fetched - 1s (0,001s fetch), on 2024-03-22
at 18:02:40
-- 12, 2092, 1478  -- 545 row(s) fetched - 1s (0,014s fetch), on 2024-03-22
at 18:03:00
-- 11, 1046, 739   -- 2223 row(s) fetched - 1s (0,051s fetch), on 2024-03-22
at 18:03:21
-- 10, 523, 369    -- 18639 row(s) fetched - 2s (0,300s fetch), on
2024-03-22 at 18:03:42
EXPLAIN (analyze,verbose,timing,costs,buffers) WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
observations AS (
```

```
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON ST_Intersects(b.envelope_4326, s.the_geom_4326)
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;


-- 14, 8372, 5916  -- 10 row(s) fetched - 1s, on 2024-03-22 at 18:04:51
-- 13, 4185, 2957  -- 46 row(s) fetched - 1s (0,001s fetch), on 2024-03-22
at 18:05:13
-- 12, 2092, 1478  -- 545 row(s) fetched - 1s (0,012s fetch), on 2024-03-22
at 18:05:30
-- 11, 1046, 739   -- 2223 row(s) fetched - 1s (0,042s fetch), on 2024-03-22
at 18:05:48
-- 10, 523, 369    -- 18640 row(s) fetched - 2s (0,291s fetch), on
2024-03-22 at 18:06:05
WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
observations AS (
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON b.envelope_4326 && s.the_geom_4326
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
```

```
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;
```

Annexe 4 - Comparaison index GIST et SP-GIST

```
-- 14, 8372, 5916  -- 10 row(s) fetched - 0,004s (0,001s fetch), on
2024-03-22 at 18:15:52
-- 13, 4185, 2957  -- 46 row(s) fetched - 0,010s (0,002s fetch), on
2024-03-22 at 18:16:04
-- 12, 2092, 1478  -- 545 row(s) fetched - 0,043s (0,013s fetch), on
2024-03-22 at 18:16:17
-- 11, 1046, 739   -- 2223 row(s) fetched - 0,162s (0,044s fetch), on
2024-03-22 at 18:16:43
-- 10, 523, 369    -- 18640 row(s) fetched - 0,954s (0,419s fetch), on
2024-03-22 at 18:16:57
WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
observations AS (
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON b.envelope_4326 && s.the_geom_4326
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
```

```
GROUP BY o.geom, o.dimension, o.perimeter ;

DROP INDEX IF EXISTS gn_synthese.i_synthese_the_geom_4326;

CREATE INDEX i_synthese_the_geom_4326 ON gn_synthese.synthese USING SPGIST
(the_geom_4326) ;

-- 14, 8372, 5916  -- 10 row(s) fetched - 0,005s, on 2024-03-22 at 20:36:49
-- 13, 4185, 2957  -- 46 row(s) fetched - 0,011s (0,001s fetch), on
2024-03-22 at 20:35:37
-- 12, 2092, 1478  -- 545 row(s) fetched - 0,037s (0,012s fetch), on
2024-03-22 at 20:35:53
-- 11, 1046, 739   -- 2223 row(s) fetched - 0,156s (0,042s fetch), on
2024-03-22 at 20:36:29
-- 10, 523, 369    -- 18640 row(s) fetched - 0,871s (0,287s fetch), on
2024-03-22 at 20:37:08
WITH tile AS (
    SELECT ST_TileEnvelope(10, 523, 369) AS envelope
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
observations AS (
    SELECT
        st_snaptogrid(s.the_geom_4326, 0.00001) AS geom,
        ST_Dimension(s.the_geom_local) AS dimension,
        round(ST_Perimeter(s.the_geom_local)) AS perimeter,
        s.id_synthese AS id,
        s."precision"
    FROM gn_synthese.synthese AS s
        JOIN bounds AS b
            ON b.envelope_4326 && s.the_geom_4326
    WHERE round(ST_Perimeter(s.the_geom_local)) <= 4000
)
SELECT
    ST_Transform(o.geom, 3857),
    o.dimension,
    o.perimeter,
    COUNT(o.id) AS nbr,
    json_agg(o.id) AS ids
FROM observations AS o
GROUP BY o.geom, o.dimension, o.perimeter ;
```

Annexe 5 - Comparaison agrégation via table relation et via intersection

```
DROP INDEX IF EXISTS ref_geo.idx_l_areas_geom_4326;
```

```
CREATE INDEX idx_l_areas_geom_4326 ON ref_geo.l_areas USING
spgist(geom_4326) ;

-- Via table de relation cor_are_synthese
-- 14, 8372, 5915  ×   --
-- 13, 4186, 2957  ×   --
-- 12, 2093, 1478  M1  -- 64 row(s) fetched - 0,218s, on 2024-03-22 at
21:08:05
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,347s (0,002s fetch), on
2024-03-22 at 21:08:24
-- 10, 523, 369    M5  -- 47 row(s) fetched - 0,805s (0,001s fetch), on
2024-03-22 at 21:09:05
-- 9, 261, 184     M5  -- 118 row(s) fetched - 1s (0,001s fetch), on
2024-03-22 at 21:09:29
-- 8, 131, 92      M10 -- 128 row(s) fetched - 6s (0,001s fetch), on
2024-03-22 at 21:09:56
-- 7, 65, 46       M10 -- 187 row(s) fetched - 9s (0,002s fetch), on
2024-03-22 at 21:10:39
-- 6, 32, 23       M10 -- 219 row(s) fetched - 10s (0,002s fetch), on
2024-03-22 at 21:11:06
-- 5, 16, 11       M10 -- 819 row(s) fetched - 1m 9s (0,011s fetch), on
2024-03-28 at 15:46:54
WITH tile AS (
    SELECT ST_TileEnvelope(12, 2093, 1478) AS envelope -- SRID 3857
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        a.id_area AS id,
        a.geom,
        a.area_code AS code
    FROM ref_geo.l_areas AS a, bounds AS b
    WHERE a.geom_4326 && b.envelope_4326
        AND a.id_type = ref_geo.get_id_area_type('M10')
)
SELECT
    ST_Transform(a.geom, 3857),
    a.code,
    COUNT(s.id_synthese) AS nbr
FROM gn_synthese.synthese AS s
    JOIN gn_synthese.cor_area_synthese AS cas
        ON s.id_synthese = cas.id_synthese
    JOIN areas AS a
        ON a.id = cas.id_area
GROUP BY a.geom, a.code ;
```

