

Lot 2 - Document d’Architecture Technique

|  |  |
| --- | --- |
|  | Intégration consolidée et agrégée des données d'essais cliniques menés dans les centres CLIP dans des bases de données  *> Lot 2 : Chargement des PUBLICATIONS PUBMED ET DES REFERENTIELS MESH* |
| V1.0 |

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

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

Contenu

[1. OBJET DU DOCUMENT 4](#_Toc95754126)

[2. STRUCTURE PROJET 5](#_Toc95754127)

[2.1. Répertoire racine du projet 5](#_Toc95754128)

[2.2. Répertoires de travail 5](#_Toc95754129)

[2.3. Script de Paramétrage 6](#_Toc95754130)

[3. SCRIPT PRINCIPAL – *MAIN* 7](#_Toc95754131)

[3.1. Présentation du *MAIN* 7](#_Toc95754132)

[3.2. Mécanique par BLOC 8](#_Toc95754133)

[3.3. Liste des fonctions 9](#_Toc95754134)

[3.4. Prérequis techniques 13](#_Toc95754135)

[3.5. Gestion des schemas postgre 15](#_Toc95754136)

[3.6 Indexes et clés primaires postgre 15](#_Toc95754137)

[4. FICHIERS INPUT 16](#_Toc95754138)

[5. TABLES POSTGRE 16](#_Toc95754139)

[6. CHEMINS RELATIFS 16](#_Toc95754140)

# OBJET DU DOCUMENT

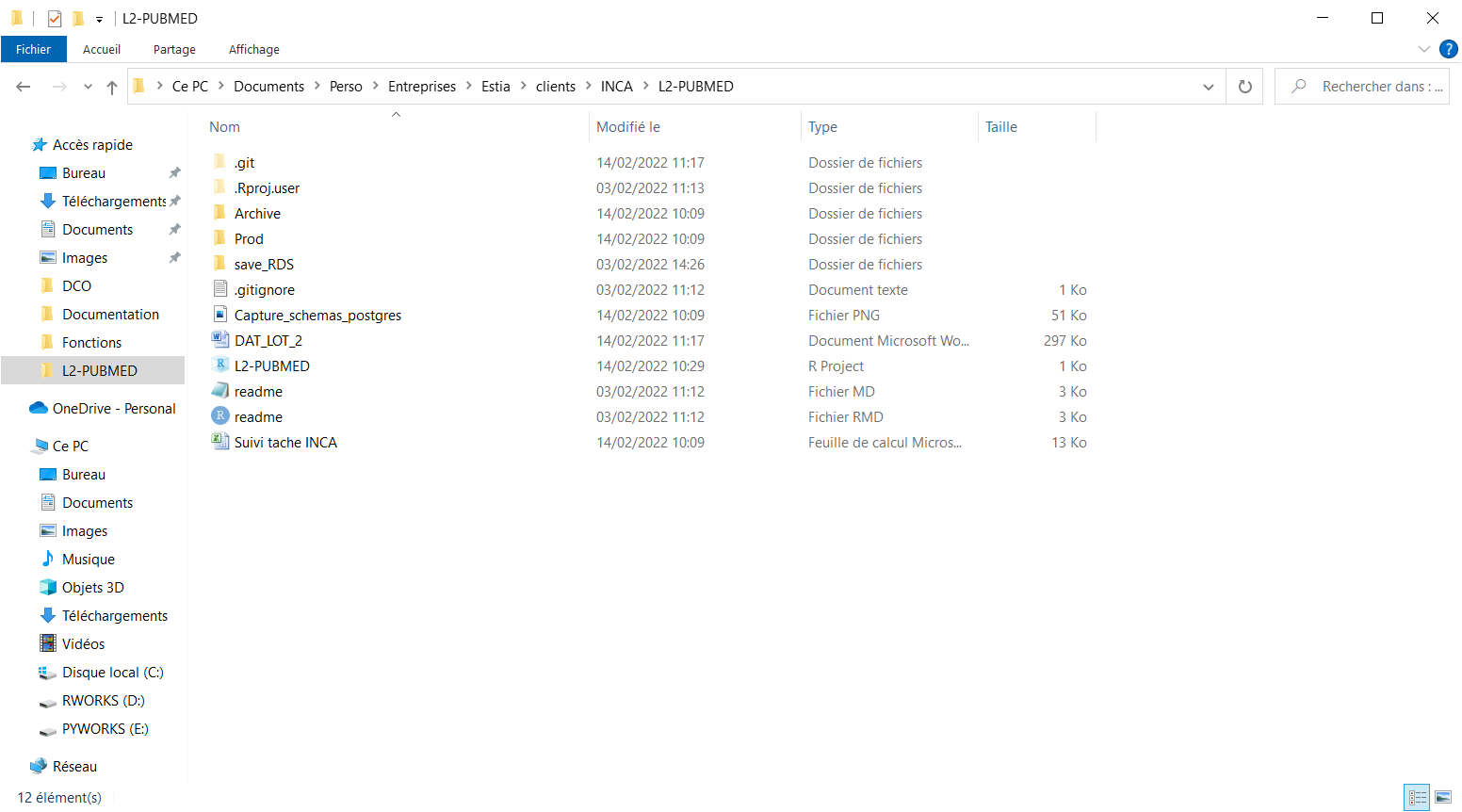
* Présenter le cadre de la mission
* Rapide objectif du document (présenté le projet R)
* Rapide summary du document
* Préciser que ce document sera accompagné d’un MODOP

