Import multi-destination

Identification des évolutions à apporter au module d'import actuel pour supporter d'autres destination que la synthèse. Réflexion menée avec pour destination typique Occtax & Occhab.

Évolutions principales

- Choix de la destination
- Données structurées en plusieurs entitées contrairement à la synthèse (*e.g.* Station / OccurenceHabitat dans le cas d'Occhab).

Hypothèses

- Malgré la présence de plusieurs entitées pour certaines destinations, on suppose l'utilisateur fournir un unique fichier. Par exemple pour Occtax, ce fichier pourra contenir des Stations, des OccurrenceHabitats, ou les deux. Pour chaque ligne, il sera déterminé si celle-ci contient une station, un habitat ou les deux.

Tradeoff

- Les permissions d'import dans une destination découle des permissions sur l'action C du module de destination
  - On a un unique module `IMPORT` en base
  - dont les permissions servent uniquement à activer ou non le module
  - et dont les permissions sur l'objet `MAPPINGS` ne sont pas contextualisées avec la destination.
  - On a alors une entrée unique « Imports » dans le menu GeoNature (mais des évolutions pourrait permettre assez facilement d'avoir des entrées multiples « Import Synthèse », « Import Occtax », ... l'idéal serait certainement de faire des sous-menus – cela servirait également aux protocoles du module monitoring).
- Les permissions d'import dans une destination peuvent être spécifier de manière indépendante des permissions de l'action C sur le module de destination
  - Le module d'import est dupliqué à la manière d'Occtax : `IMPORT_SYNTHESE`, `IMPORT_OCCTAX`, ...
  - Les permissions sur l'objet `MAPPINGS` peuvent varier suivant la destination
  - On a une entrée pour chaque destination dans le menu GeoNature (peu envisageable d'avoir qu'une entrée, mais très envisageable de les regrouper pour en faire des sous-menus comme imaginé précédemment).

Je penche pour la solution 1, l'intêret d'avoir une permission d'import différente de la permission C du module de destination me semble assez faible, et également plus complexe puisqu'il faut créer de multiple modules (l'intêret de contextualiser la gestion des mappings est pour le coup lui vraiment superflu). L'hypothèse de ce choix est faite dans la suite de ce document.

Évolutions générales

- Ajouter une table `bib_destinations` listant les destinations. Colonnes :
  - `code`
  - `label`
  - `id_module` (FK)
  - `table_name` (nom de la table pour les données transitoires, voir plus bas)

  Peut-être utiliser directement le code / label du module ? Le comportement actuel du module d'import donnerait lieu à une destination « synthèse » associé au module homonyme.
- L'ensemble des routes du backend seront préfixé par le code de la destination. Comme dans Occtax, on pourra utiliser un hook `@url_value_preprocessor` afin de définir la destination dans `g.destination`.
- Pour les destinations étant des modules externes, on tâchera d'importer les modèles dont on a besoin uniquement si le module en question est installé. Ainsi, le module d'import pourra être utilisé même s'il gère Occhab mais que ce dernier n'est pas installé.
- Rajouter une page de sélection de la destination en amont de la liste des imports (à l'image de la sélection du protocole dans monitoring)
  - La destination peut être spécifié dans l'URL pour permettre de faire des liens directe vers la liste d'imports d'une destination donnée

Évolutions du processus d'import

- Faire évoluer la pop-up permettant de sélectionner le JDD de destination :
  - Permettre de sélectionner en premier lieu la destination (à partir de `bib_destinations`)
    - Filtrer les destinations qui satisfont un C sur le module associé
    - Le choix de la destination n'apparait pas si la destination est déjà spécifié dans l'URL
      - Accès directe depuis un module de destination (*e.g.* en cliquant sur un lien « importer » dans Occtax)

· Accès en cliquant sur le « + » depuis la liste des imports d'une destination donnée

- Actuellement, les JDD proposées sont ceux qui sont :
  - ∗ associé au module d'import
  - ∗ accessible avec la portée de l'action C sur le module IMPORT

  Il faudrait proposer les JDD :
  - ∗ associé au module référencé par la destination
  - ∗ accessible avec la portée de l'action C sur le module référencé par la destination

  La liste des JDD proposées doit donc être chargé dynamiquement. À voir si le composant `pnx-datasets` le gère nativement ou doit évoluer.
- **RÉGRESSION** Une différence est qu'on ne gère plus les permissions d'importer via des permissions sur le modules d'import, mais directement via les permissions sur les modules de destination. Voir le tradeoff.
- Rajouter de pouvoir spécifier dans l'URL la destination, tous comme actuellement il est possible de référencer le JDD dans l'URL afin de pouvoir rediriger directement vers le module d'import depuis une page JDD du module méta-données par exemple.

- Rajouter une colonne `code_destination` à la table `t_imports`, et renseigner cette information lors de la création d'un import (au téléversement du fichier).
- À l'étape de configuration du fichier, on pourra imaginer que certains paramètres dépendent de la destination. Pour le moment, il n'y pas de modifications à faire (le SRID est nécessaire pour Synthèse, Occtax & Occhab).
- Rajouter une colonne `code_destination` (FK) aux tables `t_fieldmappings` et `t_contentmappings`.
- Rajouter une table `bib_entities` avec une colonne `label` et une colonne `code_destination` (FK). Rajouter une colonne `id_entity` (FK) à la table `bib_fields`. On a ainsi l'arborescence suivante : Destinations > Entitées > Thèmes > Champs.
  - La contrainte d'unicité sur `bib_fields.name_field` devra porter sur le couple `bib_fields.name_field` / `bib_categories.code_destination`.
  - L'étape de correspondances des champs propose uniquement les champs associé à la destination.
  - Les thèmes servent actuellement à regrouper les champs par thématique (*e.g.* « Informations géographique ») ; les entitées serviront à regrouper ces dernières dans des onglets distincts (*e.g.* « Station » et « Occurence d'habitats » pour Occhab).