```
-- Via opérateur d'intersection &&
-- 12, 2093, 1478  M1  -- 64 row(s) fetched - 0,125s (0,002s fetch), on
2024-03-22 at 21:17:50
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,315s (0,003s fetch), on
2024-03-22 at 21:18:15
-- 10, 523, 369    M5  -- 47 row(s) fetched - 1s (0,001s fetch), on
2024-03-22 at 21:18:43
-- 9, 261, 184     M5  -- 118 row(s) fetched - 1s (0,004s fetch), on
2024-03-22 at 21:19:04
-- 8, 131, 92      M10 -- 128 row(s) fetched - 52s (0,002s fetch), on
2024-03-22 at 21:20:21
-- 7, 65, 46       M10 -- ×
-- 6, 32, 23       M10 -- ×
-- 5, 16, 11       M10 -- ×
WITH tile AS (
    SELECT ST_TileEnvelope(8, 131, 92) AS envelope -- SRID 3857
),
bounds AS (
    SELECT
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        a.geom_4326 AS geom,
        a.id_area AS id,
        a.area_code AS code
    FROM ref_geo.l_areas AS a, bounds AS b
    WHERE a.geom_4326 && b.envelope_4326
        AND a.id_type = ref_geo.get_id_area_type('M10')
)
SELECT
    ST_Transform(a.geom, 3857),
    a.code,
    COUNT(s.id_synthese) AS nbr
FROM gn_synthese.synthese AS s
    JOIN areas AS a
        ON a.geom && s.the_geom_4326
GROUP BY a.geom, a.code ;
```

Annexe 6 - Ajout champ area_type_code à cor_area_synthese

```
-- Add column area_type_code to cor_area_synthese
ALTER TABLE gn_synthese.cor_area_synthese
ADD area_type_code VARCHAR(25) DEFAULT NULL;

UPDATE gn_synthese.cor_area_synthese AS cas SET
```

```
    area_type_code = t.type_code
FROM ref_geo.l_areas AS a
    JOIN ref_geo.bib_areas_types AS t
        ON a.id_type = t.id_type
WHERE a.id_area = cas.id_area ;


CREATE OR REPLACE FUNCTION
gn_synthese.fct_trig_l_areas_insert_cor_area_synthese_on_each_statement()
 RETURNS TRIGGER
 LANGUAGE plpgsql
AS $function$
    DECLARE
    BEGIN
        -- Intersection de toutes les observations avec les nouvelles zones
et écriture dans cor_area_synthese
        INSERT INTO gn_synthese.cor_area_synthese (id_area, id_synthese,
area_type_code)
            SELECT
                new_areas.id_area AS id_area,
                s.id_synthese AS id_synthese,
                bat.type_code
            FROM NEW AS new_areas
                JOIN ref_geo.bib_areas_types AS bat
                    ON new_areas.id_type = bat.id_type
                JOIN gn_synthese.synthese AS s
                    ON public.ST_INTERSECTS(s.the_geom_local,
new_areas.geom)
            WHERE new_areas."enable" IS TRUE
                AND (
                    ST_GeometryType(s.the_geom_local) = 'ST_Point'
                    OR
                    NOT public.ST_TOUCHES(s.the_geom_local, new_areas.geom)
            );
        RETURN NULL;
    END;
$function$ ;
```

Annexe 7 - Test de différents index sur area_type_code

```
-- Tests de 3 types d'index :
-- 379 millions de ligne

CREATE INDEX idx_cas_area_type_code ON gn_synthese.cor_area_synthese USING
hash(area_type_code) ;
--> création en + de 22h => abandon !

CREATE INDEX idx_cas_area_type_code ON gn_synthese.cor_area_synthese USING
btree(area_type_code) ;
```

```
--> création en ~2mn => 2,4G d'espace disque

CREATE INDEX idx_cas_area_type_code_id_area ON gn_synthese.cor_area_synthese
USING btree(area_type_code, id_area) ;
--> création en 3mn 16s => 2,5G d'espace disque

CREATE INDEX idx_cas_area_include_synthese ON gn_synthese.cor_area_synthese
USING btree(area_type_code, id_area) INCLUDE (id_synthese);
--> création en 3mn 51s => 12G d'espace disque (avec mailles M20, M50)
```

Annexe 8 - Test des différents index sur area_type_code

```
-- Agrégation avec champ "area_type_code" dans "cor_area_synthese" avec idx
btree(area_type_code)
-- 12, 2093, 1478  M1  -- 64  row(s) fetched - 0,253s, on 2024-03-24 at
21:28:13
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,465s (0,002s fetch), on
2024-03-24 at 21:27:45
-- 10, 523, 369    M5  -- 47  row(s) fetched - 0,804s (0,001s fetch), on
2024-03-24 at 21:29:06
-- 9, 261, 184     M5  -- 118 row(s) fetched - 1s (0,002s fetch), on
2024-03-24 at 21:29:26
-- 9, 261, 184     M10 -- 35  row(s) fetched - 2s, on 2024-03-24 at 21:26:28
-- 8, 131, 92      M10 -- 128 row(s) fetched - 6s (0,002s fetch), on
2024-03-24 at 21:30:04
-- 7, 65, 46       M10 -- 187 row(s) fetched - 9s (0,002s fetch), on
2024-03-24 at 21:30:35
-- 6, 32, 23       M10 -- 219 row(s) fetched - 10s (0,003s fetch), on
2024-03-24 at 21:22:44
-- 5, 16, 11       M10 -- 819 row(s) fetched - 1m 27s (0,009s fetch), on
2024-03-24 at 21:32:47

-- Agrégation avec champ "area_type_code" dans "cor_area_synthese" avec idx
btree(area_type_code, id_area)
-- 12, 2093, 1478  M1  -- 64  row(s) fetched - 0,114s, on 2024-03-24 at
11:28:08
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,588s (0,003s fetch), on
2024-03-24 at 11:26:21
-- 10, 523, 369    M5  -- 47  row(s) fetched - 0,732s, on 2024-03-24 at
11:23:10
-- 9, 261, 184     M5  -- 118 row(s) fetched - 1s (0,002s fetch), on
2024-03-24 at 11:22:44
-- 9, 261, 184     M10 -- 35  row(s) fetched - 1s (0,001s fetch), on
2024-03-24 at 21:38:29
-- 8, 131, 92      M10 -- 128 row(s) fetched - 6s (0,001s fetch), on
2024-03-24 at 21:39:33
-- 7, 65, 46       M10 -- 187 row(s) fetched - 9s (0,003s fetch), on
2024-03-24 at 21:40:24
-- 6, 32, 23       M10 -- 219 row(s) fetched - 10s (0,003s fetch), on
2024-03-24 at 21:40:53
```