# STRUCTURE PROJET

## 2.1. 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 :**

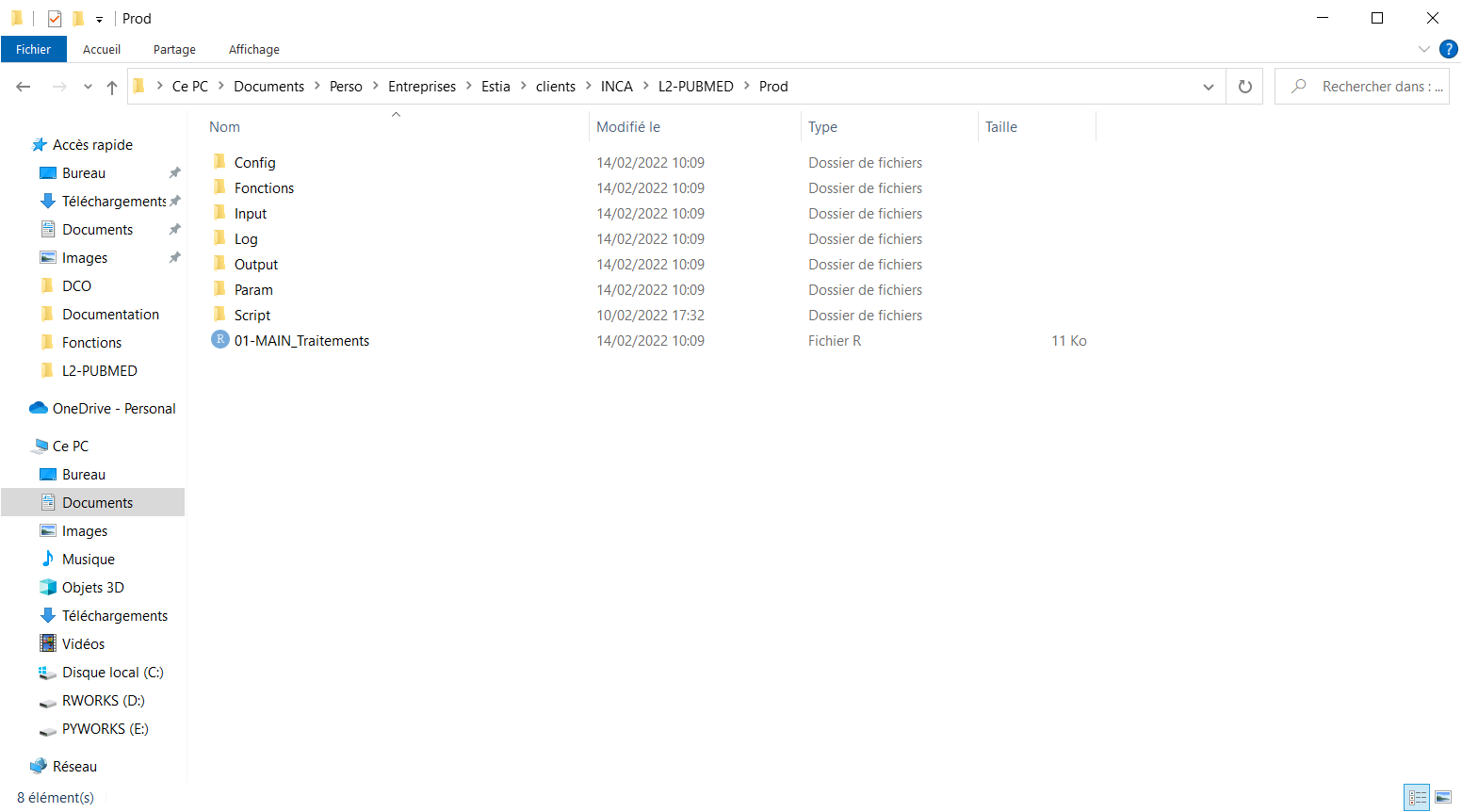


