**DRE**

**Données Répliquées en Etablissement**

Table des matières

[1 Le module DRE et ses concepts 4](#_Toc168929813)

[1.1 Les objectifs de DRE 4](#_Toc168929814)

[1.2 Pas de rétro compatibilité des formats entre versions 4](#_Toc168929815)

[1.3 Principe de fonctionnement 4](#_Toc168929816)

[2 Installation 6](#_Toc168929817)

[2.1 Pré requis administratif d’installation 6](#_Toc168929818)

[2.2 Configuration Pegase 6](#_Toc168929819)

[2.3 Pré requis technique d'installation 6](#_Toc168929820)

[2.4 Mode opératoire de mise à jour de la base DRE 6](#_Toc168929821)

[2.4.1 Mise à disposition des données (bdd PostgreSQL) 7](#_Toc168929822)

[2.4.2 Mise à disposition des données (bdd MongoDB) 7](#_Toc168929823)

[2.4.3 Fichier d’indication de la version de DRE 7](#_Toc168929825)

[2.4.4 Création de la base DRE en établissement 8](#_Toc168929826)

[2.5 Personnalisation de DRE 8](#_Toc168929827)

[3 Les données 9](#_Toc168929828)

[3.1 Les Bdd constituant DRE 9](#_Toc168929829)

[3.2 Modèle de données 9](#_Toc168929830)

[4 Utilisation de DRE 11](#_Toc168929831)

[4.1 Usage 11](#_Toc168929832)

[4.2 Outils pour accéder aux données 11](#_Toc168929833)

[4.3 Mise en commun de code et d’outils 11](#_Toc168929834)

[4.4 Utilisation de DRE pour de l'intégration applicative 11](#_Toc168929835)

[4.5 Support 12](#_Toc168929836)

[5 Annexes 13](#_Toc168929837)

[5.1 Liste des documents complémentaires et liens utiles 13](#_Toc168929838)

[5.2 Script de création de DRE en établissement 13](#_Toc168929839)

[5.2.1 Téléchargement des fichiers 13](#_Toc168929840)

[5.2.2 Principes du script 13](#_Toc168929841)

[5.2.3 Exemple de script 15](#_Toc168929842)

[5.2.1 Exemple de script SQL pour le chargement des données json (collections MongoDB) et la version Pegase 17](#_Toc168929843)

[5.3 Modèle de données ‘métier’ 21](#_Toc168929845)

[5.4 Modèle physique des données 21](#_Toc168929846)

[5.4.1 Le dictionnaire des données 21](#_Toc168929847)

[5.4.2 Les différences par rapport à la v21 21](#_Toc168929848)

[5.4.3 Extraction de la documentation embarquée 21](#_Toc168929849)

[5.5 Conseils pour l’écriture de requêtes 22](#_Toc168929850)

**Suivi des modifications du document :**

|  |  |  |
| --- | --- | --- |
| **Version de Pégase (\*)** | **Date de mise à jour** | **Modifications apportées** |
| V21 r01 | 19/07/2023 | Création du document, version initiale. |
| V21 r02 | 21/7/2023 | Ajout information téléchargement fichiers |
| V22 | 19/10/2023 | Mise à jour des modèles de données avec ceux de la v22  Fourniture d’un fichier listant les différences u modèle physique de données |
| V22 r01 | 24/11/2023 | Ajout du fichier de description de la version |
| V23 | 17/01/2023 | Mise à jour pour la version v23 |
| V24 | 30/04/2024 | Mise à jour pour la version v24 |
| V24-1 | 10/06/2024 | Ajout des données piste\_inscription  Configuration de certaines informations dans le module ADMIN |
|  |  |  |
|  |  |  |

# Le module DRE et ses concepts

Les objectifs de DRE

La mise à disposition de données de Pegase à des fins de pilotage opérationnel est un besoin identifié à l’origine du projet PC-SCOL. Une première réponse a été prototypée, ce que nous avons appelé la base de données POP.

Le besoin a été revu et élargi : il consiste à mise à disposition de toutes les données de Pégase avec un historique supérieur à 5 ans.

La solution répondant à ce besoin est nommée Données Répliquées en Etablissement (DRE).

En voici une description synthétique :

* Cette solution fournit une réplication complète des données présentes dans Pégase mais sans aucune transformation des bases de données des services (pour rappel, il y a dans Pegase une Bdd par service).
* La base de données résultat de cette réplication est hébergée dans l’infrastructure de l’établissement.
* La réplication est limitée aux bases de données Postgresql et quelques données issues de bdd MongoDB, ce qui permet de disposer de la presque totalité des données de Pegase.
* Les données sont rafraichies à J+1, en écrasant les données de la veille.
* Seul un ensemble de données est fourni. Les établissements seront en charge de mettre en place leur propre solution de requêtage et reporting (univers BO par exemple)