```
-- 5, 16, 11       M10 -- 819 row(s) fetched - 55s (0,009s fetch), on
2024-03-24 at 21:15:34
WITH tile AS (
    SELECT
        'M10' AS type_code,
        ST_TileEnvelope(5, 16, 11) AS envelope -- SRID 3857
),
bounds AS (
    SELECT
        t.type_code,
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        l.id_area AS id,
        l.geom,
        l.area_code AS code,
        b.type_code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom_4326 && b.envelope_4326
        AND l.id_type = ref_geo.get_id_area_type(b.type_code)
)
SELECT
    ST_Transform(a.geom, 3857),
    a.code,
    COUNT(s.id_synthese) AS nbr
FROM areas AS a
    JOIN gn_synthese.cor_area_synthese AS cas
        ON (a.id = cas.id_area AND cas.area_type_code = a.type_code)
    JOIN gn_synthese.synthese AS s
        ON s.id_synthese = cas.id_synthese
GROUP BY a.geom, a.code ;
```

Annexe 9 - Création vue matérialisée cor_m10_synthese

```
-- Create a specialized table to store the relationship between synthese
observations and M10 meshes
CREATE MATERIALIZED VIEW IF NOT EXISTS gn_synthese.cor_m10_synthese AS
    SELECT
        s.id_synthese,
        a.id_area
    FROM ref_geo.l_areas AS a
        JOIN gn_synthese.synthese AS s
            ON (a.geom && s.the_geom_local) -- Postgis operator && :
https://postgis.net/docs/geometry_overlaps.html
    WHERE a.id_type = ref_geo.get_id_area_type('M10') ;
```

```
CREATE UNIQUE INDEX pk_cor_m10_synthese ON gn_synthese.cor_m10_synthese
USING btree (id_synthese, id_area) ;
CREATE INDEX i_cor_m10_synthese_id_area ON gn_synthese.cor_m10_synthese
USING btree (id_area) ;
```

Annexe 10 - Test utilisation de la table "cor_m10_synthese"

```
-- Agrégation avec utilisation de la table cor_m10_synthese
-- 8, 131, 92      M10 -- 128 row(s) fetched - 6s  (0,001s fetch), on
2024-03-25 at 21:05:11
-- 7, 65, 46       M10 -- 187 row(s) fetched - 8s  (0,003s fetch), on
2024-03-25 at 21:06:16
-- 6, 32, 23       M10 -- 219 row(s) fetched - 10s (0,003s fetch), on
2024-03-25 at 21:04:51
-- 5, 16, 11       M10 -- 819 row(s) fetched - 53s (0,010s fetch), on
2024-03-25 at 21:04:26
WITH tile AS (
    SELECT
        'M10' AS type_code,
        ST_TileEnvelope(7, 65, 46) AS envelope -- SRID 3857
),
bounds AS (
    SELECT
        t.type_code,
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        l.id_area AS id,
        l.geom,
        l.area_code AS code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom_4326 && b.envelope_4326
        AND l.id_type = ref_geo.get_id_area_type(b.type_code)
)
SELECT
    ST_Transform(a.geom, 3857),
    a.code,
    COUNT(s.id_synthese) AS nbr
FROM gn_synthese.synthese AS s
    JOIN gn_synthese.cor_m10_synthese AS cas
        ON s.id_synthese = cas.id_synthese
    JOIN areas AS a
        ON (a.id = cas.id_area)
GROUP BY a.geom, a.code ;
```

Annexe 11 - Création vue matérialisée m10_observation_nbr

```
-- Create a cache table to store the number of observations per M10 mesh
only
CREATE MATERIALIZED VIEW IF NOT EXISTS gn_synthese.m10_observation_nbr AS
    SELECT
        a.id_area,
        COUNT(s.id_synthese) AS nbr
    FROM gn_synthese.synthese AS s
        JOIN gn_synthese.cor_area_synthese AS cas
            ON s.id_synthese = cas.id_synthese
        JOIN ref_geo.l_areas AS a
            ON a.id_area = cas.id_area
    WHERE a.id_type = ref_geo.get_id_area_type('M10')
    GROUP BY a.id_area ;

CREATE UNIQUE INDEX pk_m10_observation_nbr ON
gn_synthese.m10_observation_nbr USING btree (id_area) ;
```

Annexe 12 - Création vue matérialisée observation_nbr

```
-- Create a cache table to store the number of observations per mesh (all
size)
DROP MATERIALIZED VIEW IF EXISTS gn_synthese.observation_nbr ;
CREATE MATERIALIZED VIEW IF NOT EXISTS gn_synthese.observation_nbr AS
    SELECT
        a.id_area,
        bat.type_code AS type_code,
        COUNT(s.id_synthese) AS obs_nbr
    FROM gn_synthese.synthese AS s
        JOIN gn_synthese.cor_area_synthese AS cas
            ON s.id_synthese = cas.id_synthese
        JOIN ref_geo.l_areas AS a
            ON a.id_area = cas.id_area
        JOIN ref_geo.bib_areas_types AS bat
            ON a.id_type = bat.id_type
    WHERE a.id_type IN (
        ref_geo.get_id_area_type('M10'),
        ref_geo.get_id_area_type('M5'),
        ref_geo.get_id_area_type('M1')
    )
    GROUP BY a.id_area, bat.type_code
    ORDER BY bat.type_code, a.id_area ;

CREATE UNIQUE INDEX pk_observation_nbr ON gn_synthese.observation_nbr USING
btree (id_area, type_code) ;
```

Annexe 13 - Tests utilisation vue matérialisée m10_observation_nbr

```
-- Agrégation avec utilisation de la table m10_observation_nbr
-- 8, 131, 92      M10 -- 128 row(s) fetched - 0,013s (0,003s fetch), on
2024-03-29 at 17:16:08
-- 7, 65, 46       M10 -- 187 row(s) fetched - 0,014s (0,007s fetch), on
2024-03-29 at 17:15:50
-- 6, 32, 23       M10 -- 219 row(s) fetched - 0,017s (0,007s fetch), on
2024-03-29 at 17:16:34
-- 5, 16, 11       M10 -- 819 row(s) fetched - 0,048s (0,021s fetch), on
2024-03-29 at 17:12:34
WITH tile AS (
    SELECT
        'M10' AS type_code,
        ST_TileEnvelope(8, 131, 92) AS envelope -- SRID 3857
),
bounds AS (
    SELECT
        t.type_code,
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        l.id_area AS id,
        l.geom,
        l.area_code AS code,
        b.type_code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom_4326 && b.envelope_4326
        AND l.id_type = ref_geo.get_id_area_type(b.type_code)
)
SELECT
    ST_Transform(a.geom, 3857),
    a.code,
    o.nbr
FROM gn_synthese.m10_observation_nbr AS o
    JOIN areas AS a
        ON a.id = o.id_area ;
```

Annexe 14 - Tests utilisation vue matérialisée observation_nbr