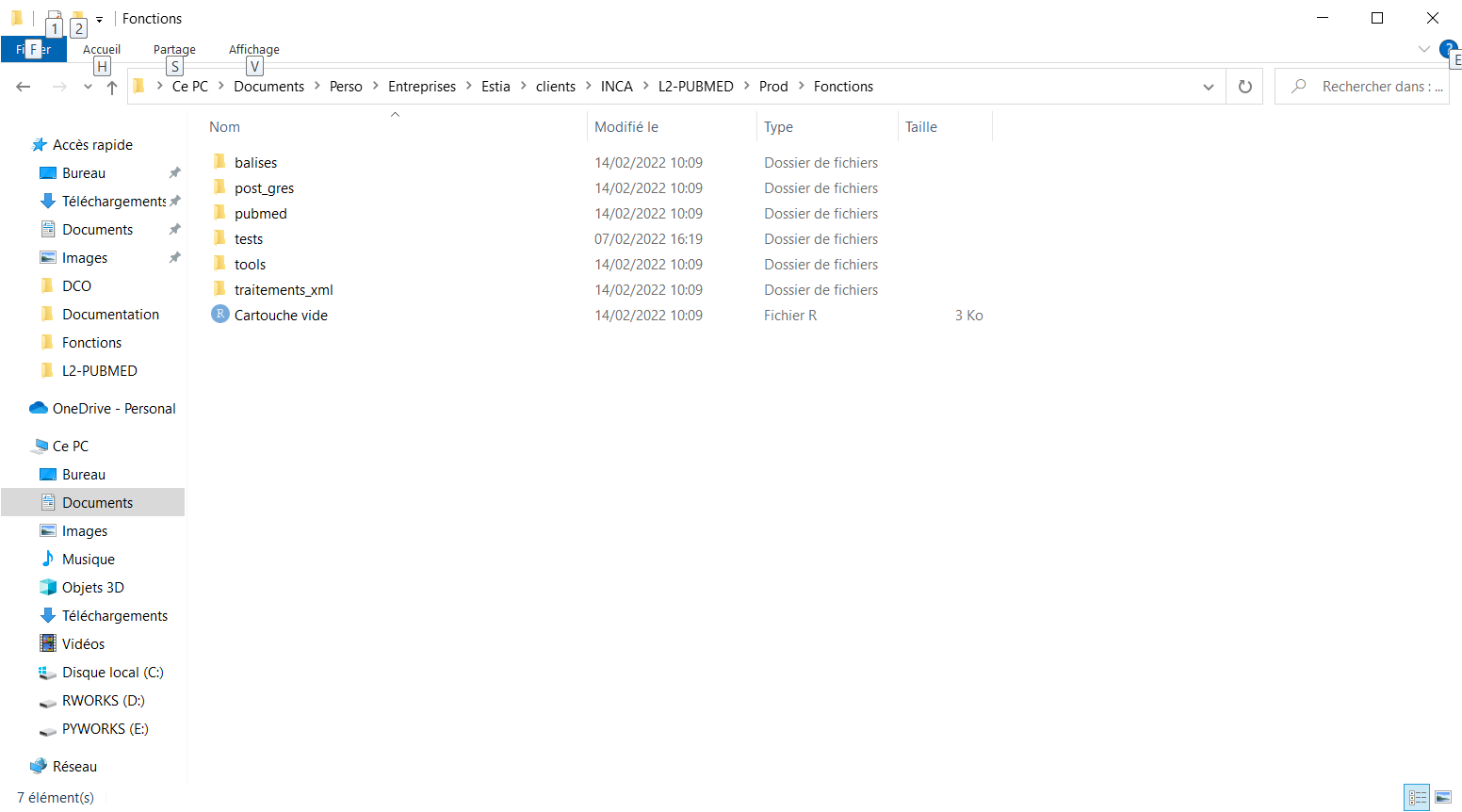
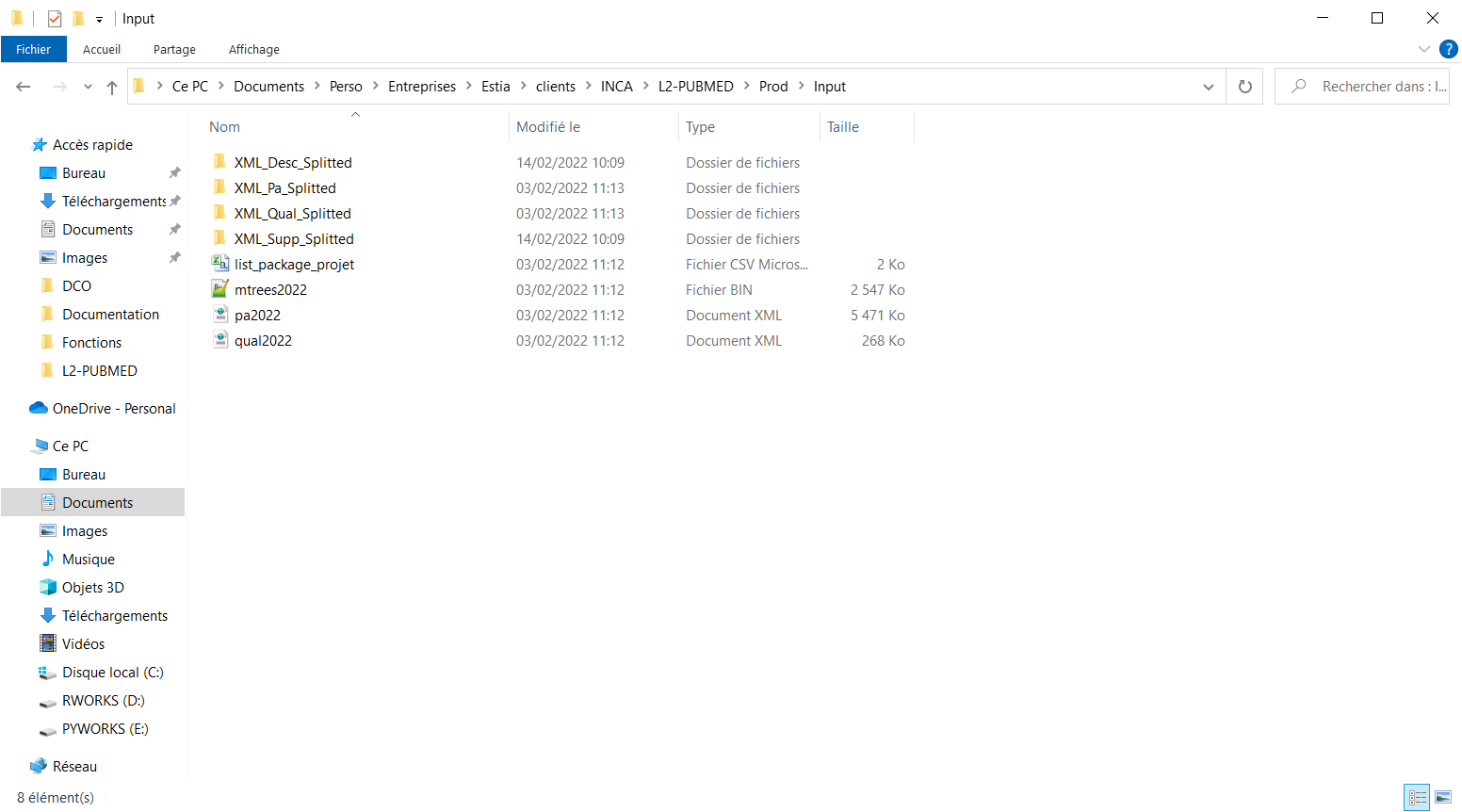
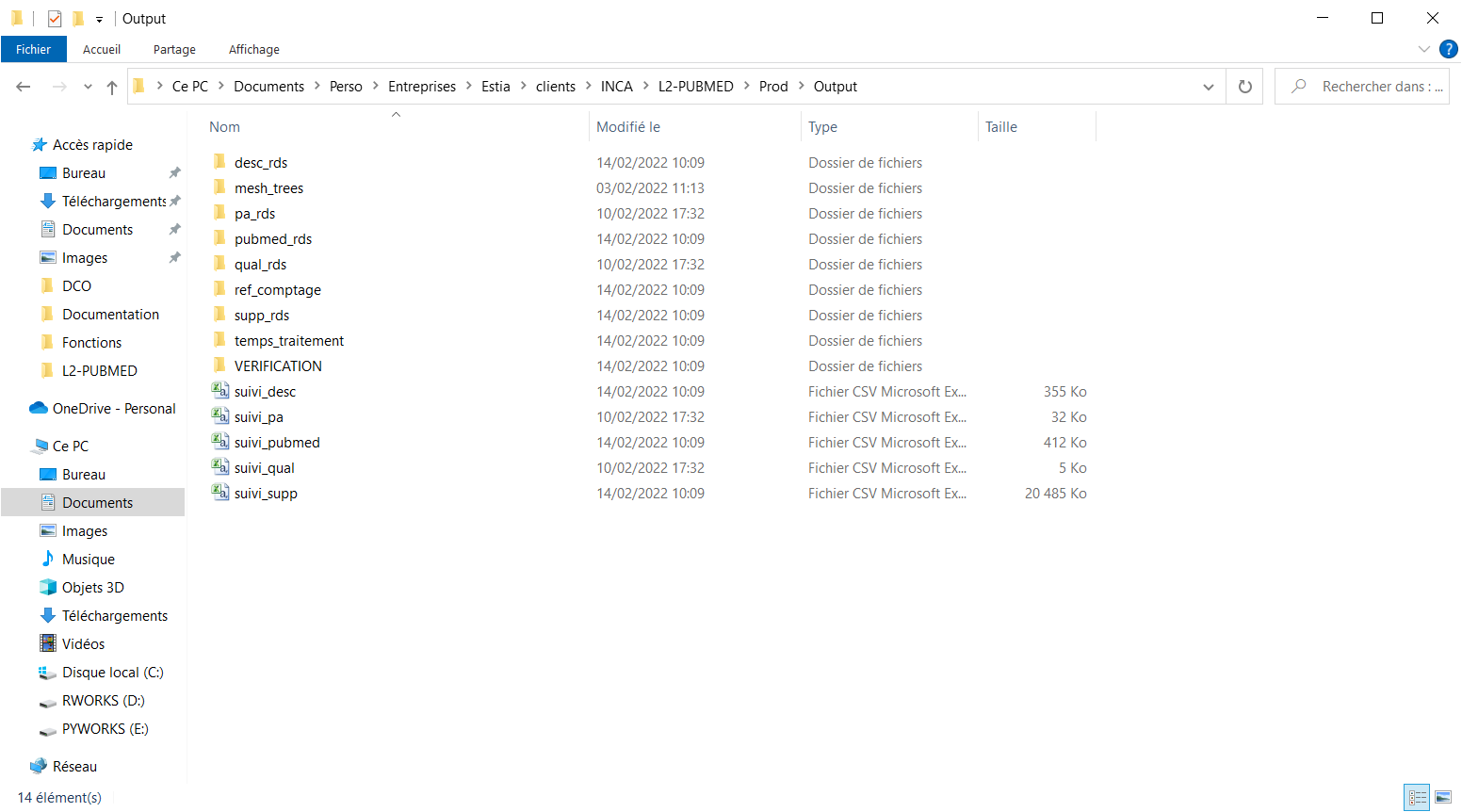
## 2.2. Répertoires de travail

