

Document d’Architecture Technique

|  |  |
| --- | --- |
|  | Intégration et chargement des publications/mesh relatives aux NCT en bases postgre |
| V1.0 |

***A Paris, le 14/02/2022***

***A destination de :*** *Frédéric LEGRAND et Stéphane DE GRAEVE*

Contenu

[1- OBJET DU DOCUMENT 3](#_Toc96343533)

[2- DEPLOIEMENT PROJET 4](#_Toc96343534)

[2.1- Installation 4](#_Toc96343535)

[2.3- Packages 4](#_Toc96343536)

[2.2- Environnement postgre 5](#_Toc96343537)

[3- STRUCTURE PROJET 6](#_Toc96343538)

[3.1- Répertoire racine du projet 6](#_Toc96343539)

[3.2- Répertoires de travail 6](#_Toc96343540)

[3.3- Paramètres 7](#_Toc96343541)

[4- SCRIPT PRINCIPAL – MAIN 9](#_Toc96343542)

[4.1- Présentation du MAIN 9](#_Toc96343543)

[4.2- Mécanique par BLOC 9](#_Toc96343544)

[4.3- Log et temps de traitement 11](#_Toc96343545)

[4.4- Liste des fonctions 11](#_Toc96343546)

[5- FICHIERS INPUT 15](#_Toc96343547)

[6- TABLES POSTGRE 15](#_Toc96343548)

[6.1- L’ensemble des tables 15](#_Toc96343549)

[6.2- Index et clés primaires postgre 17](#_Toc96343550)

[6.3- Gestion des schémas postgre 18](#_Toc96343551)

[6.4- MCD 19](#_Toc96343552)

[7- CHEMINS RELATIFS 20](#_Toc96343553)

# 1- OBJET DU DOCUMENT

Dans le cadre de la mission et de la livraison concernant le téléchargement des données de publications/mesh relatives aux NCT, la livraison des programmes sera accompagnée d’un DAT et d’un mode opératoire.

Pour remettre du contexte sur la livraison de ce lot, il est question de cibler les publications pubmed ayant des liens vers des publications NCT. Pour ce faire nous nous sommes appuyés sur l’API du site [NCBI](https://pubmed.ncbi.nlm.nih.gov/) (National Center for Biotechnology Information), site ou l’on trouve les publications pubmed. Il est également nécessaire de ramener les référentiels des codes MESH, codes présents dans les publications pubmed.

Le document d’architecture technique a pour objectif de présenter le projet dans son ensemble, son architecture et la mécanique globale des programmes.

Nous allons voir à travers ce document, toute l’organisation du projet qui va permettre d’extraire les publications et référentiels jusqu’à la phase de chargement des données dans les bases postgre INCA.

# 2- DEPLOIEMENT PROJET

## 2.1- Installation

L’ensemble du projet peut être déployé à l’aide d’un fichier zip, la gestion des chemins est interne au projet et ne nécessite pas le paramétrage de chemins absolus.

## 2.2- R et packages nécessaires

Dans cette partie sont explicitées les versions utilisées pour le développement du projet. Il est possible d’avoir des versions plus récentes (R et Packages) mais cela pourrait provoquer des effets de bords.

**R/RStudio** :

* L’installation du logiciel R (4.0.2)
* L’installation du logiciel RStudio (2021.09.02-382)

**Packages** :

Un script permet de lancer la détection de packages listés en INPUT dans le fichier *list\_package\_projet.csv*. Si les packages ne sont pas installés sur la machine, une procédure d’installation est lancée. Cette procédure va rechercher les dernières versions des packages sur le CRAN.

* Lien CRAN : *https://cran.rstudio.com/*

**Liste des packages du projet** :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Package** | **Version** | **Suggests** | **Depends** | **Imports**  Gestion automatique de l’installation des packages via le CRAN (recherche et installation des dépendances) |
| data.table | 1.13.0 | bit64, curl, R.utils, xts, nanotime, zoo, yaml, knitr,  rmarkdown | R (>= 3.1.0) | methods |
| DBI | 1.1.0 | blob, covr, hms, knitr, magrittr, rmarkdown, rprojroot,  RSQLite (>= 1.1-2), testthat, xml2 | methods, R (>= 3.0.0) | NA |
| dplyr | 1.0.1 | bench, broom, callr, covr, DBI, dbplyr (>= 1.4.3), knitr,  Lahman, lobstr, microbenchmark, nycflights13, purrr, rmarkdown,  RMySQL, RPostgreSQL, RSQLite, testthat (>= 2.1.0), withr | R (>= 3.2.0) | ellipsis, generics, glue (>= 1.3.2), lifecycle (>= 0.2.0),  magrittr (>= 1.5), methods, R6, rlang (>= 0.4.7), tibble (>=  2.1.3), tidyselect (>= 1.1.0), utils, vctrs (>= 0.3.2) |
| odbc | 1.3.3 | covr, DBItest, magrittr, RSQLite, testthat, tibble | R (>= 3.2.0) | bit64, blob (>= 1.2.0), DBI (>= 1.0.0), hms, methods, rlang,  Rcpp (>= 0.12.11) |
| rentrez | 1.2.3 | testthat, knitr, rmarkdown | R (>= 2.6.0) | XML, httr (>= 0.5), jsonlite (>= 0.9) |
| RPostgreSQL | 0.7-3 | NA | R (>= 2.9.0), methods, DBI (>= 0.3) | NA |
| stringr | 1.4.0 | covr, htmltools, htmlwidgets, knitr, rmarkdown, testthat | R (>= 3.1) | glue (>= 1.2.0), magrittr, stringi (>= 1.1.7) |
| tidyr | 1.1.1 | covr, jsonlite, knitr, repurrrsive (>= 1.0.0), rmarkdown,  readr, testthat (>= 2.1.0) | R (>= 3.1) | dplyr (>= 0.8.2), ellipsis (>= 0.1.0), glue, magrittr, purrr,  rlang, tibble (>= 2.1.1), tidyselect (>= 1.1.0), utils, vctrs  (>= 0.3.0), lifecycle |
| XML | 3.99-0.8 | bitops, RCurl | R (>= 4.0.0), methods, utils | NA |
| xml2 | 1.3.2 | covr, curl, httr, knitr, magrittr, mockery, rmarkdown,  testthat (>= 2.1.0) | R (>= 3.1.0) | methods |

## 2.3- Environnement postgre

Afin d’accéder aux environnements postgre INCA, nous devons nous authentifier sur le serveur. Pour se faire nous devons renseigner dans un fichier de config ***(./Prod/Config/config.txt***) les informations de connexion :

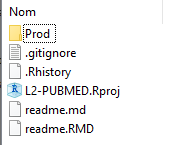
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **conn** | **port** | **user** | **password** | **dbname** |
| virt-db-hygie | 5433 | INC\_U\_PRI\_A | XXXXXXXX | INC |

# 3- STRUCTURE PROJET

## 3.1- Répertoire racine du projet

L’ensemble des codes R est inclus dans un projet. La fonctionnalité « Projet » de R permet de gérer les chemins relatifs et par conséquent le projet n’a pas besoin d’une initialisation des chemins. Le chemin « racine » du projet est le même sur n’importe quel environnement.