```
-- Agrégation avec utilisation de la table observation_nbr
-- 12, 2093, 1478  M1  -- 64  row(s) fetched - 0,007s (0,002s fetch), on
2024-03-28 at 11:09:28
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,010s (0,004s fetch), on
2024-03-28 at 11:10:04
-- 10, 523, 369    M5  -- 47  row(s) fetched - 0,008s (0,002s fetch), on
```

```
2024-03-28 at 11:08:21
-- 9, 261, 184     M5  -- 118 row(s) fetched - 0,011s (0,004s fetch), on
2024-03-28 at 11:08:00
-- 8, 131, 92      M10 -- 128 row(s) fetched - 0,010s (0,003s fetch), on
2024-03-28 at 11:06:01
-- 7, 65, 46       M10 -- 187 row(s) fetched - 0,012s (0,003s fetch), on
2024-03-28 at 11:06:23
-- 6, 32, 23       M10 -- 219 row(s) fetched - 0,015s (0,006s fetch), on
2024-03-28 at 11:05:05
-- 5, 16, 11       M10 -- 819 row(s) fetched - 0,036s (0,016s fetch), on
2024-03-28 at 11:05:27
WITH tile AS (
    SELECT
        'M1' AS type_code,
        ST_TileEnvelope(11, 1046, 739) AS envelope -- SRID 3857
),
bounds AS (
    SELECT
        t.type_code,
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        l.id_area AS id,
        l.geom,
        l.area_code AS code,
        b.type_code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom_4326 && b.envelope_4326
        AND l.id_type = ref_geo.get_id_area_type(b.type_code)
)
SELECT
    ST_Transform(a.geom, 3857),
    a.code,
    o.obs_nbr
FROM gn_synthese.observation_nbr AS o
    JOIN areas AS a
        ON (a.id = o.id_area AND a.type_code = o.type_code) ;
```

Annexe 15 - Création d'une table cor_area_synthese partitionnée

Création d'une table partitionnée nativement sur le code du type de zone géo reliée à une observation de la synthese :

```
DROP TABLE IF EXISTS gn_synthese.cor_area_synthese_partitioned ;
```

```
CREATE TABLE gn_synthese.cor_area_synthese_partitioned (
    id_synthese int4 NOT NULL,
    id_area int4 NOT NULL,
    area_type_code VARCHAR(25) DEFAULT NULL::CHARACTER VARYING NULL,
    CONSTRAINT pk_casp PRIMARY KEY (id_synthese, id_area, area_type_code),
    CONSTRAINT fk_casp_id_area FOREIGN KEY (id_area) REFERENCES
ref_geo.l_areas(id_area) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_casp_id_synthese FOREIGN KEY (id_synthese) REFERENCES
gn_synthese.synthese(id_synthese) ON DELETE CASCADE ON UPDATE CASCADE
) PARTITION BY list(area_type_code);

    CREATE INDEX idx_casp_id_area ON
gn_synthese.cor_area_synthese_partitioned USING btree (id_area);
    CREATE INDEX idx_casp_area_type_code_id_area ON
gn_synthese.cor_area_synthese_partitioned USING btree (area_type_code,
id_area);

    CREATE TABLE gn_synthese.cor_area_synthese_m10 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('M10');

    CREATE TABLE gn_synthese.cor_area_synthese_m5 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('M5');

    CREATE TABLE gn_synthese.cor_area_synthese_m1 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('M1');

    CREATE TABLE gn_synthese.cor_area_synthese_com PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('COM');

    CREATE TABLE gn_synthese.cor_area_synthese_dep PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('DEP');

    CREATE TABLE gn_synthese.cor_area_synthese_sinp PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('SINP');

    CREATE TABLE gn_synthese.cor_area_synthese_aa PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('AA');

    CREATE TABLE gn_synthese.cor_area_synthese_apb PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('APB');

    CREATE TABLE gn_synthese.cor_area_synthese_metr PARTITION OF
gn_synthese.cor_area_synthese_partitioned
```

```
        FOR VALUES IN ('METR');

    CREATE TABLE gn_synthese.cor_area_synthese_pnr PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('PNR');

    CREATE TABLE gn_synthese.cor_area_synthese_rbiol PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('RBIOL');

    CREATE TABLE gn_synthese.cor_area_synthese_rbios PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('RBIOS');

    CREATE TABLE gn_synthese.cor_area_synthese_reg PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('REG');

    CREATE TABLE gn_synthese.cor_area_synthese_ripn PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('RIPN');

    CREATE TABLE gn_synthese.cor_area_synthese_rncfs PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('RNCFS');

    CREATE TABLE gn_synthese.cor_area_synthese_rnn PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('RNN');

    CREATE TABLE gn_synthese.cor_area_synthese_rnr PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('RNR');

    CREATE TABLE gn_synthese.cor_area_synthese_scen PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('SCEN');

    CREATE TABLE gn_synthese.cor_area_synthese_scl PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('SCL');

    CREATE TABLE gn_synthese.cor_area_synthese_sic PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('SIC');

    CREATE TABLE gn_synthese.cor_area_synthese_sram PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('SRAM');
```

```
    CREATE TABLE gn_synthese.cor_area_synthese_zbiog PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('ZBIOG');

    CREATE TABLE gn_synthese.cor_area_synthese_zc PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('ZC');

    CREATE TABLE gn_synthese.cor_area_synthese_zico PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('ZICO');

    CREATE TABLE gn_synthese.cor_area_synthese_znieff1 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('ZNIEFF1');

    CREATE TABLE gn_synthese.cor_area_synthese_znieff2 PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('ZNIEFF2');

    CREATE TABLE gn_synthese.cor_area_synthese_zps PARTITION OF
gn_synthese.cor_area_synthese_partitioned
        FOR VALUES IN ('ZPS');

DROP TABLE IF EXISTS ref_geo.tmp_subdivided_areas ;

CREATE TABLE IF NOT EXISTS ref_geo.tmp_subdivided_areas AS
    SELECT
        random() AS gid,
        a.id_area AS area_id,
        bat.type_code,
        st_subdivide(a.geom, 250) AS geom
    FROM ref_geo.l_areas AS a
        JOIN ref_geo.bib_areas_types AS bat
            ON a.id_type = bat.id_type
    WHERE a."enable" = TRUE
        AND bat.type_code NOT IN ('M10', 'M5', 'M1') ;

      CREATE INDEX IF NOT EXISTS idx_tmp_subdivided_geom ON
ref_geo.tmp_subdivided_areas USING gist (geom);
      CREATE INDEX IF NOT EXISTS idx_tmp_subdivided_area_id ON
ref_geo.tmp_subdivided_areas USING btree(area_id) ;

TRUNCATE TABLE gn_synthese.cor_area_synthese_partitioned ;