  - Les mappings sont également contextualisés avec la destination. Cela concerne également la validation qui est faite du contenu des mappings. En revanche, la structure JSON des mappings n'a pas besoin d'évoluer.

- Lors de la validation de la correspondance des champs, les données sont chargé dans une table transitoire.
  - Cette table s'appelle `t_imports_synthese` dans le cas de la synthèse, et est associé au modèle `ImportSyntheseData`.
  - Rajouter une table transitoire pour chaque destination (*i.e.* `t_imports_occhab` et `t_imports_occtax`). Il s'agit d'une table à plat, comme le fichier CSV attendu, y compris pour les destinations à plusieurs entitées.
  - Référencer la table transitoire de destination dans une colonne de `bib_destinations`.
  - Supprimer le modèle `ImportSyntheseData` et le construire dynamiquement par réflexion à partir de la table transitoire.

  Ces deux derniers points permettent d'envisager l'import vers des protocoles que l'on rajoute dynamiquement uniquement par une configuration en base.

- L'étape de correspondance des nomenclatures ne nécessite pas d'évolution majeure
  - Les mappings doivent être contextualisés avec la destination
  - Les champs à nomenclature sont identifié par la colonne `menomique` dans la table `bib_fields` (pas de changement)
  - On détermine les valeurs sources à partir des données de la table transitoire (pas de changement, outre qu'il faut regarder dans la bonne table transitoire)

- Les contrôles sont lancés.
  - Les données sont chargé dans une dataframe qui sert à un premier lot de contrôle
    * remplacement des chaînes vides par `np.nan` (`None`)
    * dates min / max (spécifique synthèse)
    * champs obligatoire (générique)
    * typage des champs (générique, les types sont renseigné dans `bib_fields`)
    * géométries (spécifique synthèse avec la logique wkt vs x/y vs code département/commune/maille)
    * cohérence dénombrement (spécifique synthèse)
  - Puis les données sont réinséré en base et sont effectué des contrôles SQL (essentiellement spécifique à la synthèse)
  - Il faut donc adapter les contrôles à effectuer suivant la destination.
  - Pour les imports à plusieurs entitées :

        * Rajouter une colonne booléenne dans la table transitoire pour chaque entitée.
        * Passer à vrai cette colonne dès lorsqu'une colonne associé à l'entitée considéré n'est pas `NULL`
            · Par exemple, si la colonne `nom_station` est remplie, on suppose la ligne contenir une Station.
            · Attention aux colonnes partagé. Par exemple, l'UUID Station est une colonne Station **et** OccurenceHabitat. Une ligne peut contenir une OccurenceHabitat avec l'UUID Station spécifié, mais sans contenir de Station.
        * Le remplacement des chaînes vides par `NULL` doit préalablement avoir été effectué, idéalement en SQL.
        * Les contrôles pour une entitées données (*e.g.* champs obligatoire) s'applique uniquement sur les lignes d'intérêts (on ne remonte pas de champs obligatoire manquant pour des lignes contenant une OccurenceHabitat mais pas de Station).
        * Chaque entitée fille (*e.g.* OccurenceHabitat) doit, au choix :
            · Contenir l'UUID de l'entitée parente (*e.g.* l'UUID de la Station). On vérifiera alors que l'UUID exite en base *ou* dans le fichier fourni.
            · Contenir un ID d'entitée parente propre au fichier. Cette ID est perdu à la fin de l'import du fichier. Cette solution ne peut être utilisé si l'on souhaite importer les différentes entitées dans différents fichiers ; on utilisera nécessairement des UUID dans ce cas là.
            · Se trouver sur la même ligne que l'entitée mère (solution simple pour, par exemple, importer dans Occtax des observations avec un relevé / une occurence / un dénombrement).
    - Impacts à évaluer pour les protocoles monitorées. Il faut voir en quel mesure les contrôles peuvent être configurés dynamiquement à partir de la configuration monitoring.

- Puis les données sont insérées en base.
    - Des processus spécifiques sont à prévoir pour chaque destination, avec notamment l'import de chaque entitées (*e.g.* station puis habitats) mais cela reste très similaire à la synthèse donc évolution assez simple.
    - Impacts à évaluer pour les protocoles monitorées.

- L'import en base se termine par la récolte de quelques statistiques stoquées dans des colonnes spécifiques de la table `t_imports` :
    - Nombre d'observations insérées dans la colonne `import_count`
    - Nombre de taxons dans la colonne `taxa_count`

  À remplacer par une colonne HSTORE (clé / valeur) `result` afin de stoquer des informations spécifiques à chaque destination ?

Perspectives

Ici, le module d'import a conscience de l'existance de la synthèse, d'Occtax, d'Occhab ... Dans le futur, on pourra imaginer intégrer le module d'import à GeoNature, et que celui-ci n'est plus aucune connaissance des potentielles destinations, mais se contente de fournir une boite à outil permettant aux différents modules de se déclarer comme destination, en fournissant leur propre contrôles, etc.