**Aperçu de l’architecture projet :**

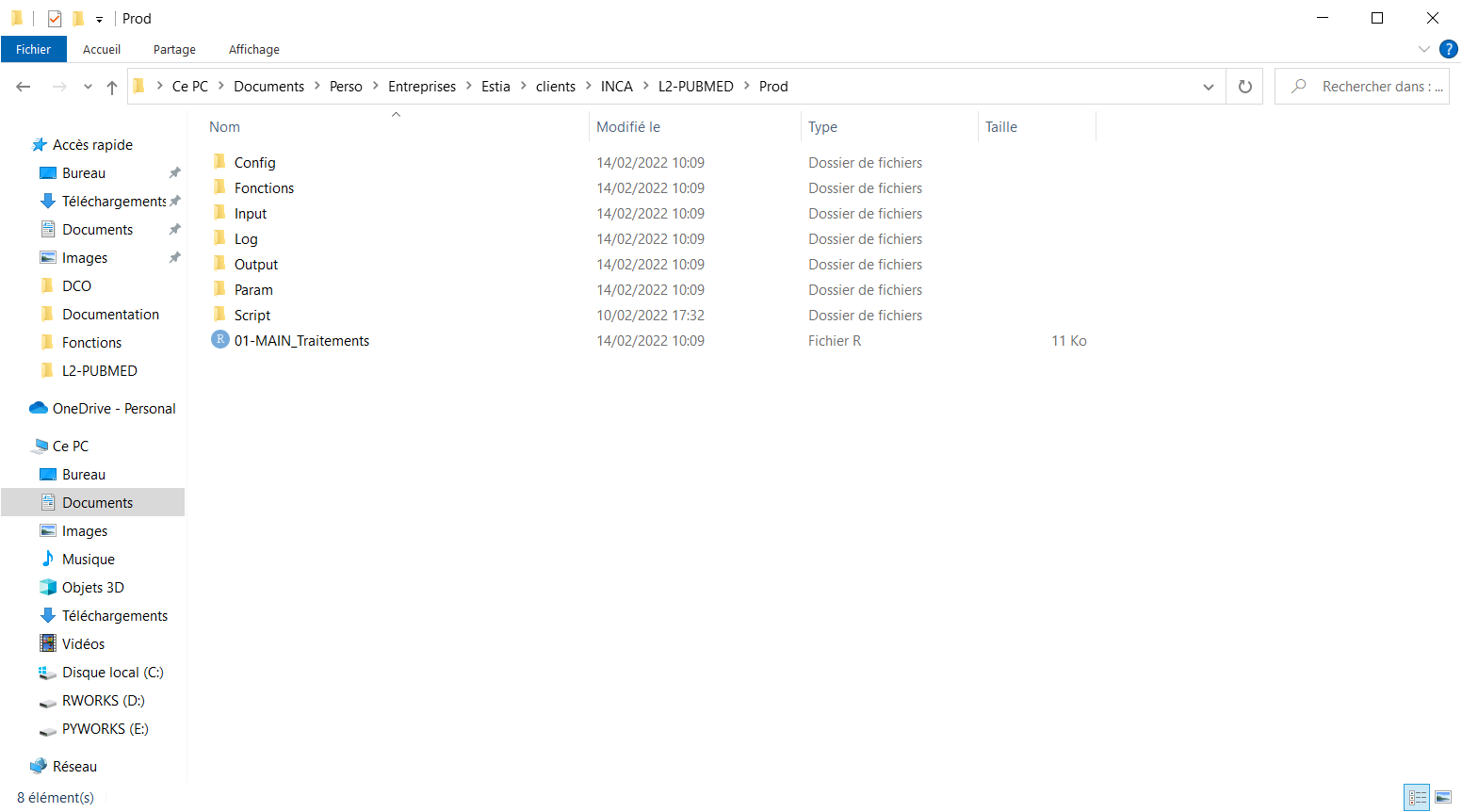


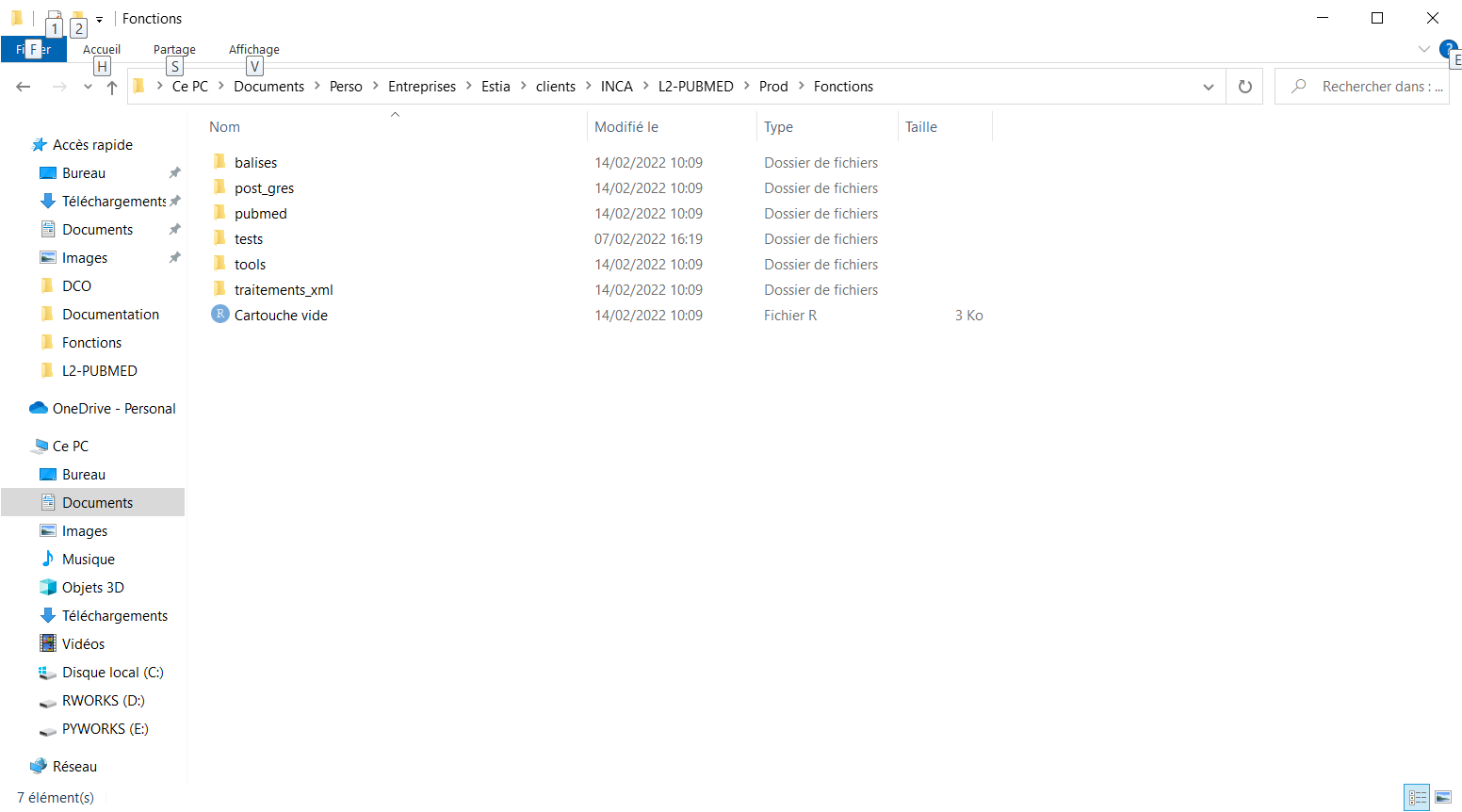
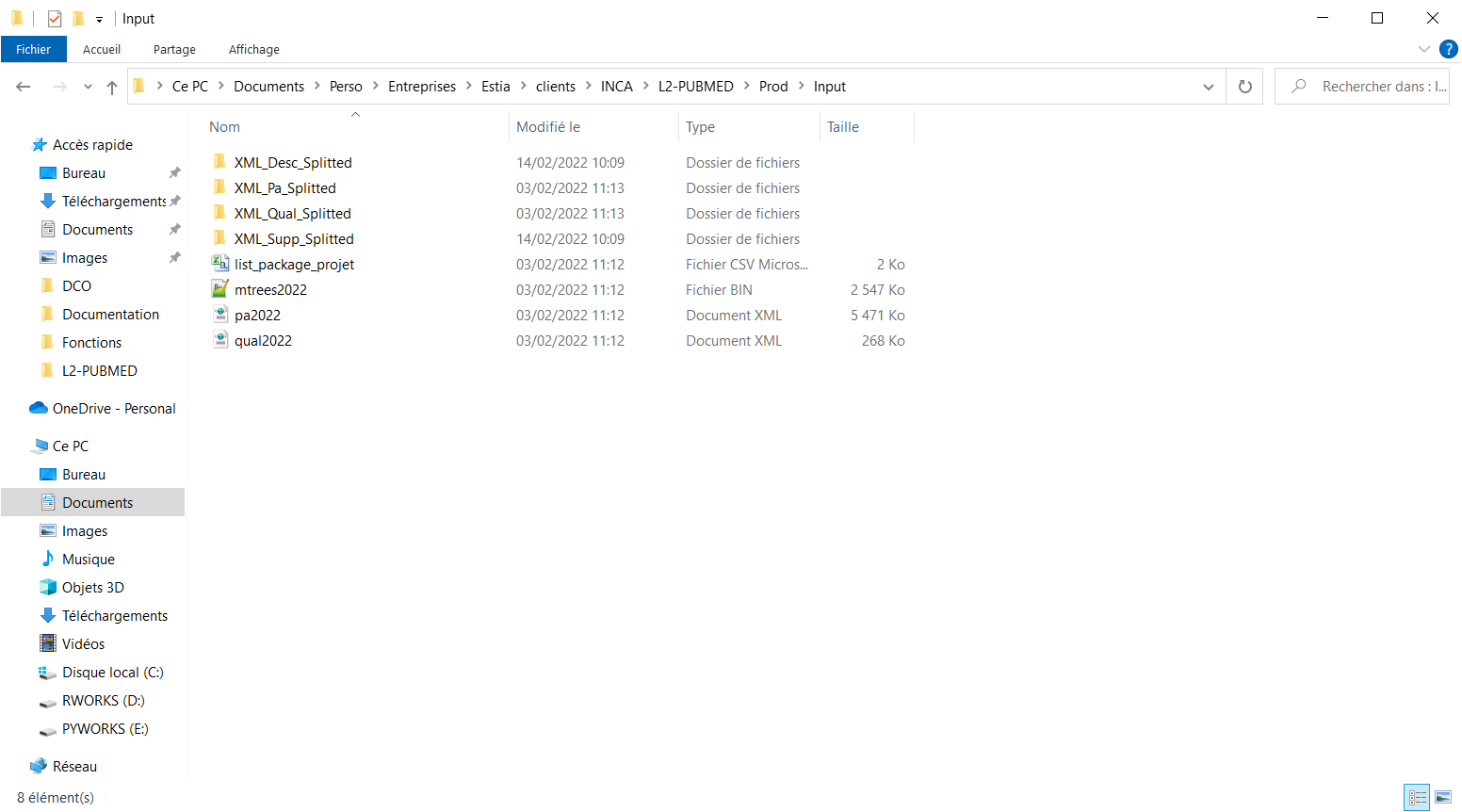
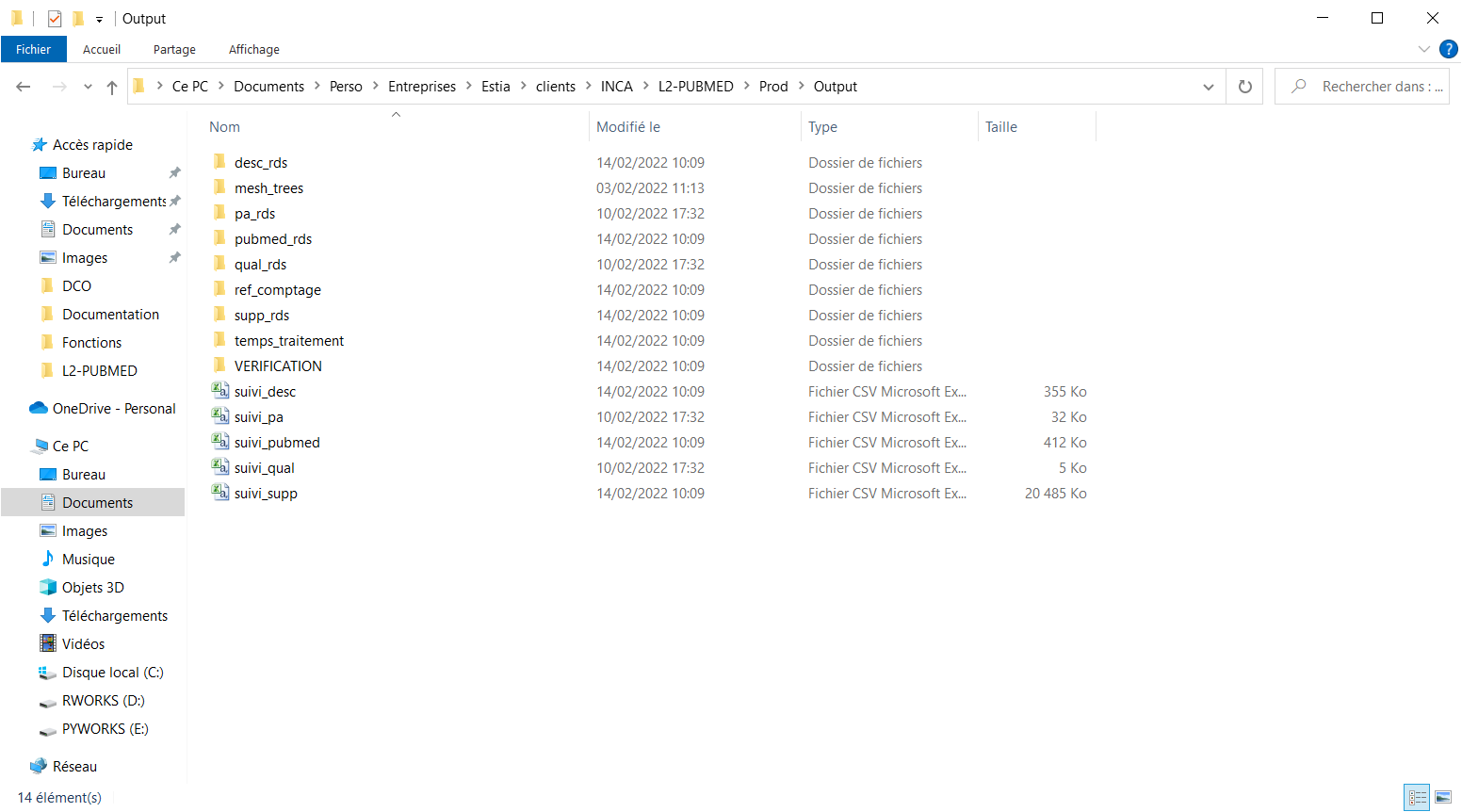
## 3.2- Répertoires de travail

Le contenu du répertoire est un projet uniquement avec la présence d’un élément de type « R Project » à la racine.