INSERT INTO gn_synthese.cor_area_synthese_partitioned
    SELECT DISTINCT
        s.id_synthese,
        a.area_id,
        a.type_code
    FROM gn_synthese.synthese AS s
```

```
        JOIN ref_geo.tmp_subdivided_areas AS a
            ON public.st_intersects(s.the_geom_local, a.geom) ;

INSERT INTO gn_synthese.cor_area_synthese_partitioned
    SELECT
        s.id_synthese,
        a.id_area,
        bat.type_code
    FROM ref_geo.l_areas AS a
        JOIN ref_geo.bib_areas_types AS bat
            ON a.id_type = bat.id_type
        JOIN gn_synthese.synthese AS s
            ON (a.geom && s.the_geom_local) -- Postgis operator && :
https://postgis.net/docs/geometry_overlaps.html
    WHERE bat.type_code IN ('M10', 'M5', 'M1') ;
```

Annexe 16 - Infos sur table "cor_area_synthese" partitionnée

- Temps d'exécution du script précédent : **4h05mn**
- Nombre d'entrées dans la table `gn_synthese.synthese` : **23 523 902**

Détails des zones géo par types de la table `ref_geo.l_areas` :

```
SELECT
    bat.type_code,
    bat.type_name,
    COUNT(a.id_area) AS "geom count"
FROM ref_geo.bib_areas_types AS bat
    JOIN ref_geo.l_areas AS a
        ON a.id_type = bat.id_type
GROUP BY bat.type_code, bat.type_name
ORDER BY bat.type_code ;
```

```
|type_code|type_name                                          |geom count|
|---------|---------------------------------------------------|----------|
|AA       |Aire d'adhésion des Parcs nationaux                |3         |
|APB      |Aires de protection de biotope                     |184       |
|COM      |Communes                                           |4 095     |
|DEP      |Départements                                       |12        |
|M1       |Mailles1*1                                         |72 230    |
|M10      |Mailles10*10                                       |819       |
|M5       |Mailles5*5                                         |3 078     |
|METR     |Métropoles et Communautés de Communes              |1         |
|PNR      |Parcs naturels régionaux                           |9         |
|RBIOL    |Réserves biologiques                               |34        |
|RBIOS    |Réserves de biosphère                              |7         |
|REG      |Région                                             |1         |
|RIPN     |Réserves intégrales de parc national               |1         |
|RNCFS    |Réserves nationales de chasse et faune sauvage     |2         |
```

```
|RNN      |Réserves naturelles nationales                    |37        |
|RNR      |Réserves naturelles regionales                    |17        |
|SCEN     |Sites acquis des Conservatoires d'espaces naturels|236       |
|SCL      |Sites du Conservatoire du Littoral                |26        |
|SIC      |Natura 2000 - Sites d'importance communautaire    |240       |
|SINP     |Territoire SINP                                   |1         |
|SRAM     |Sites Ramsar                                      |3         |
|TERRITORY|Territoire                                        |1         |
|ZBIOG    |Zones biogéographiques                            |4         |
|ZC       |Coeurs des Parcs nationaux                        |3         |
|ZICO     |Zone d'importance pour la conservation des oiseaux|32        |
|ZNIEFF1  |ZNIEFF1                                           |3 388     |
|ZNIEFF2  |ZNIEFF2                                           |272       |
|ZPS      |Natura 2000 - Zones de protection spéciales       |56        |
```

Taille de la table `cor_area_synthese` par défaut :

```
SELECT
    'gn_synthese.cor_area_synthese' AS "table",
    pg_size_pretty(pg_relation_size('gn_synthese.cor_area_synthese')) AS
internal,
    pg_size_pretty(pg_table_size('gn_synthese.cor_area_synthese') -
pg_relation_size('gn_synthese.cor_area_synthese')) AS "external", -- toast
    pg_size_pretty(pg_indexes_size('gn_synthese.cor_area_synthese')) AS
"indexes",
    pg_size_pretty( pg_total_relation_size('gn_synthese.cor_area_synthese')
) AS total,
    COUNT(*) AS "rows"
FROM gn_synthese.cor_area_synthese ;
```

```
|table                        |internal|external|indexes|total|rows       |
|-----------------------------|--------|--------|-------|-----|-----------|
|gn_synthese.cor_area_synthese|29 GB   |8304 kB |26 GB  |55 GB|378 815 208|
```

Résumé de la place occupée par la table `cor_area_synthese` partitionnée :

```
SELECT
   pi.inhparent::regclass AS "Partitioned table name",
   SUM(psut.n_live_tup) AS "rows count",
   pg_size_pretty(SUM(pg_relation_size(psu.relid))) AS internal,
   pg_size_pretty(SUM(pg_table_size(psu.relid) -
pg_relation_size(psu.relid))) AS "external", -- toast
   pg_size_pretty(SUM(pg_indexes_size(psu.relid))) AS "indexes",
   pg_size_pretty(SUM(pg_total_relation_size(psu.relid))) AS total
FROM pg_catalog.pg_statio_user_tables AS psu
    JOIN pg_class AS pc
        ON psu.relname = pc.relname
    JOIN pg_database AS pd
        ON pc.relowner = pd.datdba
    JOIN pg_inherits AS pi
        ON pi.inhrelid = pc.oid
```

```
    JOIN pg_stat_user_tables AS psut
        ON psut.relid = psu.relid
WHERE pd.datname = 'gn2_default'
GROUP BY pi.inhparent
ORDER BY SUM(pg_total_relation_size(psu.relid)) DESC;
```

```
|Partitioned table name                   |rows count
|internal|external|indexes|total|
|-----------------------------------------|-----------|--------|--------|---
----|-----|
|gn_synthese.cor_area_synthese_partitioned|378 815 305 |16 GB   |5304 kB |20
GB  |36 GB|
```

Calcul de la place occupée par les partitions :

```
SELECT
    child.relname AS "partition",
    psut.n_live_tup AS "rows count",
    pg_size_pretty(pg_relation_size(concat(nmsp_child.nspname, '.',
child.relname))) AS internal,
    pg_size_pretty(pg_table_size(concat(nmsp_child.nspname, '.',
child.relname)) - pg_relation_size(concat(nmsp_child.nspname, '.',
child.relname))) AS external, -- toast
    pg_size_pretty(pg_indexes_size(concat(nmsp_child.nspname, '.',
child.relname))) AS "indexes",
    pg_size_pretty(pg_total_relation_size(child.oid)) AS total
FROM pg_inherits
    JOIN pg_class AS parent
        ON pg_inherits.inhparent = parent.oid
    JOIN pg_class AS child
        ON pg_inherits.inhrelid   = child.oid
    JOIN pg_namespace AS nmsp_child
        ON nmsp_child.oid   = child.relnamespace
    JOIN pg_stat_user_tables AS psut
        ON psut.relid = child.oid
WHERE parent.relname = 'cor_area_synthese_partitioned'
ORDER BY child.relname ;
```