Pas de rétro compatibilité des formats entre versions

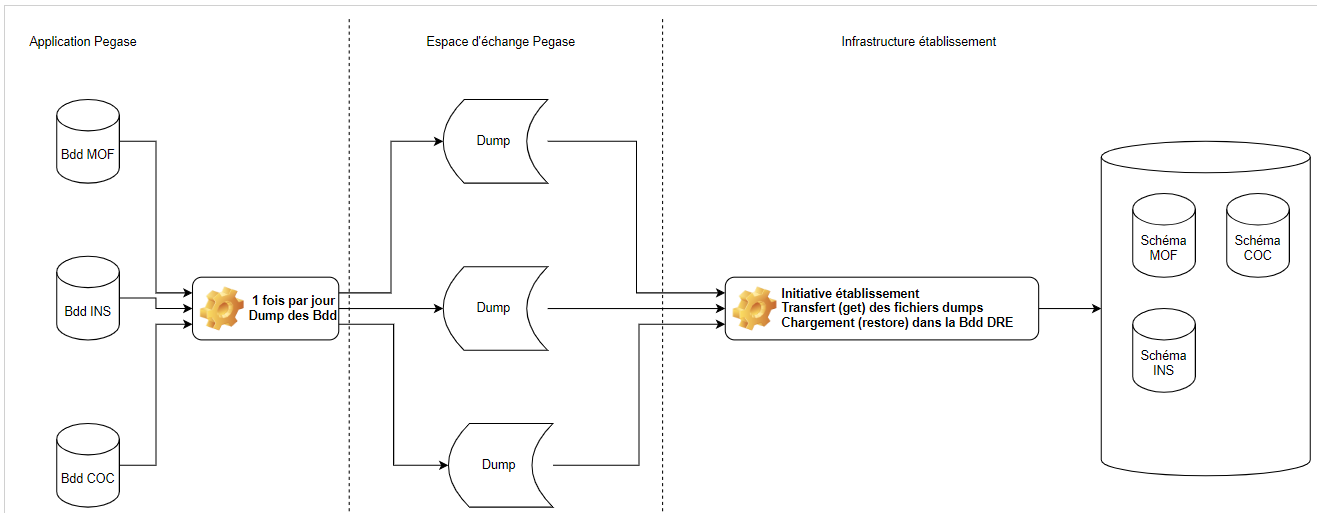
DRE étant une réplication des Bdd de Pegase, son format évolue au cours de versions de Pegase. De ce fait chaque nouvelle version s’accompagne de changements dans le modèle physique de données de DRE.

PC-SCOL fournit une liste des modifications.

PC-SCOL ne fournit aucun outil permettant de gérer la rétrocompatibilité des requêtes développées pas les établissements.

Principe de fonctionnement

Le schéma ci-dessous présente cette solution (seules 3 bases de données de services Pegase sont représentées ici) :



Chaque base de données PostgreSQL de PEGASE est exportée sous forme d’un fichier dump. Pour les Bdd Mongo DB, chaque collection mise à disposition est exportée sous forme de fichier JSON. Ce sont tous ces fichiers qui sont mis à disposition une fois par jour.

L’établissement lance depuis son infrastructure un script qui va transférer les fichiers dump et les restaurer dans la base DRE (c’est un annule et remplace).

Au final chaque Bdd Pegase se retrouve en tant qu’un schéma de la base de données DRE.

# Installation

Pré requis administratif d’installation

Etant donné le caractère sensible d’une partie des données contenues dans DRE, il faut formuler une demande écrite pour disposer des données et des accès, en nous indiquant une personne référente. Un formulaire est mis à votre disposition sur le serveur share de PC-SCOL : <https://forum.pc-scol.fr/uploads/short-url/dUeLsgrKZOPtQYEBc5jvbXkUc34.pdf> .

Les données seront fournies sans gestion d'autorisation, il est de la responsabilité de l'établissement de les gérer.

Configuration Pegase

Dans le module ADMIN de Pegase il est nécessaire de configurer la section relative à DRE (se référer à la documentation du module ADMIN).