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/ ». Le répertoire « Archive » contient des fonctions dépréciées mais qui peuvent servir de sauvegarde si besoin.

**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
* Input/ : Contient tous les éléments nécessaires au projet
  + - * + Les référentiels MESH (desc, supp, pa, qual)
        + Une arborescence de fichiers dédiées au traitement des référentiels au format XML
        + 
      * Log/ : Contient l’ensemble des fichiers de log nommé par programme et daté
      * Output/ : Contient à la racine des fichiers de suivi (pubmed + référentiels) et une arborescence de sous répertoires
        + 
        + 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/ : déprécié ?

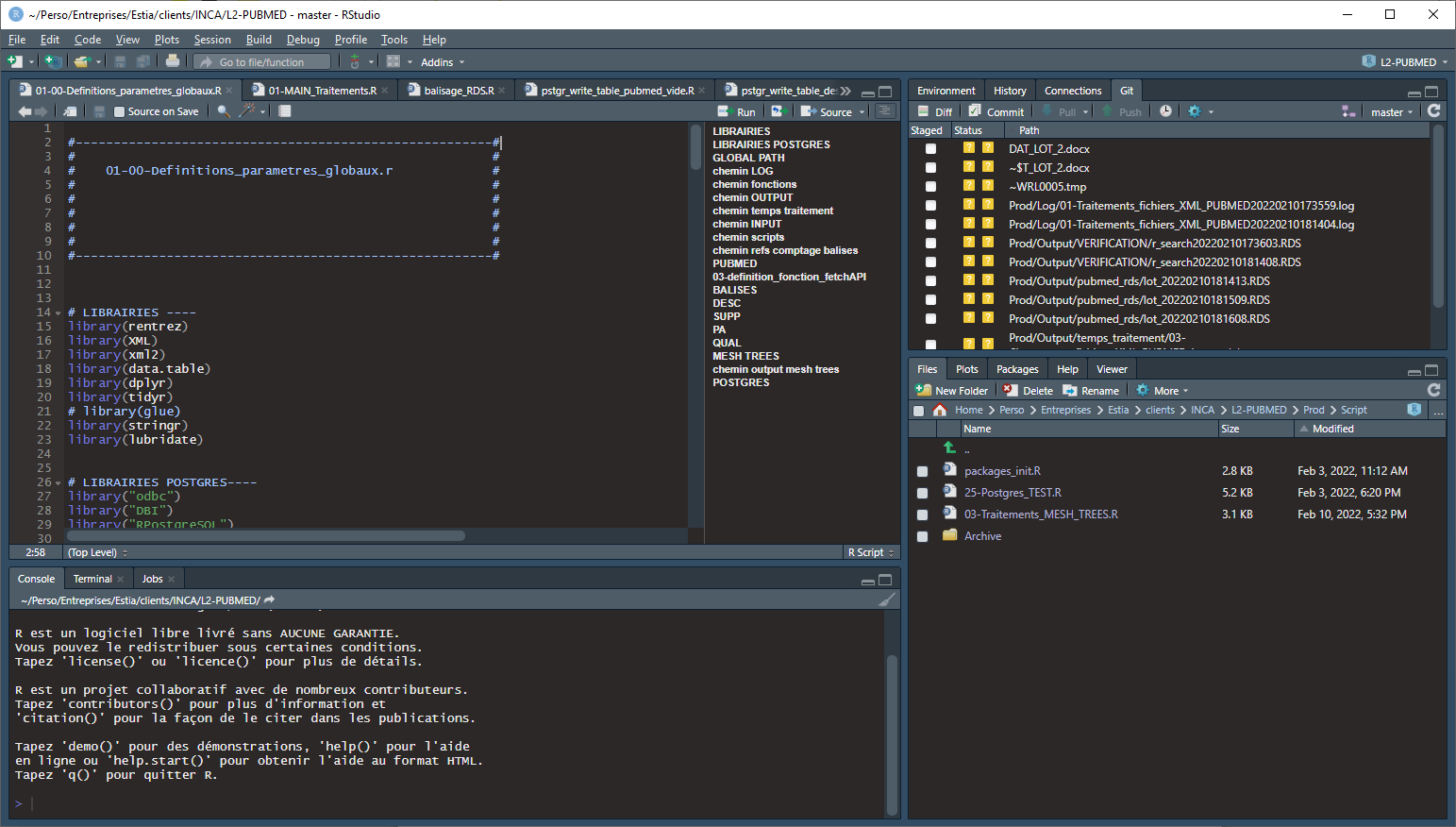
## 2.3. Script de Paramétrage

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 correspond à l’organisation du projet qui sera explicitée en détail dans la partie suivante :

* 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**



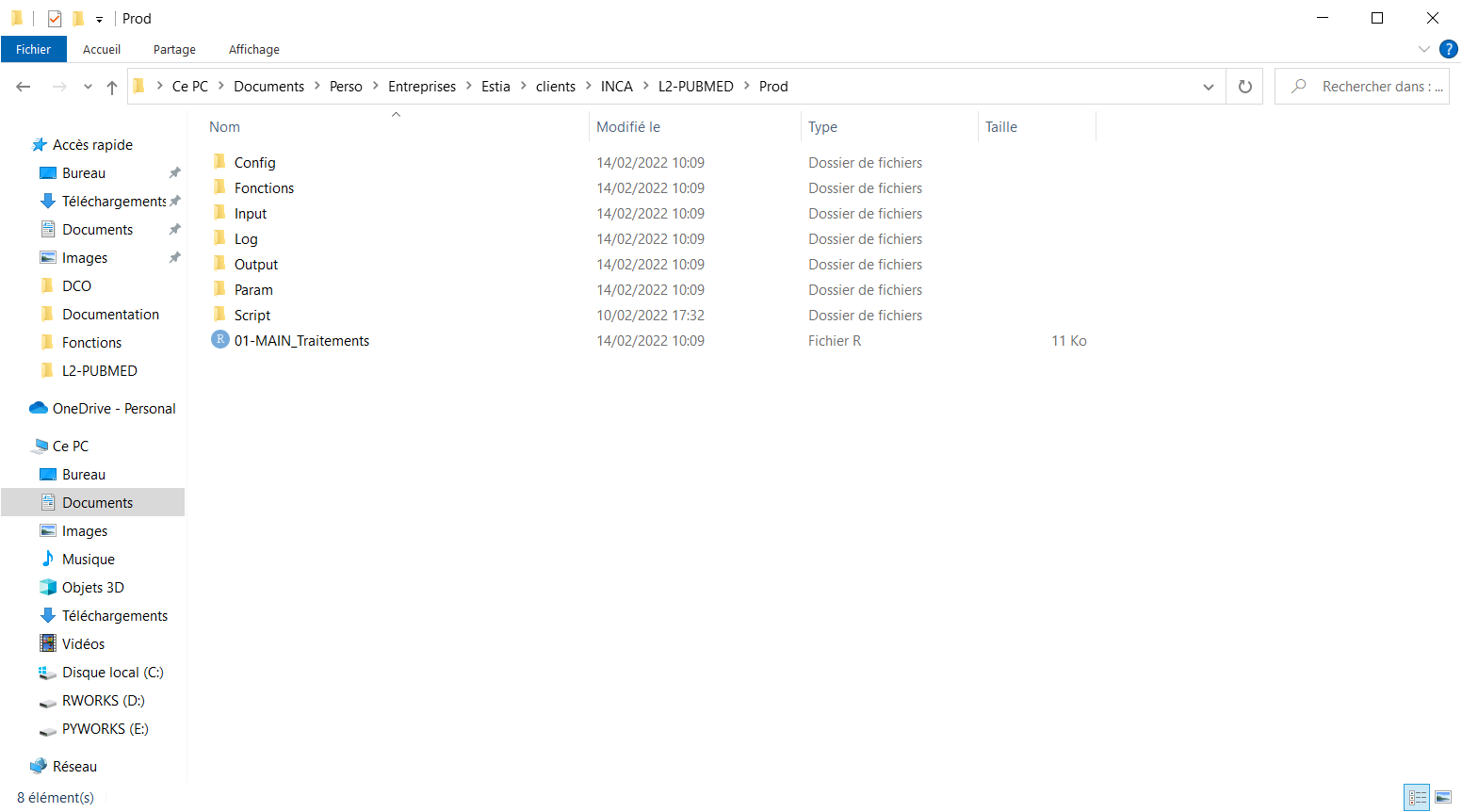
Structure des paramètres par BLOC

Le chargement des packages externes, la création des chemins relatifs à l’environnement interne au projet ainsi que les variables globales sont créées dans ce script.