```
|partition                  |rows count |internal|external|indexes |total   |
|---------------------------|-----------|--------|--------|--------|-------|
|cor_area_synthese_aa       |341 435    |14 MB   |32 kB   |17 MB   |31 MB  |
|cor_area_synthese_apb      |934 481    |39 MB   |40 kB   |50 MB   |90 MB  |
|cor_area_synthese_com      |32 497 851 |1372 MB |416 kB  |1703 MB |3076 MB|
|cor_area_synthese_dep      |23 819 618 |1006 MB |304 kB  |1223 MB |2229 MB|
|cor_area_synthese_m1       |114 355 715|4829 MB |1384 kB |6623 MB |11 GB  |
|cor_area_synthese_m10      |28 629 845 |1209 MB |368 kB  |1505 MB |2714 MB|
|cor_area_synthese_m5       |33 117 680 |1399 MB |416 kB  |1759 MB |3158 MB|
|cor_area_synthese_metr     |265 394    |11 MB   |32 kB   |14 MB   |25 MB  |
|cor_area_synthese_others   |930 895    |39 MB   |40 kB   |48 MB   |87 MB  |
|cor_area_synthese_pnr      |4 944 797  |209 MB  |80 kB   |256 MB  |465 MB |
```

```
|cor_area_synthese_rbiol    |120 425    |5208 kB |32 kB   |6464 kB |11 MB  |
|cor_area_synthese_reg      |23 495 584 |992 MB  |304 kB  |1196 MB |2188 MB|
|cor_area_synthese_ripn     |13 221     |576 kB  |32 kB   |768 kB  |1376 kB|
|cor_area_synthese_rncfs    |41 321     |1792 kB |32 kB   |2200 kB |4024 kB|
|cor_area_synthese_rnn      |728 092    |31 MB   |32 kB   |37 MB   |68 MB  |
|cor_area_synthese_rnr      |190 592    |8248 kB |32 kB   |10032 kB|18 MB  |
|cor_area_synthese_scen     |812 249    |34 MB   |40 kB   |42 MB   |76 MB  |
|cor_area_synthese_scl      |122 572    |5304 kB |32 kB   |6440 kB |12 MB  |
|cor_area_synthese_sic      |6 033 103  |255 MB  |96 kB   |311 MB  |565 MB |
|cor_area_synthese_sinp     |23 495 584 |992 MB  |304 kB  |1203 MB |2196 MB|
|cor_area_synthese_sram     |224 062    |9696 kB |32 kB   |11 MB   |21 MB  |
|cor_area_synthese_territory|23 495 584 |1169 MB |352 kB  |1474 MB |2644 MB|
|cor_area_synthese_zbiog    |23 552 681 |995 MB  |304 kB  |1200 MB |2195 MB|
|cor_area_synthese_zc       |274 089    |12 MB   |32 kB   |14 MB   |25 MB  |
|cor_area_synthese_zico     |3 982 270  |168 MB  |72 kB   |202 MB  |370 MB |
|cor_area_synthese_znieff1  |13 676 029 |578 MB  |192 kB  |721 MB  |1299 MB|
|cor_area_synthese_znieff2  |15 162 334 |640 MB  |208 kB  |776 MB  |1416 MB|
|cor_area_synthese_zps      |3 557 802  |150 MB  |64 kB   |184 MB  |334 MB |
```

Annexe 17 - Test d'utilisation d'une table "cor_area_synthese" partitionnée

- Explain : https://explain.dalibo.com/plan/71eh98d2e11fcceb

```
-- Agrégation avec champ "area_type_code" dans "cor_area_synthese_partioned"
avec idx btree(area_type_code, id_area)
-- 12, 2093, 1478  M1  -- 64 row(s)  fetched - 0,102s (0,001s fetch), on
2024-03-29 at 16:10:11
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,229s (0,002s fetch), on
2024-03-29 at 16:10:35
-- 10, 523, 369    M5  -- 47  row(s) fetched - 0,683s (0,001s fetch), on
2024-03-29 at 16:11:51
-- 9, 261, 184     M5  -- 118 row(s) fetched - 1s (0,001s fetch), on
2024-03-29 at 16:12:14
-- 9, 261, 184     M10 -- 35  row(s) fetched - 2s, on 2024-03-29 at 16:12:39
-- 8, 131, 92      M10 -- 128 row(s) fetched - 6s (0,002s fetch), on
2024-03-29 at 16:13:19
-- 7, 65, 46       M10 -- 187 row(s) fetched - 9s (0,002s fetch), on
2024-03-29 at 15:54:10
-- 6, 32, 23       M10 -- 219 row(s) fetched - 10s (0,003s fetch), on
2024-03-29 at 15:55:55
-- 5, 16, 11       M10 -- 819 row(s) fetched - 55s (0,009s fetch), on
2024-03-29 at 15:57:08
WITH tile AS (
    SELECT
        'M10' AS type_code,
        ST_TileEnvelope(8, 131, 92) AS envelope
),
bounds AS (
    SELECT
        t.type_code,
```

```
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        l.id_area AS id,
        l.geom,
        l.area_code AS code,
        b.type_code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom_4326 && b.envelope_4326
        AND l.id_type = ref_geo.get_id_area_type(b.type_code)
),
observations AS (
    SELECT
        a.geom,
        a.code,
        COUNT(s.id_synthese) AS nbr
    FROM areas AS a
        JOIN gn_synthese.cor_area_synthese_partitioned AS casp
            ON (casp.id_area = a.id AND casp.area_type_code = a.type_code)
        JOIN gn_synthese.synthese AS s
            ON casp.id_synthese = s.id_synthese
    GROUP BY a.geom, a.code
)
SELECT
    ST_Transform(o.geom, 3857) AS geom,
    o.nbr,
    o.code
FROM observations AS o;
```

Annexe 18 - Test utilisation table "cor_area_synthese" sans lien vers la "synthese"

- Explain : https://explain.dalibo.com/plan/9ddfc18gfc61g83b

```
-- Agrégation avec champ "area_type_code" dans "cor_area_synthese_partioned"
avec idx btree(area_type_code, id_area) sans relation vers "synthese"
-- 12, 2093, 1478  M1  -- 64  row(s) fetched - 0,035s (0,001s fetch), on
2024-03-29 at 17:06:16
-- 11, 1046, 739   M1  -- 204 row(s) fetched - 0,071s (0,002s fetch), on
2024-03-29 at 17:05:58
-- 10, 523, 369    M5  -- 47  row(s) fetched - 0,188s (0,001s fetch), on
2024-03-29 at 17:05:44
-- 9, 261, 184     M5  -- 118 row(s) fetched - 0,365s (0,002s fetch), on
2024-03-29 at 17:05:27
-- 9, 261, 184     M10 -- 35  row(s) fetched - 0,844s, on 2024-03-29 at
17:05:12
```