Nous allons nous intéresser principalement au répertoire « Prod/ ».

**Architecture des répertoires de travail :**



* Config/ : Contient les informations de connexion aux bases postgre
* Fonctions/ : Contient toutes les fonctions nécessaires au projet
  + - * + 
        + Les fonctions sont placées dans des répertoires spécifiques pour faciliter la navigation.
* Input/ : Contient tous les éléments nécessaires au projet
  + - * + Les référentiels MESH (desc, supp, pa, qual)
        + [Lien référentiel MESH](https://nlmpubs.nlm.nih.gov/projects/mesh/2021/xmlmesh/)
        + Une arborescence de fichiers dédiée au traitement des référentiels au format XML
        + 
      * Log/ : Contient l’ensemble des fichiers de log nommés par programme et datés
      * Output/ : Contient à la racine des fichiers de suivi des traitements (pubmed + référentiels) et une arborescence de sous répertoires de stockage
        + 
        + Les sous répertoires sont principalement des fichiers RDS contenant de la DATA (pubmed ou référentiels)
        + Nous pouvons trouver d’autres répertoires pour les temps de traitement des programmes, les référentiels des balises ou un répertoire de vérification de fichiers RDS.
      * Param/ : Contient le script de paramétrage (variables globales + chargement des librairies)
      * Script/ : Contient un script d’installation automatique des packages

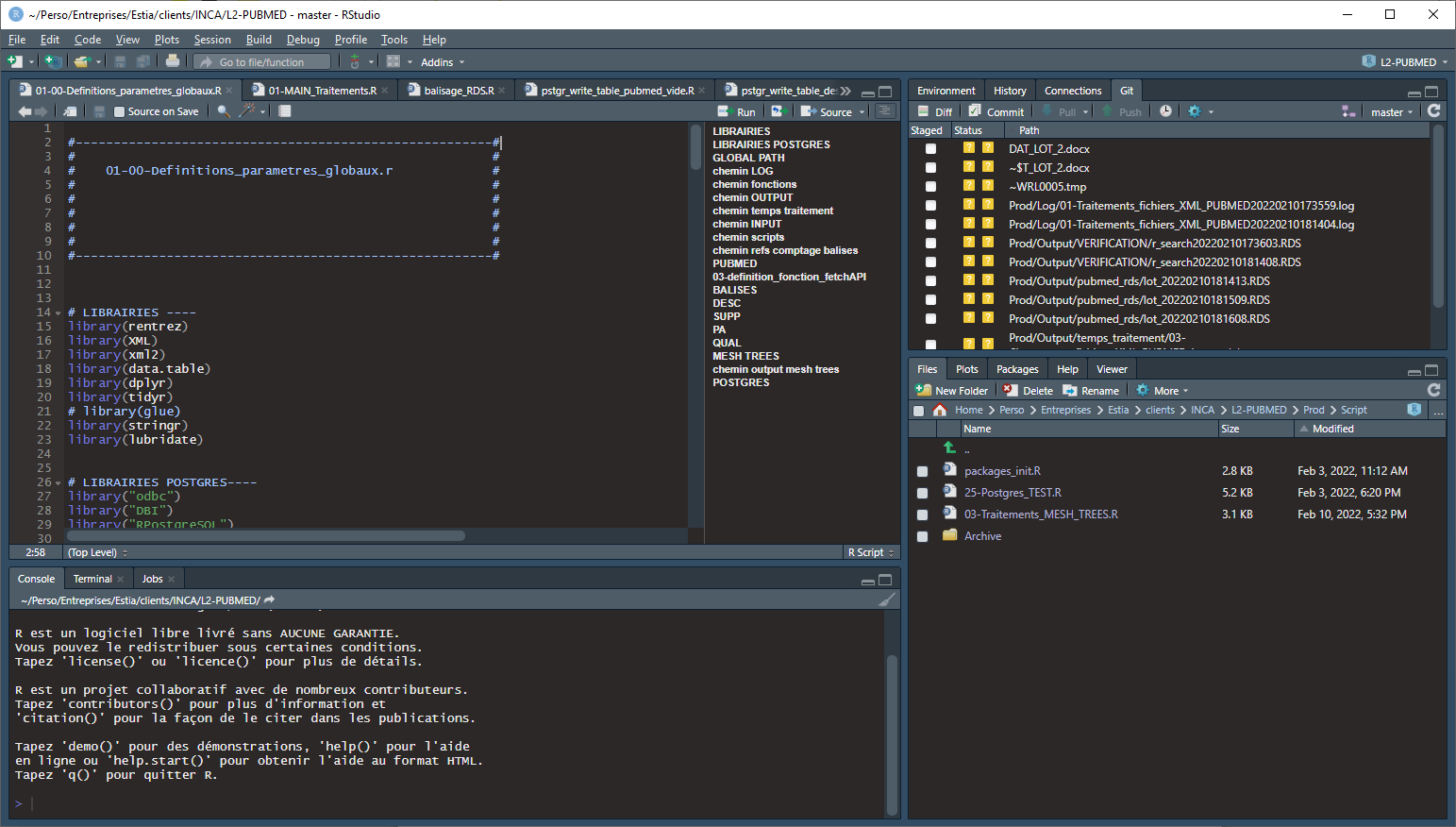
## 3.3- Paramètres

Pour rappel, le script de paramétrage du projet est présent dans le répertoire : ***./Prod/Param/***.

L’objectif du script de paramétrage est d’éviter dans le code la présence de chemins écris « en dur » (ex : « /Prod/Output/mon\_fichier.csv »). Les chemins ou autres informations globales au projet sont définies dans le script de paramétrage.

L’organisation du script est structurée par bloc :

* Chargement des **librairies** nécessaires
* Connexion à la **base postgre**
* Définition des **chemins/paramètres** globaux
* Gestion des **balises**
* Bloc traitements **pubmed**
* Bloc traitements du référentiel **Desc**
* Bloc traitements du référentiel **Supp**
* Bloc traitements du référentiel **Pa**
* Bloc traitements du référentiel **Qual**
* Bloc traitements du référentiel **TREE CODE**