# SCRIPT PRINCIPAL – *MAIN*

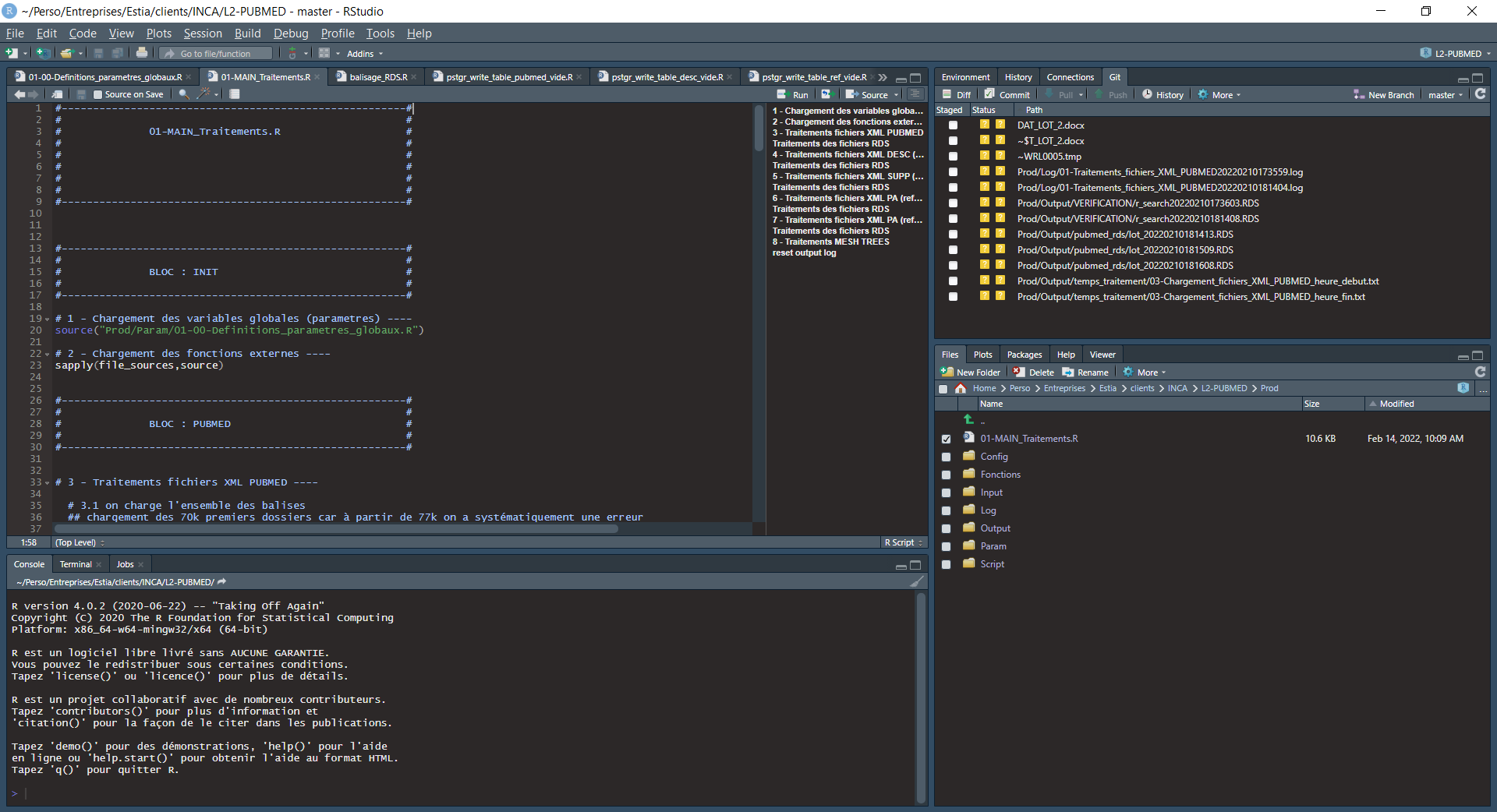
## 3.1. Présentation du *MAIN*

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.



Pour chaque bloc, il faudra exécuter la partie ***INIT***, en début de script, qui permet le chargement en mémoire des paramètres et des fonctions.

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



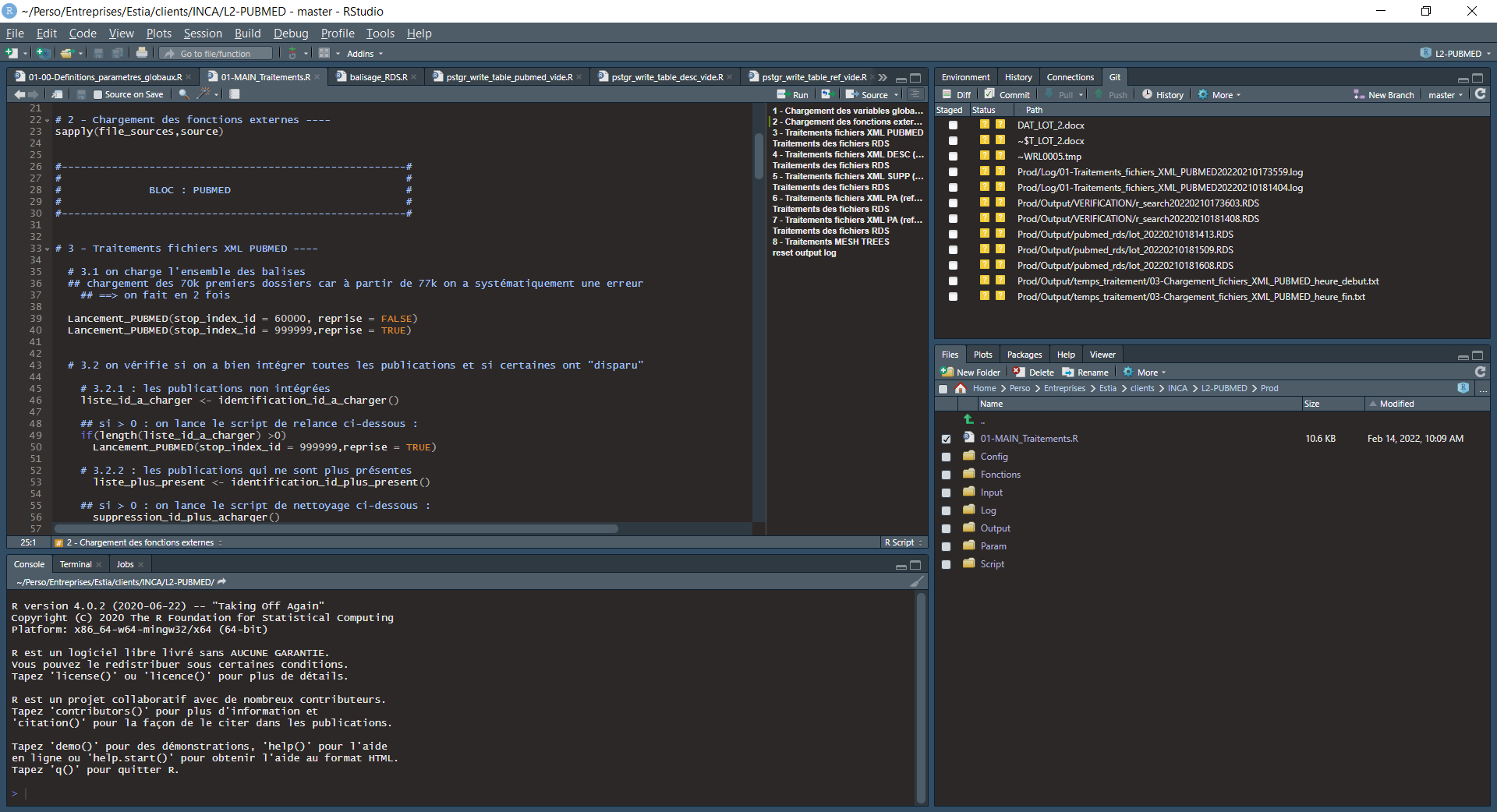
## 3.2. Mécanique par BLOC

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 :



Ce bloc permettra de cibler les publications pubmed via l’API et de les télécharger. Cette partie comprend également une mécanique de suivi des publications téléchargées, des contrôles sur la qualité des données et une mécanique de reprise en cas de problèmes.