y de renforcer le contrôle d’accès aux données

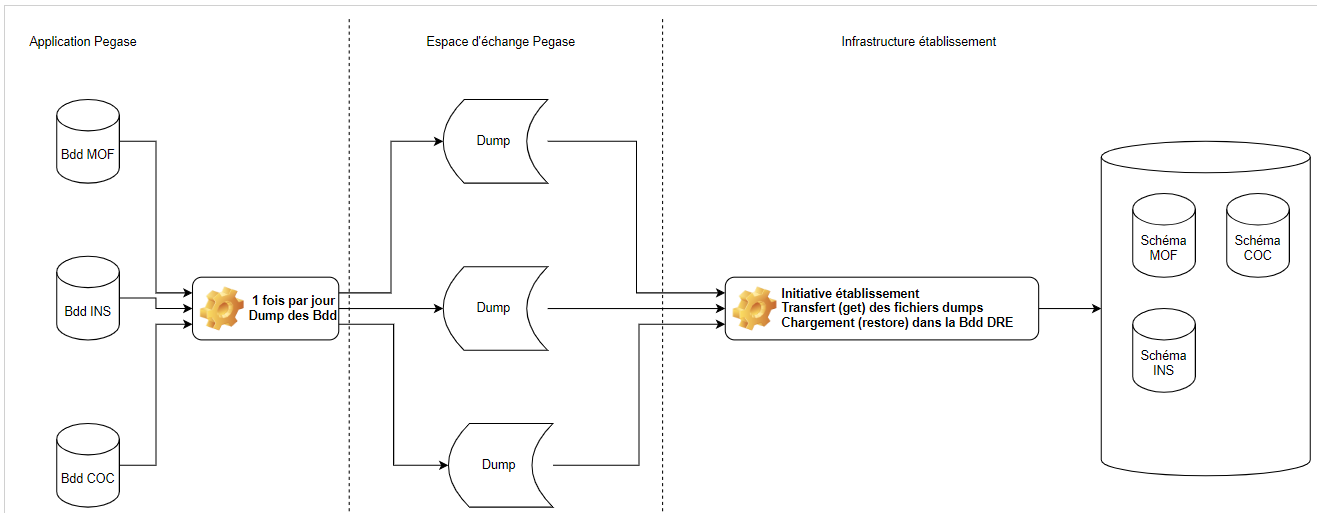
Pré requis technique d'installation

Pour pouvoir mettre en œuvre DRE dans votre infrastructure, voici ce qui est nécessaire :

* Avoir initialisé sur un serveur établissement dédié un cluster PostgreSQL 15 avec à l'intérieur une base de données vierge (futur réceptacle de tous les schémas répliqués),
* Avoir installé le paquet des outils clients PostgreSQL,
* Il est possible selon les affinités établissement d'utiliser la conteneurisation.

Mode opératoire de mise à jour de la base DRE

Pour rappel, le schéma ci-dessous décrit le mode opératoire qui va permettre de créer quotidiennement la base DRE dans l’infrastructure de l’établissement.



Les étapes qui s’enchainent et vont vous être détaillées dans les paragraphes suivants sont :

* Mise à disposition des données : Pegase produit automatiquement un extrait de ses données qui est déposé dans l’infrastructure PC-SCOL.
* Création de la base DRE en établissement : l’établissement lance un script qui va transférer les données mises à disposition et les charger dans sa base DRE locale,
* Personnalisation de DRE (option) : l’établissement peut exécuter des scripts permettant de compléter la base DRE (ajout d’utilisateurs, de droits, de vues, ….).

### Mise à disposition des données (bdd PostgreSQL)

Un dump (export de données dans un format PostgreSql) de chacune des Bdd Pegase mises à disposition pour DRE est créé tous les soirs dans un stockage dédié de l’infrastructure PC-SCOL. Cette extraction a lieu à 4h. Ceci se matérialise par plusieurs fichiers nommés :

<environnement>-<Date AAAAMMJJ>-<service>.bin

Soit par exemple pour le service CHC de l’UPHF : uphf-20230512-odf.bin

L’application Pegase (gestionnaire et web) reste entièrement disponible durant le dump.

L’espace de stockage sur lequel les fichiers seront mis à disposition est spécifique à chaque instance Pegase.

Ce mécanisme permet une mise à disposition des données à J+1 (donc pas de temps réel pour l'accès aux données via DRE).

### Mise à disposition des données (bdd MongoDB)

Dans le même traitement, les données MongoDB mises à disposition sont exportées sous forme d’un fichier json par collection, nommé :

<environnement>-<Date AAAAMMJJ>-<bdd>-<collection>.json

Soit par exemple pour la collection calendrier de la bdd piste\_inscription de l’UPHF :

uphf-20230512-piste\_inscriptioncalendrier.json

### Fichier d’indication de la version de DRE

En plus des fichiers de données décrits dans le paragraphe précédent, un fichier additionnel indique la version de DRE (donc la version de Pegase) à laquelle correspondent les fichiers.

Ceci peut être utile pour gérer des traitements différenciés selon la release de Pegase.

Le fichier se nomme

DRE\_VERSION\_YYYYMMDD

Il contient les informations de la version dans un format json dont voici un exemple

{ "majeure": "23", "mineure": "0", "patch": "0","prerelease": "feat-refactoCof.18"}

A titre d’exemple, il est possible, dans un script, de récupérer les valeurs via une commande grep ou avec un outil tel que jq :

$ jq '.majeure' DRE\_VERSION\_20231116

"23"

Il est également possible de charger les différentes valeurs dans une table PostgreSQL (cf. annexe).

### Création de la base DRE en établissement

Le script exécutant cette étape sera spécifique à chaque établissement. Nous vous fournissons en annexe les éléments qui vous permettront de le créer, ainsi qu’un exemple de script paramétrable. Les établissements auront l’initiative dans le choix de l'heure du lancement du script.

Le mode opératoire fourni permet d'importer tous les schémas (chaque schéma correspond à la base de données d’un service Pegase) dans une seule et même base de données. Toutes les données sont ainsi accessibles pour un utilisateur connecté avec les droits nécessaires et suffisants. Il suffit de préfixer dans les requêtes chaque table avec le nom du schéma auquel elle appartient.

exemple : select \* from schema\_gestion.apprenant;

Les informations utiles à la création du script et l’exemple sont fournis en annexe.