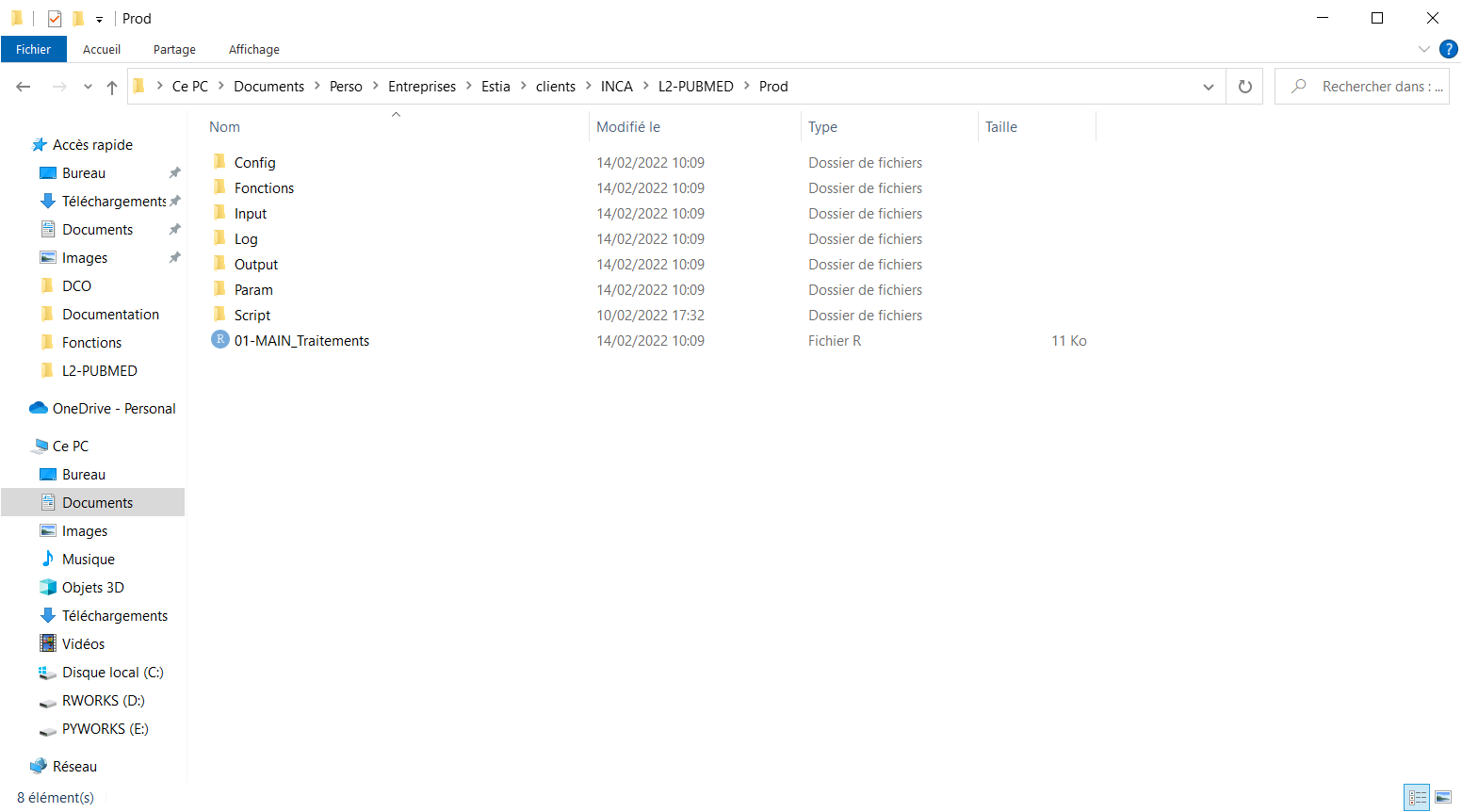


Structure des paramètres par BLOC

# 4- SCRIPT PRINCIPAL – MAIN

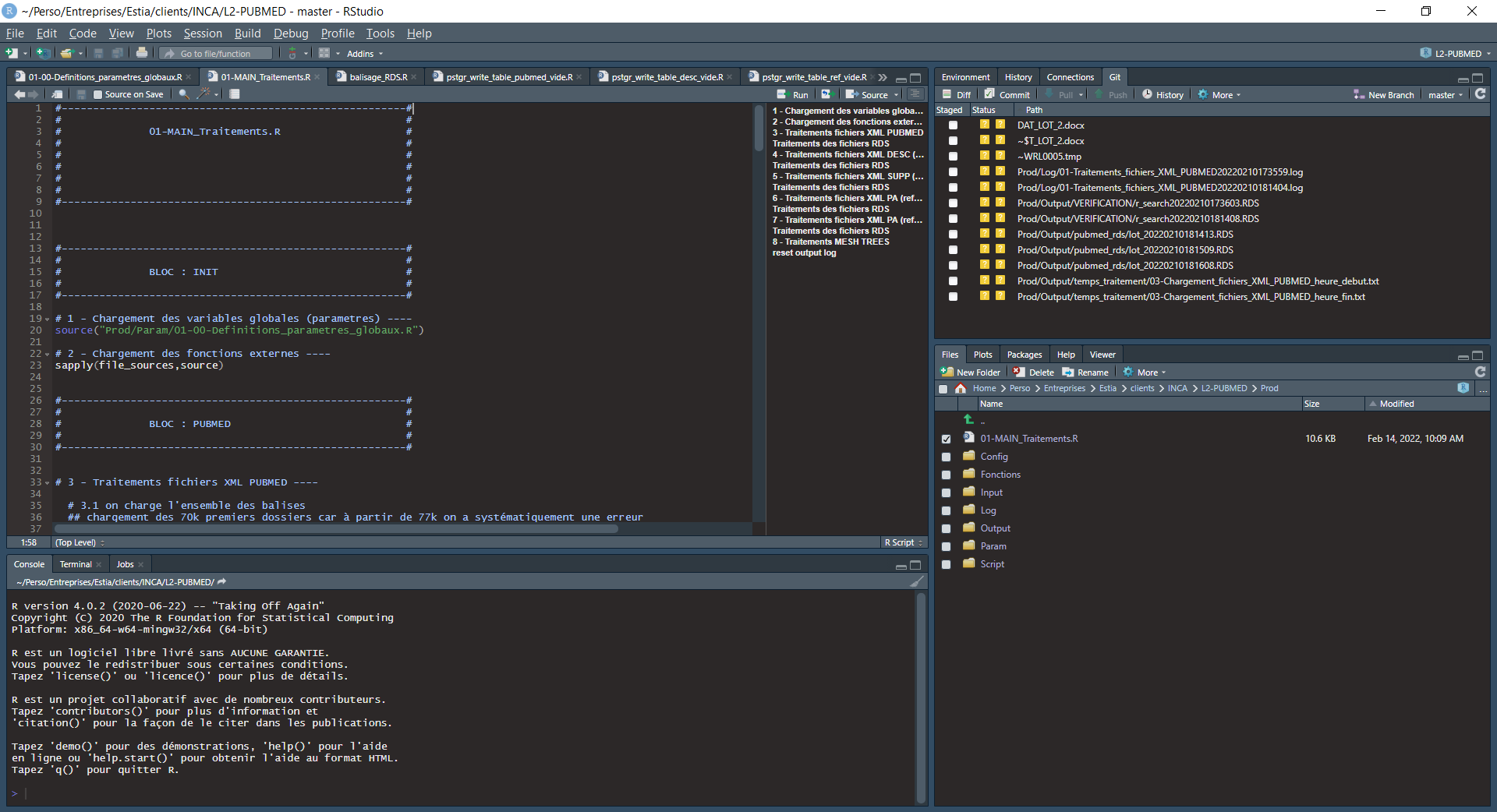
## 4.1- Présentation du MAIN

Le point d’entrée du projet est le script ***01-MAIN\_Traitements.R*** (*. /Prod/*), ce script est organisé en BLOC et chaque bloc peut s’exécuter de façon **indépendante** (hors bloc « gestion des schémas »).



Dans tous les cas, la partie ***INIT*** en début de script doit être lancée pour charger les paramètres et fonctions en mémoire.

L’organisation du script est claire et les blocs sont facilement identifiables :



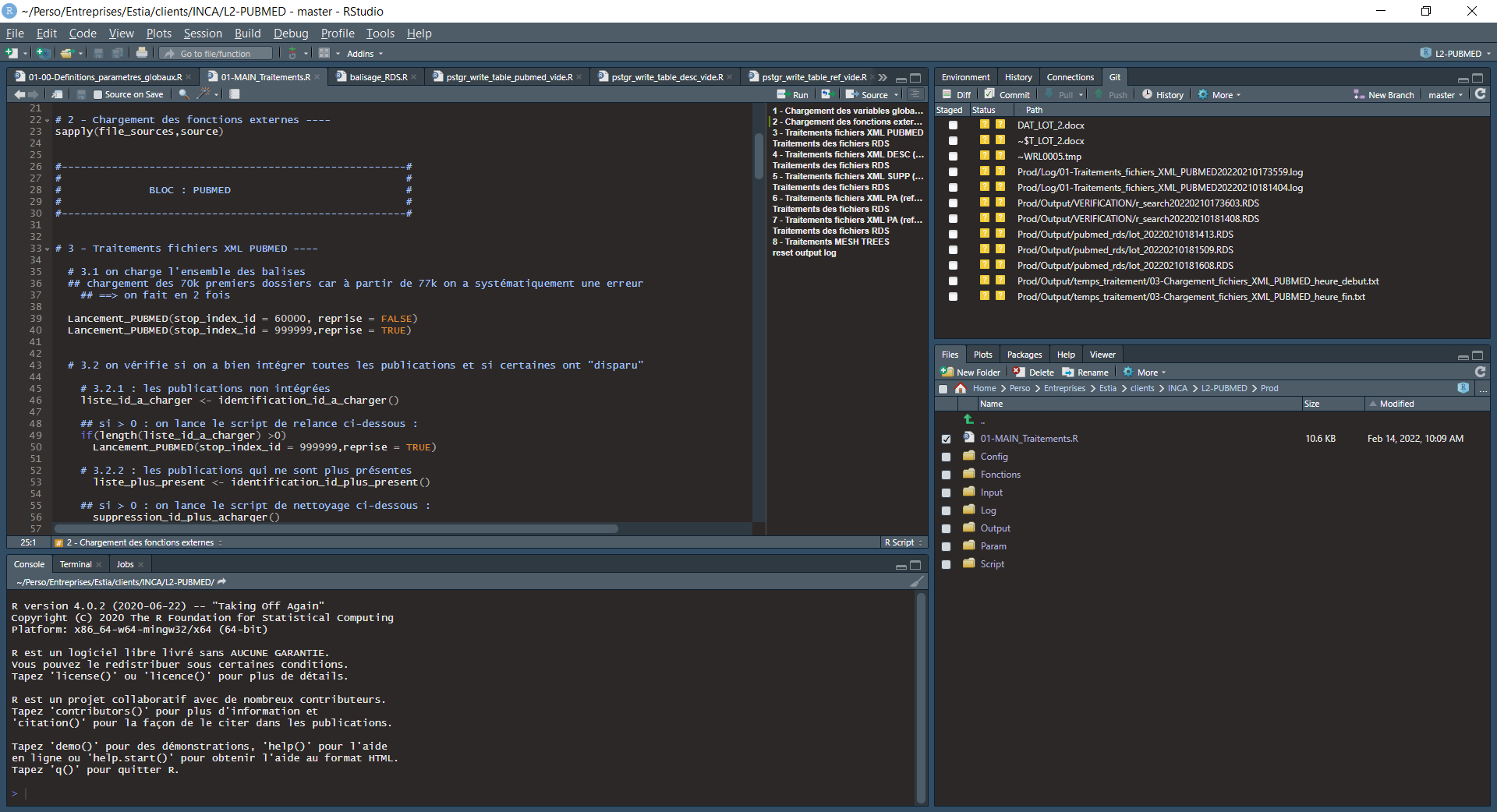
## 4.2- Mécanique par BLOC

Nous allons voir la mécanique et l’enchainement global des BLOCS à travers une présentation plus détaillée du ***MAIN*** (script : **01-MAIN\_Traitements.R**).

* **BLOC INIT** : Bloc nécessaire l’exécution de chaque bloc :

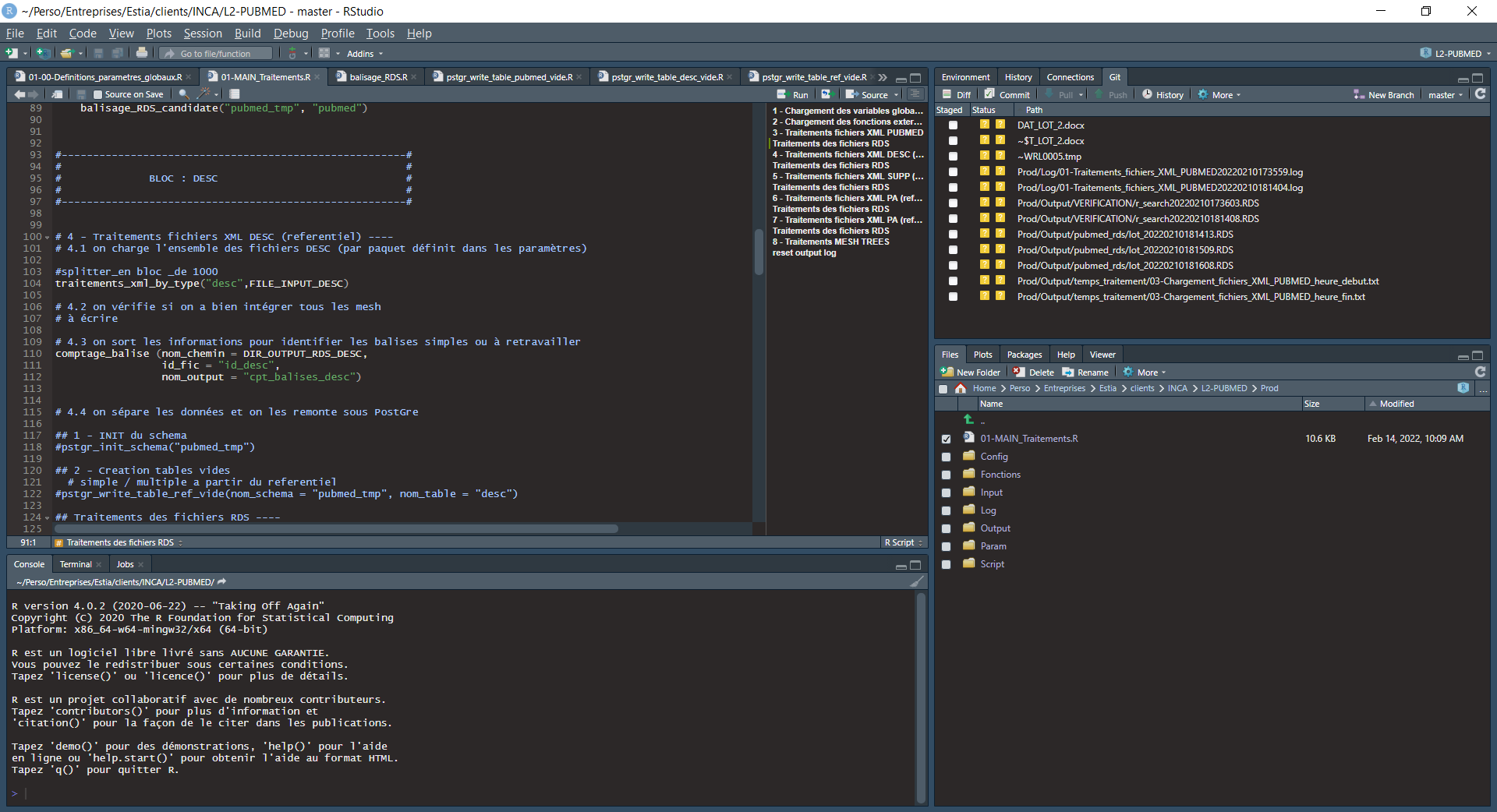
Ce bloc permet de charger en mémoire les paramètres (chemins relatifs, variables globales) et les fonctions.