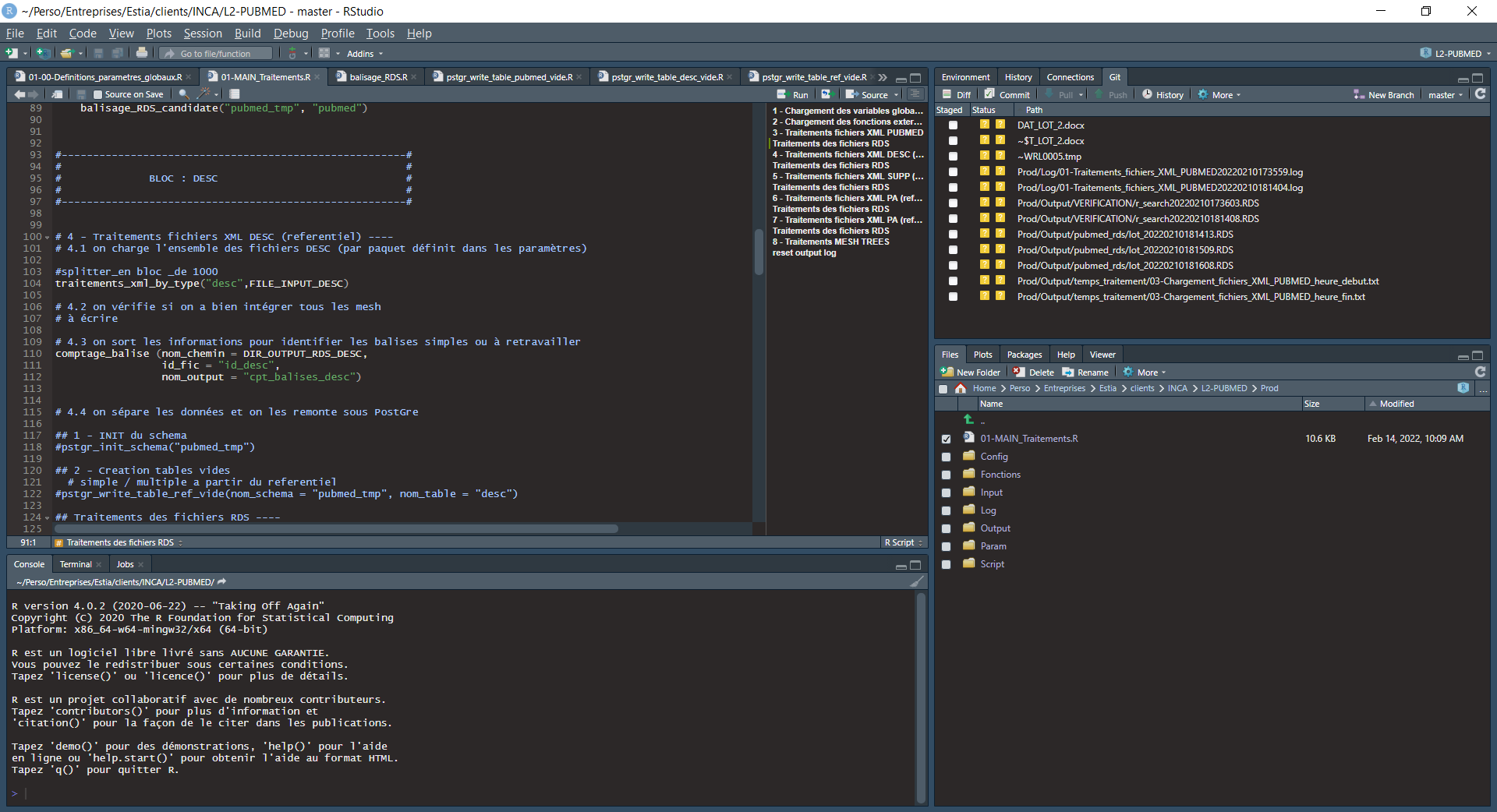
Cette partie comprend la création d’un référentiel de balises qui pourra être challengé et ensuite réintégré. Ce référentiel de comptage est créé à partir de l’ensemble des publications et servira à la création des tables simples et multiples.

La dernière partie du BLOC sera dédiée au chargement en bases postgre (Création du schéma, structure des tables et chargement).

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

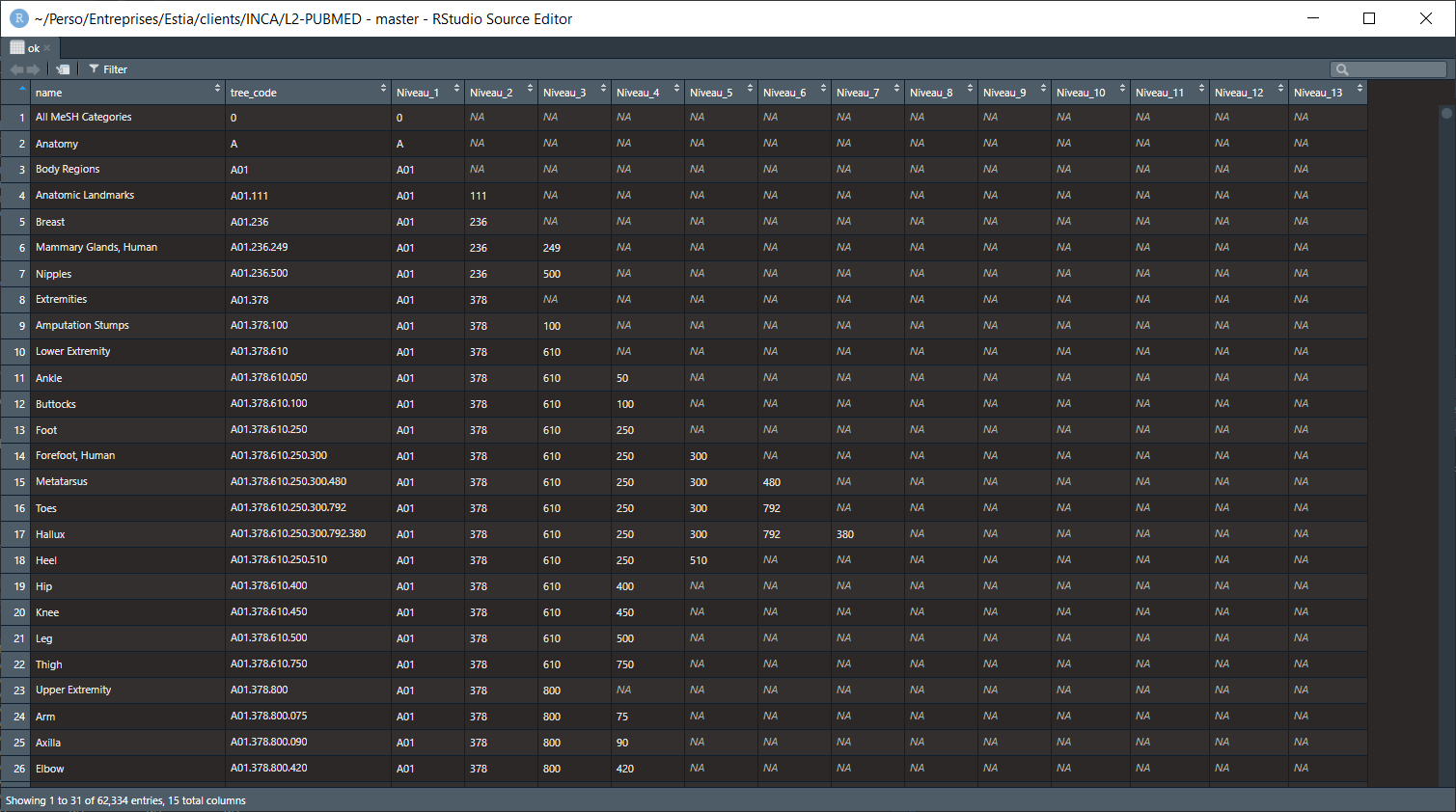
Pour chaque BLOC/référentiel, nous allons trouver dans le script une fonction qui permettra la lecture du référentiel en entrée et son traitement.

Comme pour le BLOC pubmed, nous allons créer un référentiel de balises qui pourra être challengé et réintégré.



En fin de chaque BLOC, nous trouverons une fonction qui se chargera de charger les données dans les tables postgre.

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



Pour chaque code, nous avons le libellé associé ainsi que la décomposition par niveau.