Personnalisation de DRE

Il n’y a aucune obligation de modifier le schéma de la Bdd DRE qui sera fourni par PC-SCOL.

Il est par contre possible de le faire (par exemple : ajout de vue, ajout de droits, …). En effet, une fois créée dans votre infrastructure, la base de données DRE sera entièrement gérée par l’établissement qui pourra donc la modifier à sa guise.

Attention : il est nécessaire de ré appliquer cette personnalisation après chaque exécution du script de création de DRE en établissement car la plupart des éléments sont écrasés et remplacés. La personnalisation peut être incluse dans le script quotidien.

Parmi les personnalisations possibles il est fortement recommandé de gérer les droits d’accès.

Des exemples de scripts sont fournis en annexe.

# Les données

Les Bdd constituant DRE

DRE regroupe dans une seule BDD les BDD de plusieurs services de Pegase. Chaque BDD source est disponible dans un schéma de la base DRE.

Les BDD Pegase sont fournies dans leur format d’origine, à l’exception de certaines tables techniques qui ne seront pas fournies.

Le tableau ci-dessous liste les bases de données PostGreSQL qui vont alimenter la base DRE.

|  |  |  |  |
| --- | --- | --- | --- |
| **Module Pegase** | **Service Pegase** | **Bdd Pegase** | **schema\_bdd** |
| CHC | chc-server | chc-db-postgres | schema-chc |
| COC | coc-server | coc-db-postgres | schema-coc |
| ADMISSION (inclus dans INS) | INS-candidature-server | ins-db-postgres | schema-inscription |
| INS (gestionnaire) | ins-gestion-server | ins-gestion-db-postgres | schema-gestion |
| ~~MOF~~ | ~~mof-server~~ | ~~mof-db-postgres~~ | ~~schema-mof~~ |
| ODF | odf-server | odf-db-postgres | schema-odf |
| PAI | pai-server | pai-db-postgres | schema-pai |
| REF | ref-server | ref-db-postgres | schema\_ref |

Le nom du schéma dans la bdd DRE est la colonne de droite.



Comme vous pouvez le noter, compte tenu des changements de gestion de l’offre de formation en v24, le schema-mof a disparu, et le schema-odf a été ajouté.

Pour les bases de données MongoDB, voici les collections qui sont mises à disposition :

|  |  |  |  |
| --- | --- | --- | --- |
| **Module Pegase** | **Service Pegase** | **Bdd Pegase** | **Collection** |
| Inscriptions en ligne | INS-piste server | piste\_inscription | calendrier |
| Inscriptions en ligne | INS-piste server | piste\_inscription | chemins |
| Inscriptions en ligne | INS-piste server | piste\_inscription | dossiers\_inscription |
| Inscriptions en ligne | INS-piste server | piste\_inscription | paiements |

Modèle de données

Les modèles de données sont fournis en annexe et dans des fichiers hors du présent document :

* Un modèle métier qui décrit les entités et leurs identifiants. Il se présente sous forme de schéma et d’un tableau des entités
* Un dictionnaire des données identifiant pour chacun des schémas les tables et colonnes, avec des commentaires descriptifs qui ne couvrent pas à ce jour l’ensemble du modèle.
* Un tableau précisant les évolutions de tables et colonnes depuis la version précédente.

Pour chaque entité métier, il est indiqué le schéma et la table qui est son stockage primaire. Une même entité pourra à partir de cette table être répliquée dans d’autres schéma avec tout ou partie de ses attributs (colonnes).

# Utilisation de DRE

## Usage

DRE est un des composants permettant d’accéder aux données de Pegase depuis l’extérieur de l’application. C’est un complément aux API et connecteurs, mais également à des outils d’extraction qui seront intégrés dans l’application Pegase.

DRE est orienté vers le reporting et les enquêtes, et plus largement toutes les requêtes. Il est particulièrement adapté :

* Aux extractions demandant des jointures complexes
* Aux extractions combinant des données issues de plusieurs modules de Pegase
* Aux extractions retournant des volumes de données importants

## Outils pour accéder aux données

PC-SCOL ne fournit aucun outil d’accès aux données de DRE.

Pour exécuter des requêtes, créer des rapports ou des tableaux de bord, vous pouvez utiliser tous les outils permettant d’accéder une base Postgresql. Par exemple Excel, BO, Jasper report, …

## Mise en commun de code et d’outils

Tous les établissements auront des données dans un format identique. Ceci facilite la réutilisation d’outils (scripts, code SQL, rapports et tableaux de bord, …) d’un établissement à l’autre avec une adaptation minimale.

Nous vous proposons d’utiliser le site GitHub du projet : <https://github.com/PC-Scol> afin de partager ou de co construire vos outils.

## Utilisation de DRE pour de l'intégration applicative

DRE mettant à votre disposition la plupart des données Pegase, rien n’empêche un établissement ou un éditeur d’utiliser DRE pour construire une interface entre Pegase et une autre application. Mais ceci n’est pas recommandé à l’exception de certains cas.