* **BLOC PUBMED** : Requêtes vers l’API NCBI + traitements des publications pubmed :



1. Lancement de la partie ciblage/téléchargement des publications pubmed au format XML
   1. Traitement et conversion des fichiers XML en fichiers .RDS en sortie (*Prod/Output/pubmed\_rds/*)
   2. Gestion des téléchargements et contrôle présence NCT, création d’un fichier de suivi (*Prod/Output/ suivi\_pubmed.csv*)
2. Phase de vérification
   1. Check des id téléchargés via l’API
   2. Lancement de la procédure de reprise si nécessaire
   3. Nettoyage et traitement des doublons
3. Identification des balises
   1. Analyse des balises et comptage
   2. Création d’un référentiel (validation du référentiel pour split des tables)
4. Traitements des fichiers RDS
   1. Split en tables simples et multiples
   2. Chargement des tables dans l’environnement postgres

* **BLOCS REFERENTIELS** : Chaque BLOC est organisé de la même manière et permet de traiter les différents référentiels MESH (Desc, Supp, Pa, Qual) :



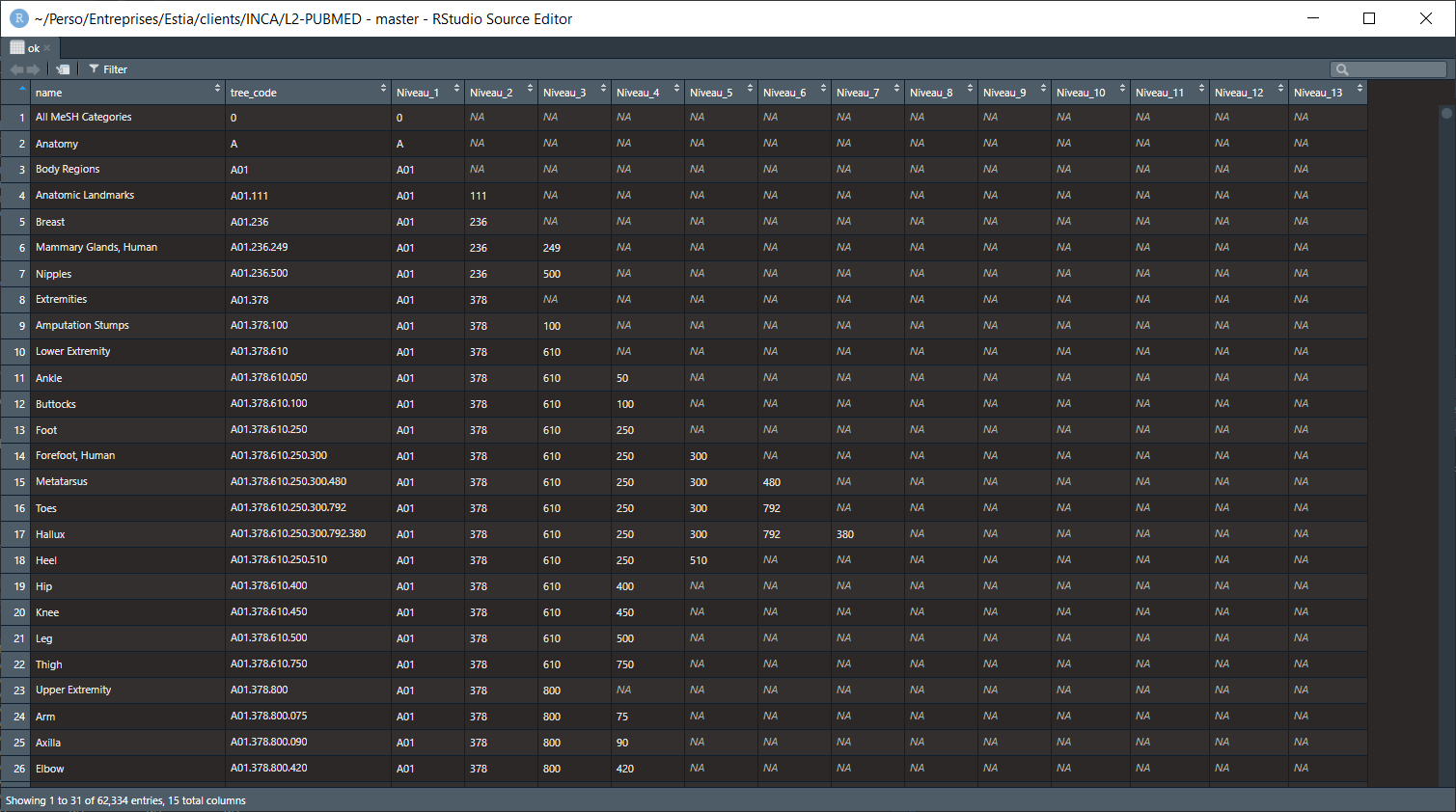
1. Traitements des référentiels splittés au format XML
   1. Création d’un fichier de suivi des traitements (*Prod/Output/*)
   2. Traitement et conversion des fichiers en .RDS
2. Phase de vérification
   1. Check des id présents entre les fichiers en sortie et le fichier en entrée
3. Identification des balises
   1. Analyse des balises et comptage
   2. Création d’un référentiel (validation du référentiel pour split des tables)
4. Traitements des fichiers RDS
   1. Split en tables simples et multiples
   2. Chargement des tables dans l’environnement postgres

* **BLOC MESH TREES** : Ce dernier BLOC a pour fonction de charger en base postgre l’ensemble des codes MESH.

Le fichier est disponible via le site [NIH - NLM National Library of Medecine](https://nlmpubs.nlm.nih.gov/projects/mesh/2021/meshtrees/)

1. Chargement du fichier et ajout des catégories (« 0 » : « All MeSH Categories », « A » : « Anatomy », …, « Z » : « Geographicals »)
2. Création de niveaux en fonction de l’arborescence des codes
3. Chargement du référentiel dans les bases postgres

**Aperçu en sortie** :



## 4.3- Log et temps de traitement

Le déroulement du programme est tracé dans des fichiers de log (***./Prod/Log***). Ces fichiers sont nommés avec le nom du programme suivi de la date.

De la même façon, le temps des traitements des fonctions/programmes sont tracés dans des fichiers (*.Prod/Output/temps\_traitment*). Chaque fichier de suivi comporte le nom du programme et la date.

## 4.4- Liste des fonctions

Pour rappel, le script de lancement principal ***01-MAIN\_Traitements.R*** appellera l’ensemble des fonctions nécessaires au projet.

Chaque fonction est précédée d’un cartouche avec un résumé, la description des paramètres et la valeur en sortie.