## 3.3. 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 indexes 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 |

## 3.4. Prérequis techniques

L’ensemble du projet a pour but de charger les publications pubmed et les référentiels associés en bases postgre. 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 | GhTHINE2TIVA | INC |

Le projet, pour son bon fonctionnement nécessite l’installation de packages externes :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Package** | **Version** | **Suggests** | **Depends** | **Imports**  Gestion automatique de l’installation des packages via le CRAN (recherche et installation des dernières versions) |
| 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 |

## 3.5. Gestion des schemas postgre

## 3.6 Indexes et clés primaires postgre

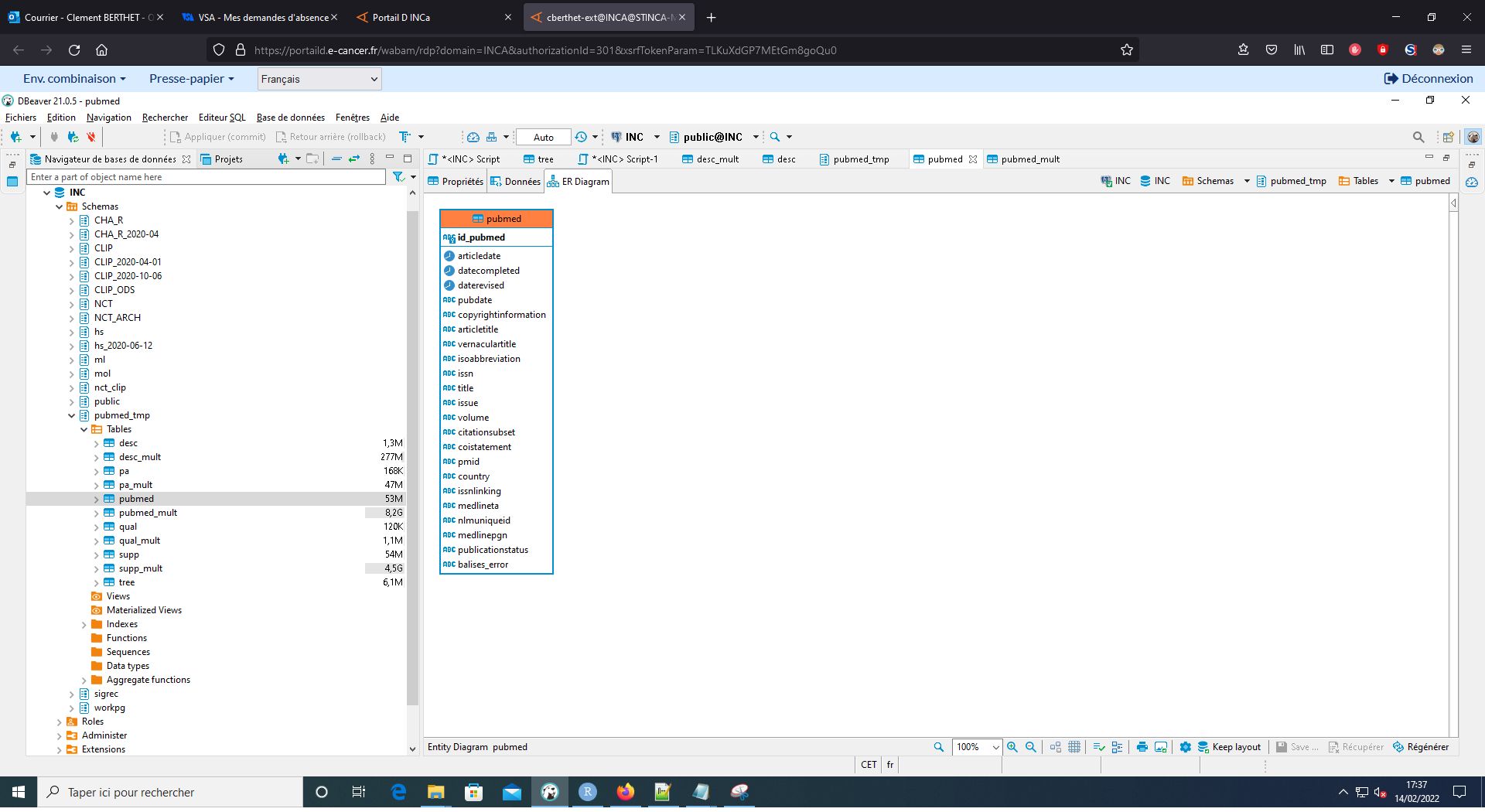
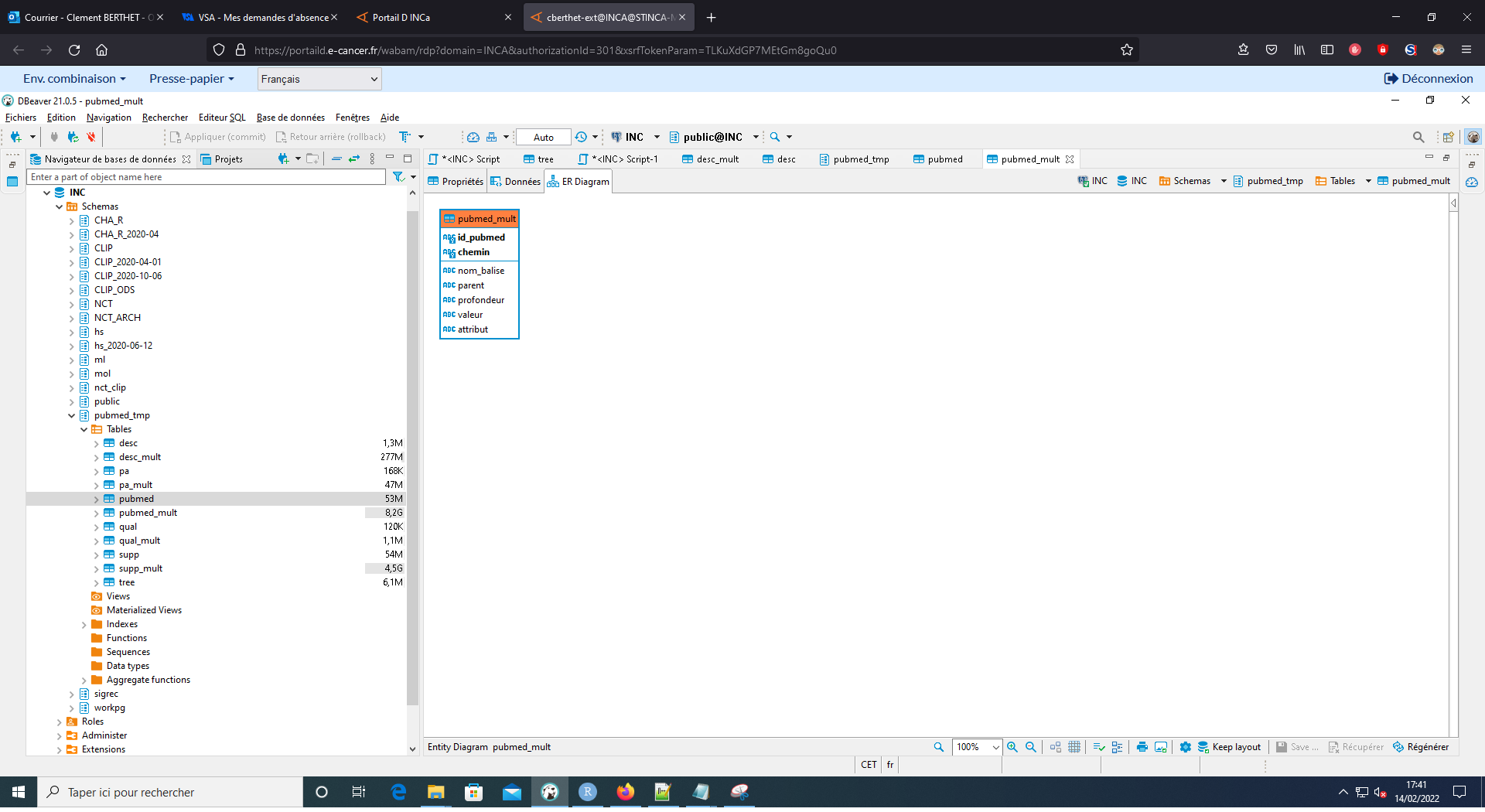
Lors de l’étape de création des tables vides (simples et multiples), pour les données pubmed et référentiels, nous allons créer des clés primaires et indexes.