Voici un certain nombre de "bonnes pratiques" pour déterminer quand utiliser les API / connecteurs et quand utiliser DRE :

Pour rappel : la solution privilégiée d’interfaçage entre Pegase et les applications extérieures est l’utilisation des API, si possible en mode événementiel (exemple des connecteurs SINAPS et GRHUM).

Toutefois nous devons gérer les cas où une application n’est pas en mesure de communiquer via API. Dans ce cas, l’utilisation d’extractions issues de DRE peut s’avérer une alternative efficace. Ceci ne fonctionne pas pour mettre à jour des données Pegase.

Voici quelques exemples que nous entrevoyons :

* L’alimentation quotidienne d’un SiD (système d’information décisionnel) s’appuie en général sur des scripts ETL qui vont extraire des volumes de données importants dans la BDD de l’application source. Les API ne sont pas adaptées. L’alimentation des SID (SIROCCO ou STRENNES) s’appuiera donc sur DRE.
* Certaines applications demandent en entrée un fichier (en général au format csv) à intervalle régulier et ne disposent / disposeront pas de la capacité à appeler ou exposer des API. Dans ce cas, il est préférable de s’appuyer sur DRE plutôt que d’appeler de multiples API pour constituer un stockage local.

## Support

PC-SCOL s’engage sur la mise à disposition des dumps des Bdd dans l’espace dédié de chaque instance. Au-delà, les traitements relèvent de la responsabilité de l’établissement

PC-SCOL fournit une documentation de type dictionnaire de données, et la fera évoluer en fonction de tickets signalant des imprécisions.

# Annexes

## Liste des documents complémentaires et liens utiles

Documents de cartographie Pegase (comprend les modèles de données : <https://share.pc-scol.fr/smart-link/0bf9f89a-b274-4cf7-aacd-5cb8ef482a71/> (comprend un répertoire par version)

Espace DRE sur le Forum PC-SCOL : <https://forum.pc-scol.fr/t/base-de-donnees-relationnelle/2023/24>

Documentation PostgreSQL : <https://www.postgresql.org/docs/>

## Script de création de DRE en établissement

Ce paragraphe décrit les éléments qui permettent de développer le script établissement de transfert et restauration de la base DRE. Il donne également un exemple de script qui peut être utilisé comme point de départ.

### Téléchargement des fichiers

L’établissement dispose d’une interface web pour downloader les fichiers dump ou peut les accéder en ligne de commande (wget ou curl).

Pour une instance *mon-instance* l’adresse est au format <https://dre-dump.mon-instance.pc-scol.fr>

exemple : pour l’instance

partenaires

l’URL est

[https//dre-dump.partenaires.pc-scol.fr](https/dre-dump.partenaires.pc-scol.fr)

Une authentification basique login/pwd permet d’accéder le dump, ceci est disponible dans le fichier keypass de l’environnement.

### Principes du script

Pour un environnement donné, autant de dumps (PostgreSQL) que de modules à exporter sont fournis aux établissements. Chacun contient un export du schéma des données applicatives.

Le mode opératoire fourni permet d'importer tous les schémas dans une seule et même base de données. Toutes les données sont ainsi accessibles pour un utilisateur connecté avec les droits nécessaires et suffisants. Il suffit de préfixer dans les requêtes chaque table avec le nom du schéma auquel elle appartient, comme par exemple : select \* from schema\_gestion.apprenant;

Prérequis :

* Avoir initialisé sur un serveur établissement dédié un cluster PostgreSQL 15 avec à l'intérieur une base de données vierge (futur réceptacle de tous les schémas répliqués)
* Avoir installé le paquet des outils clients postgresql
* Il est possible selon les affinités établissement d'utiliser la conteneurisation