| **Nom** | **Chemin relatif** | **Description** |
| --- | --- | --- |
| ***split\_rds.R*** | **./Prod/Fonctions/balises** | Application des règles de gestion par fichier RDS à partir d’un  référentiel de comptage  En sortie :  Split en data frame « simples » ou « multiples » |
| ***balises\_simples\_dates.R*** | **./Prod/Fonctions/balises** | Fonction traitant les dates  Transforme plusieurs lignes d'une date (day-month-year) en une seule ligne  En sortie : Pour chaque id et pour chaque type de date, la date est concaténée sur une ligne |
| ***balisage\_RDS.R*** | **./Prod/Fonctions/balises** | Fonction qui scan un répertoire contenant des .RDS  Appel la fonction de découpage en table simples/multiples : *split\_rds()*  En sortie : Ecrit les tables simples/multiples sous postgre : *pstgr\_write\_table()* |
| ***balisage\_RDS\_candidate.R*** | **./Prod/Fonctions/balises** | Cette fonction a pour but de remonter les données sous Postgre en séparant les données des balises simples et les balises multiples  1) création du schéma cible s'il n'existe pas  2) création des structures vides pour accueillir les données  3) séparation des rds en partie simple/mult puis remontée sous Postgre |
| ***comptage\_balise.R*** | **./Prod/Fonctions/balises** | Cette fonction a pour but de créer le référentiel qui permettra de définir les balises simples et les balises multiples  En sortie :  un fichier csv avec les différents éléments pour qualifier les balises simples/mult |
| ***load\_rds\_file\_to\_db.R*** | **./Prod/Fonctions/balises** |  |
| ***pstgr\_write\_table.R*** | **./Prod/Fonctions/post\_gres** | Chargement dans un schéma et une table paramétrable d’un fichier RDS |
| ***pstgr\_init\_schema.R*** | **./Prod/Fonctions/post\_gres** | Création d’un schéma avec le nom en paramètre  Check si le schéma existe sinon le créé |
| ***create\_index.R*** | **./Prod/Fonctions/post\_gres** | Création des index sur un schéma et un nom de table passé en paramètre  Check si l’index existe sinon le créé |
| ***create\_primary\_key.R*** | **./Prod/Fonctions/post\_gres** | Création des clés primaires sur un schéma et un nom de table passé en paramètre |
| ***grant\_rights.R*** | **./Prod/Fonctions/post\_gres** | Application des autorisations sur l’utilisateur **INC\_U\_PRI** |
| ***initialize\_df\_table\_simple\_mult.R*** | **./Prod/Fonctions/post\_gres** | Structure des noms de colonnes des tables simples et multiples |
| ***pstgr\_write\_table\_ref\_vide.R*** | **./Prod/Fonctions/post\_gres** | Organisation des noms de colonnes et typages pour chargement des tables vides (structure) en bases postgre |
| ***check\_xml.R*** | **./Prod/Fonctions/pubmed** | Vérification de la présence d’un NCT dans les publications téléchargées |
| ***test\_reprise\_necessaire.R*** | **./Prod/Fonctions/pubmed** | test si la publication est bien dans la liste des id à charger |
| ***fetchAPI.R*** | **./Prod/Fonctions/pubmed** | Cette fonction permet la récupération des publications liées à un NCT au format XML  Cette fonction encapsule une fonction du package ***rentrez***  En sortie :  Une liste de fichiers XML correspondant aux id |
| ***full\_traitements\_xml.R*** | **./Prod/Fonctions/pubmed** | Cette fonction permet l'intégration en RDS de l'ensemble des XML  Chaque RDS contiendra le nombre de publication défini en paramètre  En sortie :  Les fichiers RDS issus de la structuration des XML  Création du fichier de suivi des intégrations |
| ***identification\_id\_a\_charger.R*** | **./Prod/Fonctions/pubmed** | Cette fonction a pour vocation de récupérer l'ensemble des *id pubmed* restant à charger  Elle fait le delta entre l'ensemble des *id pubmed* à charger et ce qui est tracé dans le fichier de suivi |
| ***identification\_id\_plus\_present.R*** | **./Prod/Fonctions/pubmed** | Cette fonction a pour vocation de récupérer l'ensemble des *id pubmed* qui ont été charges mais qui ne sont plus rattaches à une publication désormais  Ce cas peut apparaitre lors d'une reprise notamment |
| ***Lancement\_PUBMED.R*** | **./Prod/Fonctions/pubmed** | Cette fonction lance toute la récupération des publications :  1) récupération des publications liées à un NCT  2) récupération par paquet des publications au format XML  3) scrutation des données en fichier structurés |
| ***searchAPI.R*** | **./Prod/Fonctions/pubmed** | Cette fonction permet la récupération des id des publications liées à un NCT  Elle permet également de remonter sur l'API la liste des publications que l’on ramènera par la suite pour les récupérer en masse (option *use\_history = TRUE*) |
| ***identification\_id\_doublons.R*** | **./Prod/Fonctions/pubmed** | Cette fonction a pour vocation de récupérer l'ensemble des *id pubmed* qui ont été charges plusieurs fois  Dans ce cas on garde la dernière intégration  En sortie :  un vecteur avec l'ensemble des publications a supprimer et des fichiers concernes |
| ***suppression\_en\_doublons.R*** | **./Prod/Fonctions/pubmed** | Cette fonction a pour vocation de supprimer l'ensemble des *id pubmed* qui sont chargées plusieurs fois  Elle se base sur le vecteur liste\_doublons créé précédemment et sur le fichier de suivi pour trouver les .RDS à épurer  On ne garde que la dernière intégration |
| ***identification\_id\_a\_charger.R*** | **./Prod/Fonctions/pubmed** | Cette fonction a pour vocation de supprimer l'ensemble des *id pubmed* qui ne sont plus charger  Elle se base sur le vecteur liste\_plus\_present créer précédemment et sur le fichier de suivi pour trouver les .RDS à épurer |
| ***Test.R*** | **./Prod/Fonctions/tests** | Regroupement de fonctions de tests sur la partie qualité des data des référentiels |
| ***01-definition\_fonction\_log .R*** | **./Prod/Fonctions/tools** | Création d’un fichier de log et redirection de la console vers ce fichier |
| ***13-definition\_not\_in.R*** | **./Prod/Fonctions/tools** | Création d’une fonction pour symbole « not in » |
| ***write\_data\_check\_to\_csv.R*** | **./Prod/Fonctions/tools** | Ecrit un data frame dt dans un fichier filename situé dans le répertoire dir\_file |
| ***calcul\_temps\_trt.R*** | **./Prod/Fonctions/tools** | Fonction qui permet de calculer le temps des traitements d'une étape  En sortie :  Un fichier txt avec 4 colonnes (label, heure de début, heure de fin, durée en mn) |
| ***initialize\_df\_record.R*** | **./‘’/’’/traitements\_xml** | Initialise un data frame vide avec 6 colonnes  *Nom\_balise, Parent, Chemin, Profondeur, Valeur, Attribut* |
| ***split\_elements.R*** | **./‘’/’’/traitements\_xml** | Collection de plusieurs fonctions spécifiques aux traitements des fichiers XML (organisation et split en plusieurs XML) |
| ***treat\_level\_child.R*** | **./‘’/’’/traitements\_xml** | Traite un niveau d'arborescence du .XML pour un enregistrement donné |
| ***treat\_xml\_file.R*** | **./‘’/’’/traitements\_xml** | Traite les données XML après lecture du fichier .XML |
| ***traitements\_xml\_pubmed.R*** | **./‘’/’’/traitements\_xml** | Export au format RDS un data frame contenant l'ensemble des fichiers xml d'un lot |
| ***traitements\_xml\_pubmed\_file.R*** | **./‘’/’’/traitements\_xml** | Fonction qui traite des lots de fichiers XML pubmed à la maille "fichier" |
| ***treat\_record.R*** | **./‘’/’’/traitements\_xml** | Traite un enregistrement children\_0 |
| ***create\_df\_tag\_attributes.R*** | **./‘’/’’/traitements\_xml** | Cette fonction écrit dans un data frame les différents éléments d'une balise  Elle intervient au point le plus profond de l'arborescence |
| ***traitements\_xml\_by\_type.R*** | **./‘’/’’/traitements\_xml** | L'objectif de cette fonction est de structurer des données d’un fichier XML  Traite un type de données parmi les référentiels supp, qual, desc et pa en écrivant les .rds |

# 5- FICHIERS INPUT

Pour rappel, les référentiels sont les données MESH organisées en plusieurs catégories (Desc, Supp, Pa, Qual et le référentiel tree code). Tous ces fichiers sont au format XML sauf le référentiel des codes « tree view » au format BIN.

Le répertoire *./Prod/Input* permet de stocker tous les référentiels qui seront ensuite traités. Chaque référentiel, hors tree code, est un fichier XML contenant des blocs (identifiés par des balises). Pour chaque bloc nous avons les informations liées à chaque ID.

* **MeSH Descriptor** (Desc) : Le fichier contient des identifiants uniques Mesh (MeSH Unique ID, ex : **D**011716). Pour chaque id, nous avons des informations (description, dates, …) mais aussi des informations vers les autres référentiels (Supp, Pa, Qual et Tree).
* **MeSH Supplemental Concept Records** (Supp) : Le fichier est similaire au référentiel Desc mais ne contient pas les « tree numbers » (ex : **C**038500).
* **MeSH Qualifier files** (Qual) : Le fichier contient comme le fichier Desc toutes les informations sur un code (ex : **Q**000000981)
* **MeSH Pharmacological Action** (Pa) : Le fichier contient des codes Mesh qui existent dans le référentiel Desc (pas de « tree Numbers ») avec des informations complémentaires
* **Mesh Trees** (tree) : Le fichier contient tous les codes permettant de hiérarchiser les Mesh (arborescence mesh).

# 6- TABLES POSTGRE

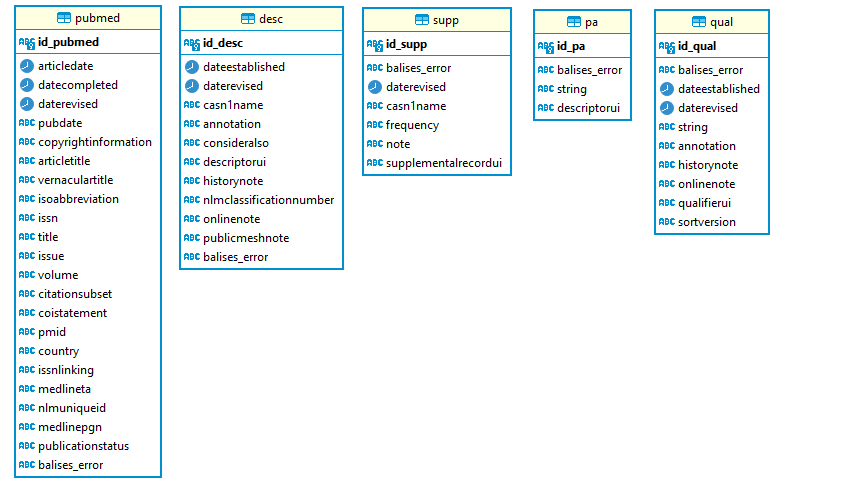
Cette partie est dédiée au chargement des tables « simples » et « multiples » dans les environnements postgre INCA. Le déroulement global se fera en trois étapes :

1. ***Etape INIT*** : Initialisation des tables vides (simples et multiples) et des types pour les champs (caractère, date) + création du schéma temporaire si il n’existe pas
2. ***Chargement des bases*** : Ecriture dans les tables
3. ***Gestion des schémas*** : Stockage des différents schémas

## 6.1- L’ensemble des tables

Une classification des *balises/balises parentes* a permis de découper les données à charger en deux niveaux, les tables simples et multiples.

Les tables simples :

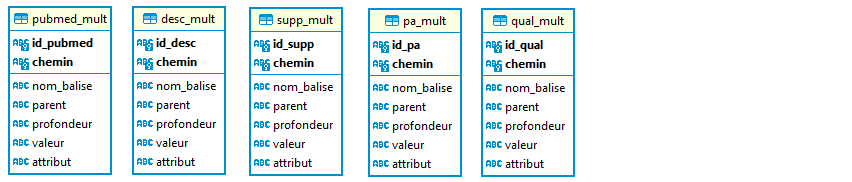


Les champs des tables simples peuvent évoluer du fait qu’ils sont définis à l’aide du référentiel de comptage pour chaque source de donnée (pubmed/mesh). Nous allons identifier les champs clés ou définis par le programme des champs DATA.

Les champs clés ou systèmes :

* Le champ « id\_ », toujours en première position, porte le nom de la source de données (ex : « id\_pubmed » pour les publications)
* Les champs *date* sont placés après l’id et sont du type « date » dans postgres (hormis « pubdate »)
* Les champs suivants sont de la *DATA*
* Le champ *« balises\_error* » est un marqueur afin d’identifier si pour une publication (un id) nous avons des balises non exploitables (ex : les balises de style)

Les tables multiples :



Les champs des tables multiples ont toujours le même format avec le nom du champ id propre à la donnée.

## 6.2- Index et clés primaires postgre

Lors de l’étape de création des tables vides (simples et multiples), pour les données pubmed et référentiels, nous allons créer des clés primaires et index.