```
-- 8, 131, 92      M10 -- 128 row(s) fetched - 1s (0,002s fetch), on
2024-03-29 at 17:04:43
-- 7, 65, 46       M10 -- 187 row(s) fetched - 2s (0,002s fetch), on
2024-03-29 at 17:07:44
-- 6, 32, 23       M10 -- 219 row(s) fetched - 2s (0,003s fetch), on
2024-03-29 at 17:07:58
-- 5, 16, 11       M10 -- 819 row(s) fetched - 10s (0,009s fetch), on
2024-03-29 at 17:08:35
-- EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON)
WITH tile AS (
    SELECT
        'M10' AS type_code,
        ST_TileEnvelope(8, 131, 92) AS envelope
),
bounds AS (
    SELECT
        t.type_code,
        t.envelope,
        ST_Transform(t.envelope, 4326) AS envelope_4326
     FROM tile AS t
),
areas AS (
    SELECT
        l.id_area AS id,
        l.geom,
        l.area_code AS code,
        b.type_code
    FROM ref_geo.l_areas AS l, bounds AS b
    WHERE l.geom_4326 && b.envelope_4326
        AND l.id_type = ref_geo.get_id_area_type(b.type_code)
),
observations AS (
    SELECT
        a.geom,
        a.code,
        COUNT(casp.id_synthese) AS nbr
    FROM areas AS a
        JOIN gn_synthese.cor_area_synthese_partitioned AS casp
            ON (casp.id_area = a.id AND casp.area_type_code = a.type_code)
    GROUP BY a.geom, a.code
)
SELECT
    ST_Transform(o.geom, 3857) AS geom,
    o.nbr,
    o.code
FROM observations AS o;
```

Annexe 19 - Test requêtes Synthese uniquement sur VM

v_synthese_for_webapp

```
CREATE EXTENSION IF NOT EXISTS intarray;


DROP TABLE IF EXISTS ref_geo.areas_subdivided ;

CREATE TABLE IF NOT EXISTS ref_geo.areas_subdivided AS
    SELECT
        random() AS gid,
        g.area_id,
        g.type_code,
        g.geom
    FROM (
        SELECT
            a.id_area AS area_id,
            bat.type_code,
            st_subdivide(a.geom_4326, 250) AS geom
        FROM ref_geo.l_areas AS a
            JOIN ref_geo.bib_areas_types AS bat
                ON a.id_type = bat.id_type
        WHERE a."enable" = TRUE
            AND a.id_type NOT IN (
                ref_geo.get_id_area_type('M50'),
                ref_geo.get_id_area_type('M20'),
                ref_geo.get_id_area_type('M10'),
                ref_geo.get_id_area_type('M5'),
                ref_geo.get_id_area_type('M1')
            )
        UNION
        SELECT
            a.id_area AS area_id,
            bat.type_code,
            a.geom_4326 AS geom
        FROM ref_geo.l_areas AS a
            JOIN ref_geo.bib_areas_types AS bat
                ON a.id_type = bat.id_type
        WHERE a."enable" = TRUE
            AND a.id_type IN (
                ref_geo.get_id_area_type('M50'),
                ref_geo.get_id_area_type('M20'),
                ref_geo.get_id_area_type('M10'),
                ref_geo.get_id_area_type('M5'),
                ref_geo.get_id_area_type('M1')
            )
    ) AS g
;
CREATE INDEX IF NOT EXISTS idx_areas_subdivided_geom ON
ref_geo.areas_subdivided USING gist (geom);
```

```
CREATE INDEX IF NOT EXISTS idx_areas_subdivided_area_id ON
ref_geo.areas_subdivided USING btree(area_id) ;
CREATE INDEX IF NOT EXISTS idx_areas_subdivided_type_code ON
ref_geo.areas_subdivided USING btree(type_code) ;


DROP MATERIALIZED VIEW IF EXISTS gn_synthese.v_synthese_for_web_app_full ;

CREATE MATERIALIZED VIEW gn_synthese.v_synthese_for_web_app_full AS
    SELECT
        s.id_synthese,
        s.unique_id_sinp,
        s.unique_id_sinp_grp,
        s.id_source,
        s.entity_source_pk_value,
        s.count_min,
        s.count_max,
        s.nom_cite,
        s.meta_v_taxref,
        s.sample_number_proof,
        s.digital_proof,
        s.non_digital_proof,
        s.altitude_min,
        s.altitude_max,
        s.depth_min,
        s.depth_max,
        s.place_name,
        s."precision",
        s.the_geom_4326,
        st_asgeojson(s.the_geom_4326)::jsonb  AS st_asgeojson,
        -- ADD blurring geom
        s.date_min,
        s.date_max,
        s.validator,
        s.validation_comment,
        s.observers,
        s.id_digitiser,
        s.determiner,
        s.comment_context,
        s.comment_description,
        s.meta_validation_date,
        s.meta_create_date,
        s.meta_update_date,
        s.last_action,
        d.id_dataset,
        d.dataset_name,
        d.id_acquisition_framework,
        (SELECT array_agg(DISTINCT cda.id_organism) FROM
gn_meta.cor_dataset_actor AS cda WHERE cda.id_dataset = d.id_dataset AND
cda.id_organism IS NOT NULL) AS organisms,
        s.id_nomenclature_geo_object_nature,
```