Protocole d'import

* + - Si un import a déjà été effectué et que la base n'a pas été purgée, nettoyée ou recréée, supprimer les schémas existants
      * Alternative 1, en environnement non-conteneurisé : **psql -d <nom\_database\_dre> -U <user\_database\_dre>** **-c 'DROP SCHEMA IF EXISTS <nom schema 1>, <nom schema 2>, ... CASCADE;'**  
        exemple : psql -d dre -U postgres -c 'DROP SCHEMA IF EXISTS schema\_chc, schema\_coc, schema\_gestion, schema\_mof, schema\_ref CASCADE;'
      * Alternative 2, en environnement conteneurisé: **docker exec -i <nom\_conteneur> psql -d <nom\_database\_dre> -U <user\_database\_dre> -c 'DROP SCHEMA IF EXISTS <nom schema 1>, <nom schema 2>, ... CASCADE;'**  
        exemple : docker exec -i dre-db-1 psql -d dre -U postgres -c 'DROP SCHEMA IF EXISTS schema\_chc, schema\_coc, schema\_gestion, schema\_mof, schema\_ref CASCADE;'
    - Pour chacun des dumps que l'on souhaite importer
      * Supprimer au préalable dans la base cible de l'import les extensions si elles existent (<=> si elles ont été importées lors d'un import précédent). Cela est nécessaire pour éviter les problèmes de cohabitation : 1 extension donnée ne peut être installée qu'une seule fois sur une base de données)
        + Alternative 1, en environnement non-conteneurisé: **psql -d <nom\_database\_dre> -U <user\_database\_dre>** **-c 'DROP EXTENSION IF EXISTS unaccent, pg\_trgm CASCADE;'**  
          exemple : psql -d dre -U postgres -c 'DROP EXTENSION IF EXISTS unaccent, pg\_trgm CASCADE;'
        + Alternative 2, en environnement conteneurisé : **docker exec -i <nom\_conteneur> psql -d <nom\_database\_dre> -U <user\_database\_dre> -c 'DROP EXTENSION IF EXISTS unaccent, pg\_trgm CASCADE;'**  
          exemple : docker exec -i dre-db-1 psql -d dre -U postgres -c 'DROP EXTENSION IF EXISTS unaccent, pg\_trgm CASCADE;'
      * Importer le dump
        + Alternative 1, en environnement non-conteneurisé : **pg\_restore -d <nom\_database\_dre> -U <user\_database\_dre> --no-owner --no-acl --clean --if-exists <nom\_dump>**  
          exemple : pg\_restore -d dre -U postgres --no-owner --no-acl --clean --if-exists mof-db-postgres.bin
        + Alternative 2, en environnement conteneurisé : **docker exec -i <nom\_conteneur> pg\_restore -d <nom\_database\_dre> -U <user\_database\_dre> --no-owner --no-acl --clean --if-exists < <nom\_dump>**  
          exemple : docker exec -i dre-db-1 pg\_restore -d dre -U postgres --no-owner --no-acl --clean --if-exists < mof-db-postgres.bin
      * Si :
        + Aucune occurrence de ERROR n'est apparue alors les dumps se sont importés correctement
        + Aucun message "pg\_restore: warning: errors ignored on restore: xxx" n'est apparu après chaque pg\_restore alors l'import s'est bien terminé
      * Pour les collections issues de MongoDB :
        + Le principe est de charger chaque collection dans une table dédiée (nous recommandons de créer un schéma pour chaque Bdd Mongo, mais ceci est au choix de l’établissement).
        + Dans chaque table, une colonne de type jsonb contiendra une ligne du fichier, donc un enregistrement de la collection.
        + PostgreSQL dispose dans son SQL de fonctions permettant d’extraire certains éléments du json.
        + Il est possible d’ajouter des colonnes non json pour y extraire des données de l’enregistrement json. Par exemple vous pouvez extraire des identifiants qui vous permettrons de faciliter les jointures avec d’autres tables.

### Exemple de script

Voici un script rudimentaire exemple implémentant ce protocole (pas de logs, pas de paramètres passables à l'appel du script, ...) :

|  |
| --- |
| #!/bin/bash  #Exemple de lancement : ./import.sh 2> log.txt  #################### VARIABLES A RENSEIGENR AVANT LANCEMENT DU SCRIPT ###################  # informations de connexion base cible import  db\_host=localhost  db\_user=postgres  db\_password=password  db\_name=dre  db\_port=5432  # nom de l'environnement à partir duquel les dump ont été extraits  environnement=dev  # date du backup à restaurer au format AAAAMMJJ  date\_backup=20230517  # repertoire où sont déposés les dumps  repertoire\_backup=/data/project/dre/dumps  # liste des modules (séparés par des "," sans espaces) dont la restauration des données est demandée  module\_list=chc,coc,gestion,inscription,mof,ref  # nom du conteneur dans lequel tourne la base dre  # si pas de conteneur ou si client postgresql disponible en local, laisser vide  #container\_name=dre-db  ###################################### FIN VARIABLES #####################################  IFS="," read -a tableau\_modules <<< $module\_list  echo "import des dumps pour les modules : ${module\_list}"  echo "repertoire des dumps : ${repertoire\_backup}"  if [ -z "$container\_name" ]  then      echo "travail avec pg\_restore directement"      export PGPASSWORD=${db\_password}      echo "purge des anciens schemas"      psql -p ${db\_port} -d ${db\_name} -U ${db\_user} -h ${db\_host} -c 'DROP SCHEMA IF EXISTS schema\_chc, schema\_coc, schema\_gestion, schema\_inscription, schema\_mof, schema\_ref CASCADE;'        echo "restauration des modules demandés..."      #parcours de tous les modules dont la restauration est demandée      for (( i=0; i<${#tableau\_modules[@]}; i++ )); do          nom\_dump="${environnement}-${date\_backup}-${tableau\_modules[$i]}.bin"          echo "suppression des extensions pour eviter un conflit ..."          psql -p ${db\_port} -d ${db\_name} -U ${db\_user} -h ${db\_host} -c 'DROP EXTENSION IF EXISTS unaccent, pg\_trgm CASCADE;'          echo "import du dump ${nom\_dump} ..."          pg\_restore -p ${db\_port} -d ${db\_name} -U ${db\_user} -h ${db\_host} --no-owner --no-acl --clean --if-exists ${repertoire\_backup}/${nom\_dump}      done  else      echo "travail avec conteneurs"      echo "purge des anciens schemas..."      docker exec -i ${container\_name} psql -p ${db\_port} -d ${db\_name} -U ${db\_user} -c 'DROP SCHEMA IF EXISTS schema\_chc, schema\_coc, schema\_gestion, schema\_inscription, schema\_mof, schema\_ref CASCADE;'        echo "restauration des modules demandés..."      #parcours de tous les modules dont la restauration est demandée      for (( i=0; i<${#tableau\_modules[@]}; i++ )); do          nom\_dump="${environnement}-${date\_backup}-${tableau\_modules[$i]}.bin"          echo "suppression des extensions pour eviter un conflit ..."          docker exec -i ${container\_name} psql -p ${db\_port} -d ${db\_name} -U ${db\_user} -c 'DROP EXTENSION IF EXISTS unaccent, pg\_trgm CASCADE;'          echo "import du dump ${nom\_dump} ..."          docker exec -i ${container\_name} pg\_restore -p ${db\_port} -d ${db\_name} -U ${db\_user} --no-owner --no-acl --clean --if-exists < ${repertoire\_backup}/${nom\_dump}      done  fi  echo "import des dumps terminé. Vérifier dans les traces que tout s'est bien passé" |