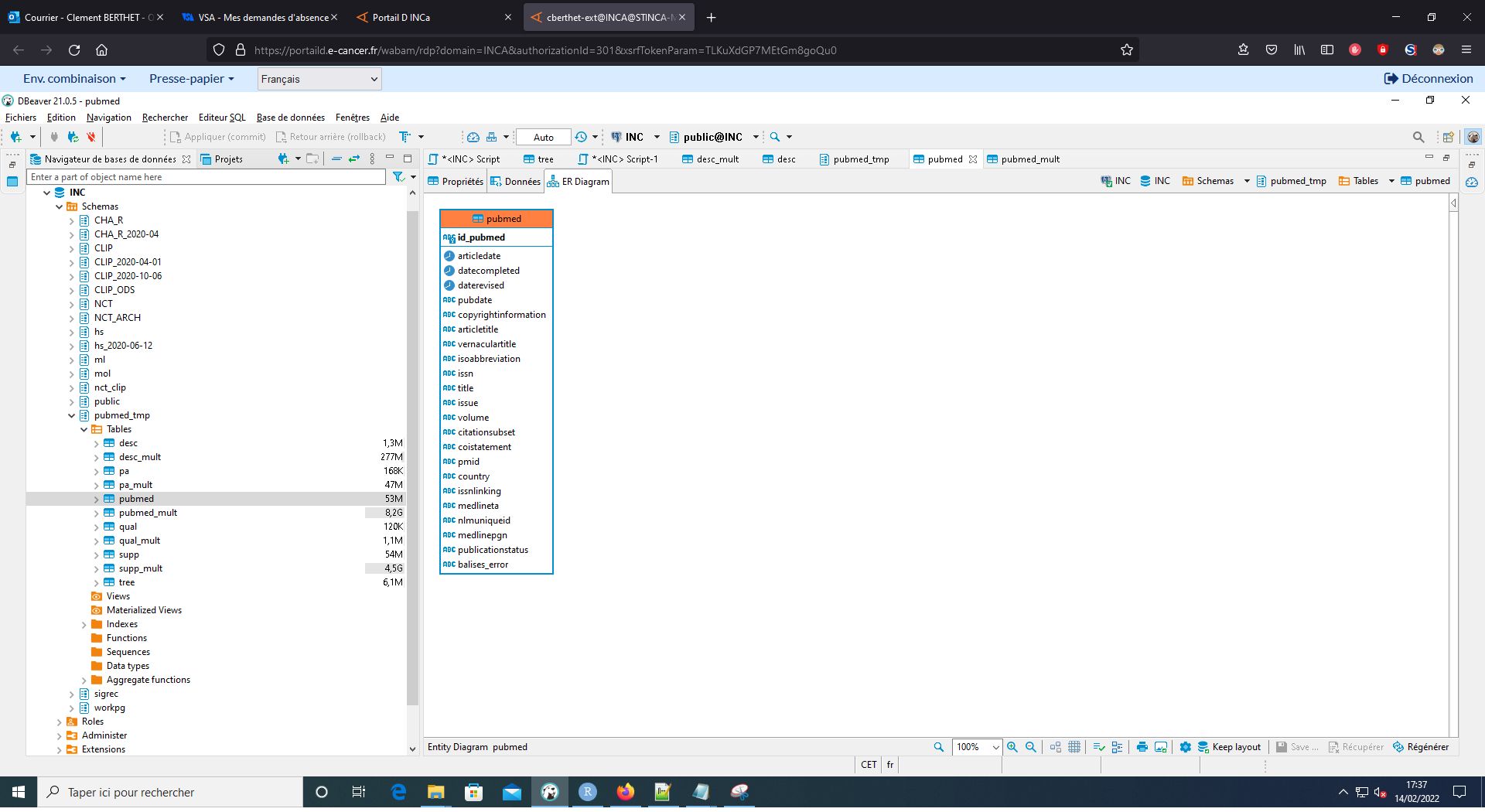
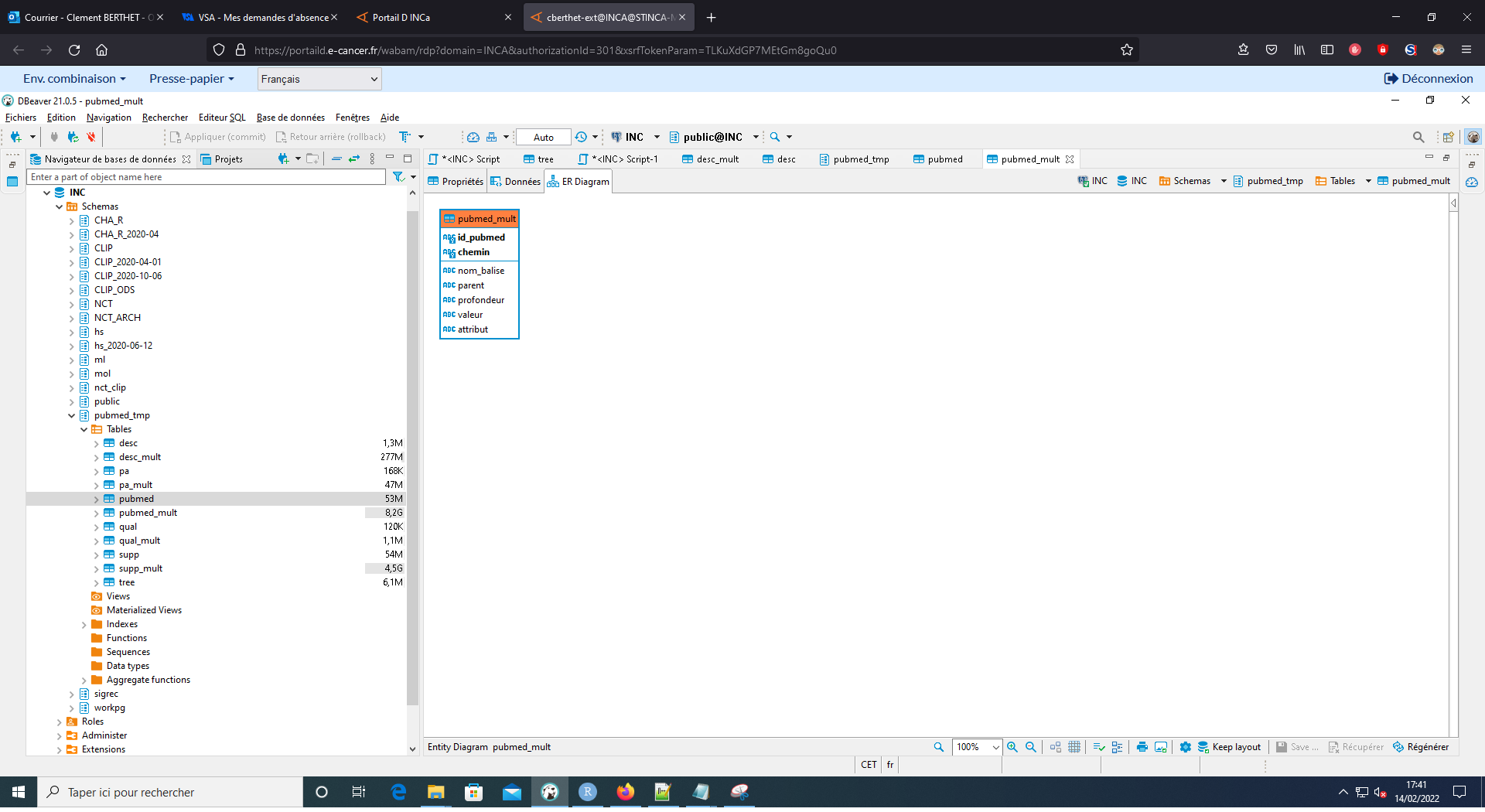
*NOTE* : La création d’une clé primaire implique la création d’un index sur la table.

**TABLE MULTIPLE PUBMED :**

* **Un index (clé primaire) sur l’id de la publication et le chemin**
* **Un index sur le champ id\_pubmed**

**TABLE SIMPLE PUBMED :**

* **Un index (clé primaire) sur l’id unique de la publication**



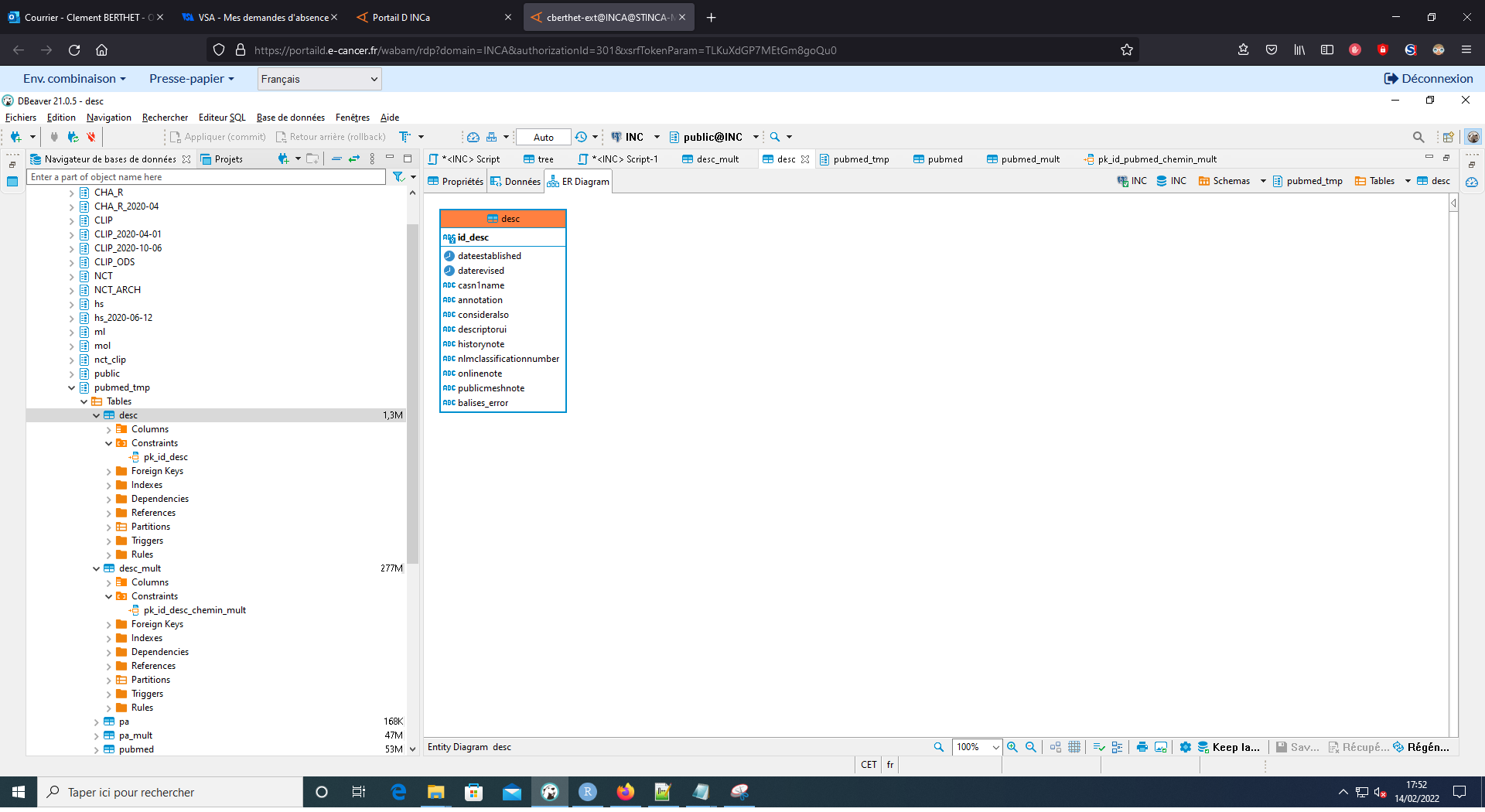
Sur l’ensemble des référentiels, les clés primaires et index sont les mêmes.