```
        s.id_nomenclature_info_geo_type,
        s.id_nomenclature_grp_typ,
        s.grp_method,
        s.id_nomenclature_obs_technique,
        s.id_nomenclature_bio_status,
        s.id_nomenclature_bio_condition,
        s.id_nomenclature_naturalness,
        s.id_nomenclature_exist_proof,
        s.id_nomenclature_valid_status,
        s.id_nomenclature_diffusion_level,
        s.id_nomenclature_life_stage,
        s.id_nomenclature_sex,
        s.id_nomenclature_obj_count,
        s.id_nomenclature_type_count,
        s.id_nomenclature_sensitivity,
        s.id_nomenclature_observation_status,
        s.id_nomenclature_blurring,
        s.id_nomenclature_source_status,
        s.id_nomenclature_determination_method,
        s.id_nomenclature_behaviour,
        s.reference_biblio,
        sources.name_source,
        sources.url_source,
        t.cd_nom,
        t.cd_ref,
        t.nom_valide,
        t.lb_nom,
        t.nom_vern,
        s.id_module,
        t.group1_inpn,
        t.group2_inpn,
        t.group3_inpn,
        -- TODO : use blurring geom instead
        (SELECT a.geom FROM ref_geo.areas_subdivided AS a WHERE
s.the_geom_point && a.geom AND a.type_code = 'M5' LIMIT 1) AS m5,
        (SELECT a.geom FROM ref_geo.areas_subdivided AS a WHERE
s.the_geom_point && a.geom AND a.type_code = 'M10' LIMIT 1) AS m10,
        (SELECT array_agg(DISTINCT a.area_id) FROM ref_geo.areas_subdivided
AS a WHERE st_intersects(s.the_geom_4326, a.geom) AND a.type_code = 'COM')
AS municipalities,
        (SELECT array_agg(DISTINCT a.area_id) FROM ref_geo.areas_subdivided
AS a WHERE st_intersects(s.the_geom_4326, a.geom) AND a.type_code = 'DEP')
AS departements
    FROM gn_synthese.synthese AS s
        JOIN taxonomie.taxref AS t
            ON t.cd_nom = s.cd_nom
        JOIN gn_meta.t_datasets AS d
            ON d.id_dataset = s.id_dataset
        JOIN gn_synthese.t_sources AS sources
            ON sources.id_source = s.id_source
```

```
    WHERE s.the_geom_4326 IS NOT NULL
        AND s.the_geom_4326 != ''
    ORDER BY s.id_synthese
    --LIMIT 100000
    --OFFSET 0
;

CREATE UNIQUE INDEX idx_sfwa_id_synthese_cd_nom ON
gn_synthese.v_synthese_for_web_app_full USING btree (id_synthese) INCLUDE
(cd_nom);
CREATE INDEX idx_sfwa_altitude_max ON
gn_synthese.v_synthese_for_web_app_full USING btree (altitude_max);
CREATE INDEX idx_sfwa_altitude_min ON
gn_synthese.v_synthese_for_web_app_full USING btree (altitude_min);
CREATE INDEX idx_sfwa_cd_nom ON gn_synthese.v_synthese_for_web_app_full
USING btree (cd_nom);
CREATE INDEX idx_sfwa_date_max ON gn_synthese.v_synthese_for_web_app_full
USING btree (date_max DESC);
CREATE INDEX idx_sfwa_date_min ON gn_synthese.v_synthese_for_web_app_full
USING btree (date_min DESC);
CREATE INDEX idx_sfwa_id_dataset ON gn_synthese.v_synthese_for_web_app_full
USING btree (id_dataset);
CREATE INDEX idx_sfwa_id_sources ON gn_synthese.v_synthese_for_web_app_full
USING btree (id_source);
CREATE INDEX idx_sfwa_the_geom_4326 ON
gn_synthese.v_synthese_for_web_app_full USING spgist (the_geom_4326);

CREATE INDEX idx_sfwa_geojson ON gn_synthese.v_synthese_for_web_app_full
USING gist (st_asgeojson);
CREATE INDEX idx_sfwa_observers ON gn_synthese.v_synthese_for_web_app_full
USING btree (observers);
CREATE INDEX idx_sfwa_id_acquisition_framework ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_acquisition_framework);
CREATE INDEX idx_sfwa_organisms ON gn_synthese.v_synthese_for_web_app_full
USING gin (organisms);
CREATE INDEX idx_sfwa_municipalities ON
gn_synthese.v_synthese_for_web_app_full USING gin (municipalities);
CREATE INDEX idx_sfwa_departements ON
gn_synthese.v_synthese_for_web_app_full USING gin (departements);
--CREATE INDEX idx_sfwa_m5 ON gn_synthese.v_synthese_for_web_app_full USING
gist (m5);
--CREATE INDEX idx_sfwa_m10 ON gn_synthese.v_synthese_for_web_app_full USING
gist (m10);

CREATE INDEX idx_sfwa_id_digitiser ON
gn_synthese.v_synthese_for_web_app_full USING btree (id_digitiser);
CREATE INDEX idx_sfwa_id_module ON gn_synthese.v_synthese_for_web_app_full
USING btree (id_module);
CREATE INDEX idx_sfwa_id_nomenclature_bio_condition ON
gn_synthese.v_synthese_for_web_app_full USING btree
```

```
(id_nomenclature_bio_condition);
CREATE INDEX idx_sfwa_id_nomenclature_bio_status ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_bio_status);
-- CREATE INDEX idx_sfwa_id_nomenclature_biogeo_status ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_biogeo_status);
CREATE INDEX idx_sfwa_id_nomenclature_blurring ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_blurring);
CREATE INDEX idx_sfwa_id_nomenclature_determination_method ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_determination_method);
CREATE INDEX idx_sfwa_id_nomenclature_diffusion_level ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_diffusion_level);
CREATE INDEX idx_sfwa_id_nomenclature_exist_proof ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_exist_proof);
CREATE INDEX idx_sfwa_id_nomenclature_geo_object_nature ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_geo_object_nature);
-- CREATE INDEX idx_sfwa_id_nomenclature_id_nomenclature_grp_typ ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_id_nomenclature_grp_typ);
CREATE INDEX idx_sfwa_id_nomenclature_info_geo_type ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_info_geo_type);
CREATE INDEX idx_sfwa_id_nomenclature_life_stage ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_life_stage);
CREATE INDEX idx_sfwa_id_nomenclature_obj_count ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_obj_count);
CREATE INDEX idx_sfwa_id_nomenclature_obs_technique ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_obs_technique);
CREATE INDEX idx_sfwa_id_nomenclature_observation_status ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_observation_status);
CREATE INDEX idx_sfwa_id_nomenclature_sensitivity ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_sensitivity);
CREATE INDEX idx_sfwa_id_nomenclature_sex ON
gn_synthese.v_synthese_for_web_app_full USING btree (id_nomenclature_sex);
CREATE INDEX idx_sfwa_id_nomenclature_source_status ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_source_status);
CREATE INDEX idx_sfwa_id_nomenclature_type_count ON
gn_synthese.v_synthese_for_web_app_full USING btree
```

```
(id_nomenclature_type_count);
CREATE INDEX idx_sfwa_id_nomenclature_valid_status ON
gn_synthese.v_synthese_for_web_app_full USING btree
(id_nomenclature_valid_status);
CREATE INDEX idx_sfwaf2_geojson ON gn_synthese.v_synthese_for_web_app_full
USING gin ("st_asgeojson");
```

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

Permanent link:
http://sinp-wiki.cbn-alpin.fr/fonctionnalites/geonature/synthese-amelioration-performance-test?rev=1719313829

Last update: **2024/06/25 11:10**