### Exemple de script SQL pour le chargement des données json (collections MongoDB) et la version Pegase

Voici un exemple de script rudimentaire qui est bien entendu adaptable et perfectionnable :

-- suppression du schéma

drop schema if exists schema\_piste\_inscription CASCADE;

-- création du schéma

create schema schema\_piste\_inscription;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Import de la version de Pegase dans une table schema\_piste\_inscription.version\_instance

-- (dans cet exemple, j'utilise le schema\_piste\_inscription mais ce n'est pas forcément le plus logique)

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- suppression de la table

drop table if exists schema\_piste\_inscription.version\_instance;

-- creation de la table au préalable

create table schema\_piste\_inscription.version\_instance (majeure varchar, mineure varchar, source\_json JSONB);

-- recopie du JSON

\COPY schema\_piste\_inscription.version\_instance(source\_json) FROM 'C:\Users\stephane.coutin\Documents\Develop\postgresql\dre\dumps\DRE\_VERSION' csv quote e'\x01' delimiter e'\x02';

-- extraction des valeurs

update schema\_piste\_inscription.version\_instance set majeure = source\_json->>'majeure', mineure = json\_brut->>'mineure' ;

select \* from schema\_piste\_inscription.version\_instance;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Collection dossiers\_inscription

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- suppression de la table

drop table if exists schema\_piste\_inscription.dossiers\_inscription;

-- creation de la table au préalable

create table schema\_piste\_inscription.dossiers\_inscription (

admission\_numero\_candidat varchar,

code\_espace varchar,

lib\_contexte varchar,

code\_apprenant varchar,

source\_json JSONB);

-- recopie du JSON

\COPY schema\_piste\_inscription.dossiers\_inscription(source\_json) FROM 'C:\Users\stephane.coutin\Documents\Develop\postgresql\dre\dumps\test-piste\_inscription-dossiers\_inscription.json' csv quote e'\x01' delimiter e'\x02';

-- extraction des id (exemple)

update schema\_piste\_inscription.dossiers\_inscription

set

admission\_numero\_candidat = source\_json->'choixInscription'->0->'admisison'->'identifiantAdmis'->'numeroCandidat',

code\_espace = source\_json->'choixInscription'->0->'periode'->'code',

lib\_contexte = source\_json->'choixInscription'->0->'cheminInscription'->'cibleInscriptionRef'->'chemin',

code\_apprenant = source\_json->'apprenant'->'codeApprenant'->'value'

;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Collection calendrier

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- suppression de la table

drop table if exists schema\_piste\_inscription.calendrier;

-- creation de la table au préalable

create table schema\_piste\_inscription.calendrier (code\_calendrier varchar, source\_json JSONB);

-- recopie du JSON

\COPY schema\_piste\_inscription.calendrier(source\_json) FROM 'C:\Users\stephane.coutin\Documents\Develop\postgresql\dre\dumps\test-piste\_inscription-calendrier.json' csv quote e'\x01' delimiter e'\x02';

-- extraction des id

update schema\_piste\_inscription.calendrier set code\_calendrier = source\_json->'code';

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Collection chemins

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- suppression de la table

drop table if exists schema\_piste\_inscription.chemins;

-- creation de la table au préalable

create table schema\_piste\_inscription.chemins (source\_json JSONB);

-- recopie du JSON

\COPY schema\_piste\_inscription.chemins(source\_json) FROM 'C:\Users\stephane.coutin\Documents\Develop\postgresql\dre\dumps\test-piste\_inscription-chemins.json' csv quote e'\x01' delimiter e'\x02';

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Collection paiements

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- suppression de la table

drop table if exists schema\_piste\_inscription.paiements;

-- creation de la table au préalable

create table schema\_piste\_inscription.paiements (source\_json JSONB);

-- recopie du JSON

\COPY schema\_piste\_inscription.paiements(source\_json) FROM 'C:\Users\stephane.coutin\Documents\Develop\postgresql\dre\dumps\test-piste\_inscription-paiements.json' csv quote e'\x01' delimiter e'\x02';

## Modèle de données ‘métier’

Il est disponible dans le share PC-SCOL, dans l’espace cartographie.