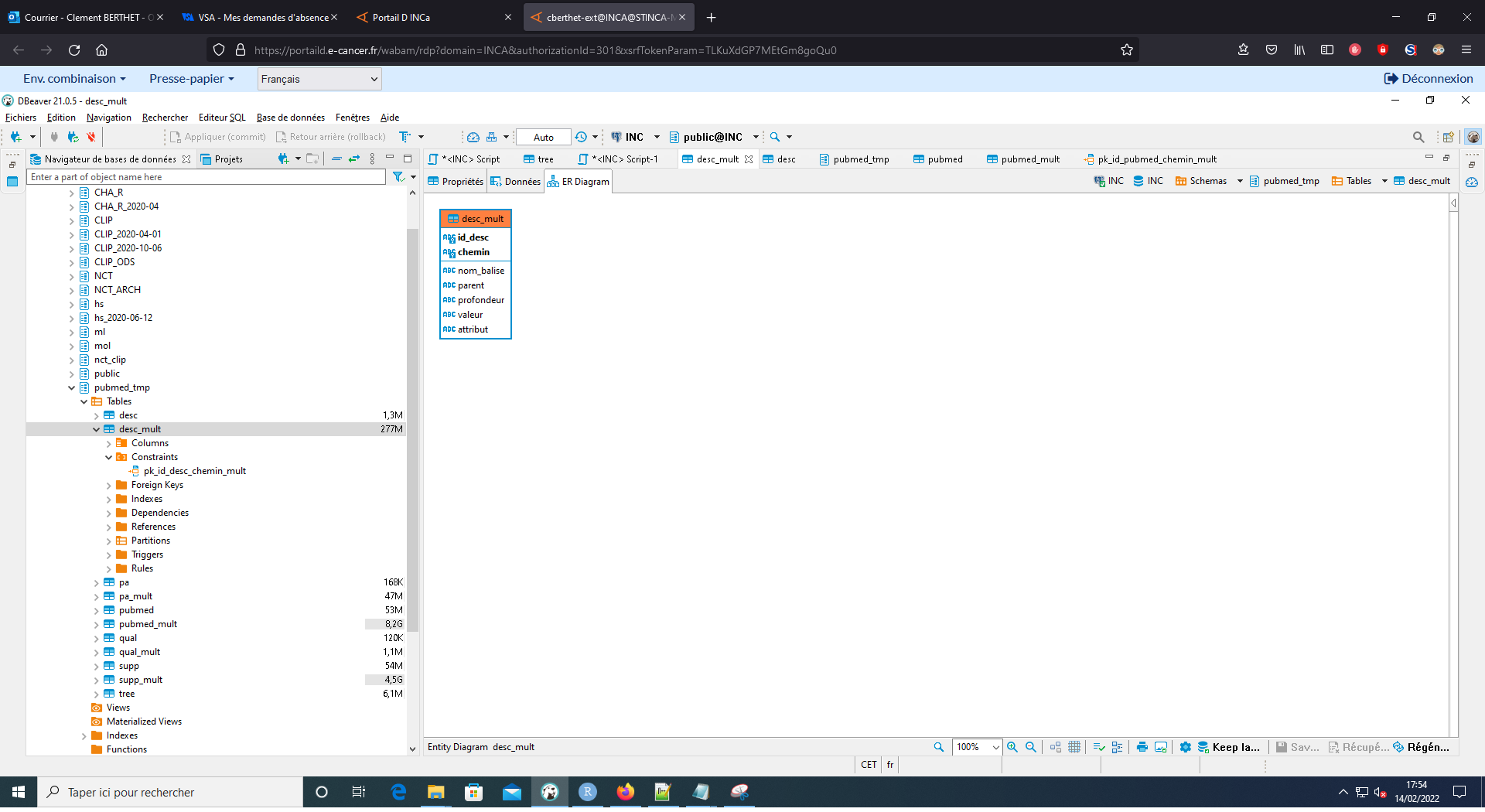
**TABLE MULTIPLE Desc/Supp/Pa/Qual :**

* **Un index (clé primaire) sur l’id du référentiel et le chemin**
* **Un index sur le champ id\_desc/\_supp/\_pa/\_qual**

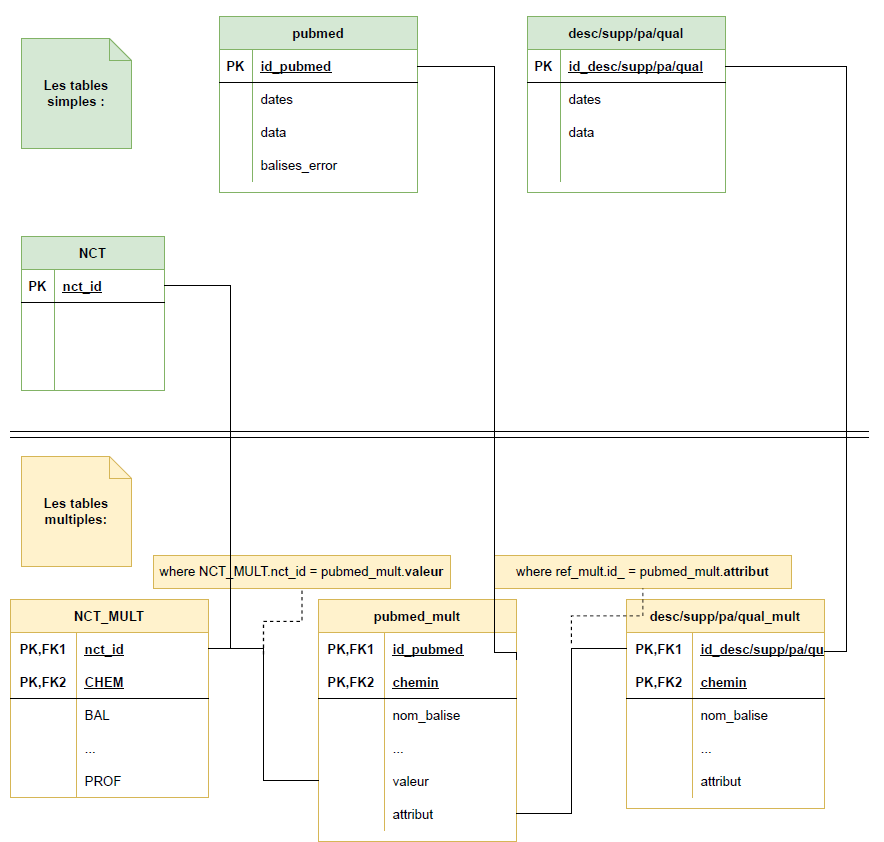
**TABLE SIMPLE Desc/Supp/Pa/Qual :**

* **Un index (clé primaire) sur l’id unique du référentiel**





## 6.3- MCD



*Les cardinalités ne sont pas précisées dans ce schéma, seules les jointures sont illustrées*

Les tables **NCT** et **pubmed\_mult** sont liées par les id *nct\_id* présents dans le champ *valeur* de la table **pubmed\_mult**.

Les tables **pubmed\_mult** etles tables référentiels multiples (**desc\_mult**, **supp\_mult**, …) sont liées par les id des référentiels présents dans le champ *attribut*  de la table **pubmed\_mult**.

## 6.4- Gestion des schémas postgre

Cette partie décrit la mécanique des actions faites par le programme sur la gestion des schémas postgre INCA.

L’ensemble des actions sera traitée par une fonction (*gestion\_schema()*, */Prod/Fonctions/post\_gres/ gestion\_schema.R*). Cette fonction est appelée à la fin des traitements dans le *MAIN*.

Cette fonction est appelée après la vérification et la validation du schéma courant chargé (le schéma temporaire).

**Nous avons trois niveaux de schéma** :

1. **Le schéma temporaire**
2. **Le schéma de prod**
3. **Le schéma archivé**

La fonction de gestion des schémas opère une bascule du niveau du schéma vers un autre.

Temporaire

Archivé

Prod

**Les étapes** :

1. Test de l'existence du schéma temporaire (*pubmed\_tmp*)
2. Test de l'existence du schéma archivé (*pubmed\_arch*)
3. Test de l'existence du schéma de prod (*pubmed*)
4. Gestion de la bascule des schémas de *pubmed\_tmp* vers *pubmed* et de *pubmed* vers *pubmed\_arch*

# 7- CHEMINS RELATIFS

Nous avons vu dans la partie *2* (STRUCTURE PROJET), l’architecture de fichiers nécessaire au fonctionnement du programme. Egalement nous avons vu que le fonctionnement du projet est découpé en blocs, ces blocs correspondent aux différentes catégories de données à traiter :

* Bloc traitements **pubmed**
* Bloc traitements du référentiel **Desc**
* Bloc traitements du référentiel **Supp**
* Bloc traitements du référentiel **Pa**
* Bloc traitements du référentiel **Qual**
* Bloc traitements du référentiel **TREE CODE**

Pour chaque partie, nous avons besoin d’accéder à des fichiers en INPUT ou de produire des fichiers ou exports en OUPUT. L’ensemble des chemins sont créés dans le fichier de paramètres.

**Les principaux chemins** :

Le point d’entrée étant le répertoire ***Prod/*** à la racine du projet.

* Config/ : chemin\_configuration <- "Prod/Config/"
* Fonctions/ : chemin\_fonctions <- "Prod/Fonctions/"
* Input/: DIR\_INPUT\_DATA = "./Prod/Input/"
* Log/: chemin\_log <- "Prod/Log/"
* Output/: DIR\_OUTPUT = "./Prod/Output/"
* Script/: chemin\_script <- "Prod/Script/"

Lorsque l’on créé un nouveau répertoire, nous devons mettre le chemin relatif dans une variable dans le script de paramétrage.