**TABLE MULTIPLE PUBMED :**

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

**TABLE SIMPLE PUBMED :**

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



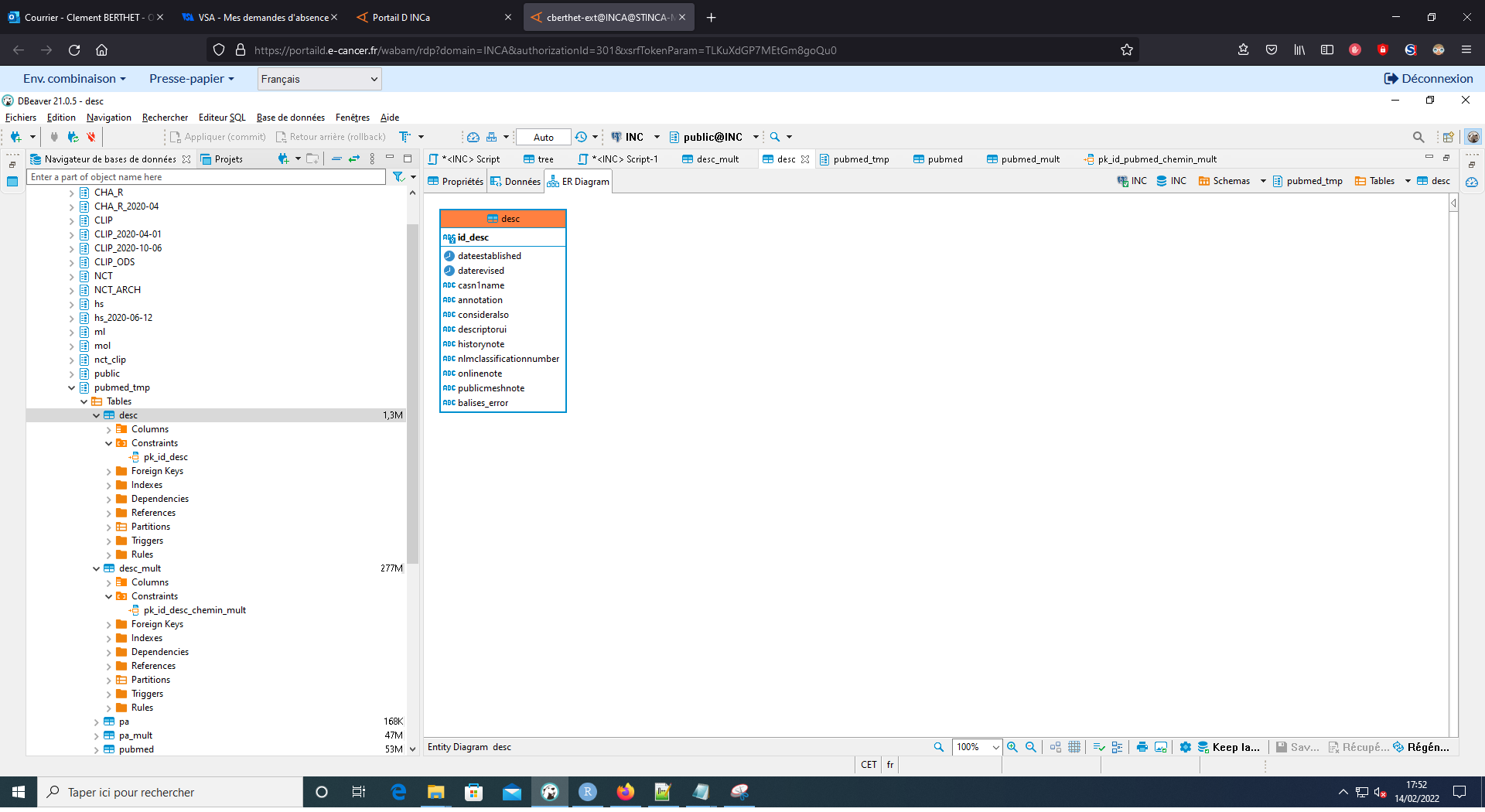
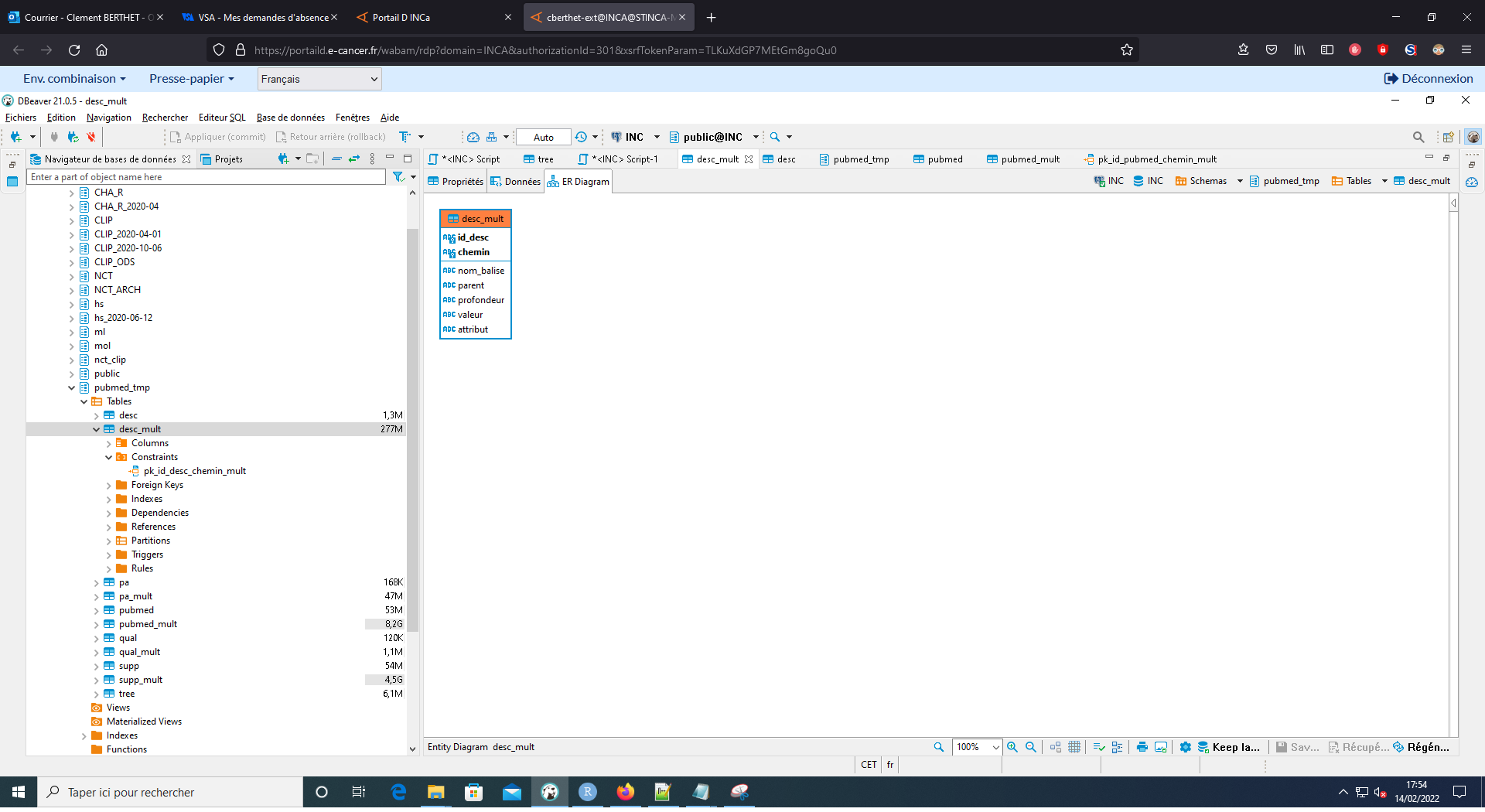
Sur l’ensemble des référentiels, les clés primaires (cela implique de créer un indexe) sont les mêmes.

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

* **Un index (clé primaire) sur l’id du référentiel et le chemin**

**TABLE SIMPLE Desc/Supp/Pa/Qual :**

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



# FICHIERS INPUT

# TABLES POSTGRE

# CHEMINS RELATIFS