Pour la v24, le lien vers le répertoire est <https://share.pc-scol.fr/smart-link/11586707-2915-4794-a4b2-83940493515c/>

Et le fichier est ***Modèle objet métier Pegase v24.pdf***

## Modèle physique des données

Les fichiers sont également disponibles dans le share PC-SCOL, dans l’espace cartographie.

Pour la v24, le lien vers le répertoire est <https://share.pc-scol.fr/smart-link/11586707-2915-4794-a4b2-83940493515c/>

### Le dictionnaire des données

Le fichier excel ***MPDPegaseV24-1.xlsx*** contient le dictionnaire de données. Il est à utiliser conjointement avec le modèle objet métier.

### Les différences par rapport à la v21

Le fichier excel ***Différence MPD PEGASE v23 v24.xlsx*** liste, schéma par schéma et table par table les modifications apportées au modèle physique de données.

### Extraction de la documentation embarquée

La documentation des tables et colonnes est embarquée dans des commentaires tels que gérés par PostgreSQL.

Pour extraire en un coup celles des tables et colonnes pour une bdd, la requête ci-dessous. Cette requête est utilisée pour générer le modèle physique de données.

|  |
| --- |
| SELECT  isc.table\_schema,  isc.table\_name,  obj\_description(format('%s.%s',isc.table\_schema,isc.table\_name)::regclass::oid, 'pg\_class') as table\_description,  isc.column\_name,  isc.data\_type,  isc.character\_maximum\_length,  pg\_catalog.col\_description(format('%s.%s',isc.table\_schema,isc.table\_name)::regclass::oid,isc.ordinal\_position) as column\_description  FROM  information\_schema.columns isc  WHERE  isc.table\_schema like 'schema\\_%'  and isc.table\_name not like '\\_%'  order by 1,2,4 |

A noter une requête pour extraire les commentaires des tables uniquement

|  |
| --- |
| SELECT relname as nom\_table,obj\_description(oid) as description FROM pg\_class WHERE relkind = 'r' AND relname not like 'pg\\_%' AND relname not like 'sql\\_%' AND relname not like '\\_\_%' ORDER BY relname |

## Conseils pour l’écriture de requêtes

Ces conseils et exemples s’appliquent à la version 23 de PEGASE.

| **Titre** | **Contenu** |
| --- | --- |
| Champs code et code\_ | Les champs code sont utilisés en général en tant que clé fonctionnelle d'éléments. Les règles de nommage en bdd sont de nommer code le champs clé fonctionnelle d'un élément (ex : apprenant.code) et 'code\_nomTable' la clé fonctionnelle d'une table liée (exemple contact.code\_apprenant).  Dans le modèle de données métier, afin de faciliter la compréhension nous nommerons toujours le champ code par 'code\_nomObjet'. |
| Champs id et id\_\* | De nombreuses tables possèdent un champ id qui est la clé primaire. Ces champs sont attribués automatiquement par le SGBD et sont basés sur des séquences.  A l'intérieur d'un même schéma ils sont utilisés en tant que FK, et peuvent donc être utilisés pour réaliser des jointures efficaces.  ATTENTION : à l’exception des tables du schema\_odf (voir ci-dessous), leur valeur n'est pas préservée au travers des divers schémas. De ce fait il ne faut pas les utiliser pour réaliser une jointure entre deux tables n'appartenant pas au même schéma. Il faudra utiliser les clés fonctionnelles. |
| Utilisation des uuid dans schema\_odf | Dans schema\_odf, tous les objets possèdent un champ nommé id de type uuid. Il est utilisé pour les FK dans les autres tables du schema\_odf.  Il est propagé aux autres schémas de Pegase et en général utilisé comme FK pour référencer les objets issus de ODF. Il est donc possible de faire des jointures vers les tables odf depuis d’autres schemas via ce uuid. |
| Objet sur lequel portent admission et inscription | L'inscription porte sur un contexte (précédemment un chemin) alors que l'admission porte sur un objet maquette (formation ou objet de formation). Théoriquement il peut y avoir plusieurs contextes pour un même objet\_maquette. |
| Utilisation des uuid dans schema\_chc | Dans schema\_chc, les tables utilisent des uuid en tant qu'identifiants techniques.  A l'intérieur du schéma ils sont utilisés en tant que FK, et peuvent donc être utilisés pour réaliser des jointures efficaces.  ATTENTION : leur valeur n'est pas préservée au travers des divers schémas. De ce fait il ne faut pas les utiliser pour réaliser une jointure entre deux tables n'appartenant pas au même schéma. Il faudra utiliser les clés fonctionnelles. |