Livrable 2

PROJET CLOUD HEALTHCARE UNIT

Oussama ALI-BEY - Alen AMIRBEKYAN

Adam MAHRAOUI - Antonin RUDONI

CESI

ÉCOLE D'INGÉNIEURS

FISA INFO 23-26

Table des matières

1. Introduction

Le domaine de la santé évolue à un rythme très rapide, et les institutions médicales sont confrontées à la nécessité de s'adapter à de nouvelles techniques plus digitales. Dans cette perspective, le groupe CHU (Cloud Healthcare Unit) se trouve à un moment critique, cherchant à établir un entrepôt de données efficace pour gérer le flux continu de données de ses systèmes de gestion des soins et des systèmes FTP. Le but de ce projet est de concevoir une solution intégrale qui permette l'extraction, le stockage, l'intégration, l'exploration et la visualisation des données médicales de manière optimale. En collaboration avec le groupe CHU, nous devons identifier et répondre aux besoins spécifiques des praticiens et des administrateurs d'établissements, en leur fournissant des recommandations sur les outils d'intégration, de stockage et de visualisation les plus adaptés. Nos solutions vont permettre de contribuer à améliorer l'efficacité des services de santé en exploitant intelligemment leurs données.

2. Descriptions des JOBS dans Talend

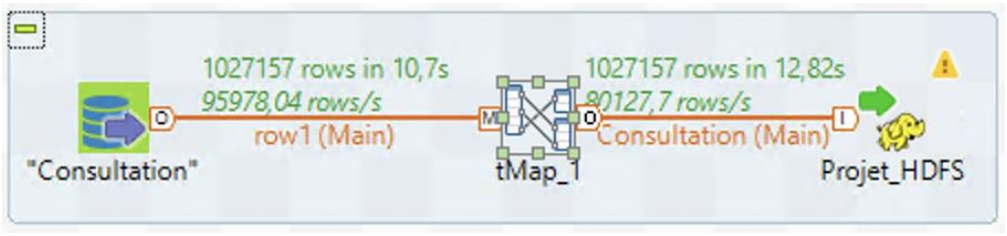


Figure 1: Capture d'écran de Talend du Jobs pour la table Consultation

Figure 2 : Capture d'écran du tMap du Jobs pour la table Consultation

Ce job permet de récupérer la date de consultation depuis la base de données PostgreSQL au format date JJ-MM-AAAA et de la découper en 3 entiers : jour_consultation, mois_consultation, annee_consultation. Il permet également de changer le nom des colonnes pour être en accord avec notre schéma.

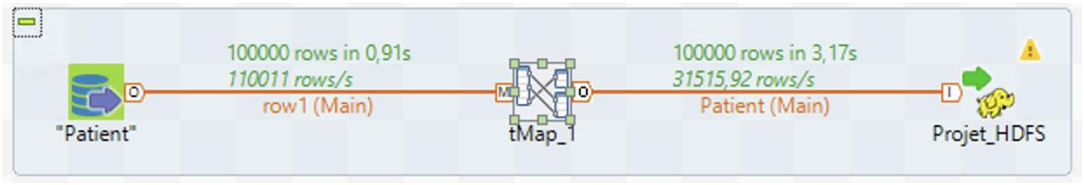


Figure 3 : Capture d'écran de Talend du Jobs pour la table Patient

Ce job permet de vérifier la conformité de la date de naissance du patient (jour < 32, mois < 13, année < année actuelle) et la sépare en 3 entiers : jour_naissance, mois_naissance, annee_naissance. Il vérifie aussi que le sexe du patient est sous la forme de « f » ou « m » et convertit les données de la source dans les bons types attendus dans les colonnes.

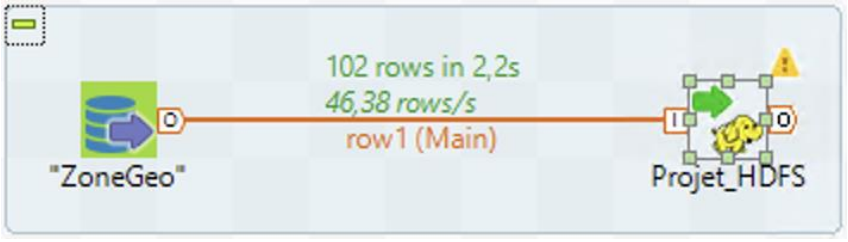


Figure 4 : Capture d'écran de Talend du Jobs pour la table ZoneGeo

Ce job permet de relier un code de département à son nom de région (54 -> Grand-Est) pour la table ZoneGeo. Nous avons créer cette table a la main en tannant compte des nouvelles régions de 2016.

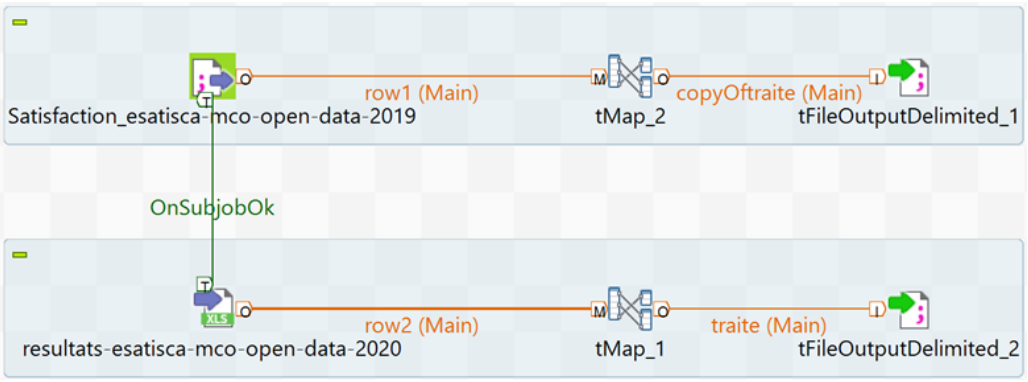


Figure 5 : Capture d'écran de Talend du Jobs pour la table Satisfaction

Ces deux Jobs permettent d'alimenter la table Satisfaction en prenant comme source 2 « resultats-esatisca-mco-open-data-2019 » (fichier CSV) et « resultats-esatisca-mco-open-data-2020 » (fichier Excel).

Les deux tMaps permettent de supprimer les lignes ou il n'est pas mentionné le taux global de satisfaction (score_all_ajust) et d'obtenir en sortie 4 colonne : Id_etablissement (''F'' + finess_geo), Satisfaction (score_all_ajust), Departement(les 2 premières lettre de finess_geo) et enfin une colonne Année. Les 2 fichiers d'entré par un Trigger OnSubjobOk pour que le job commence par le premier fichier et puis passe au second fichier.

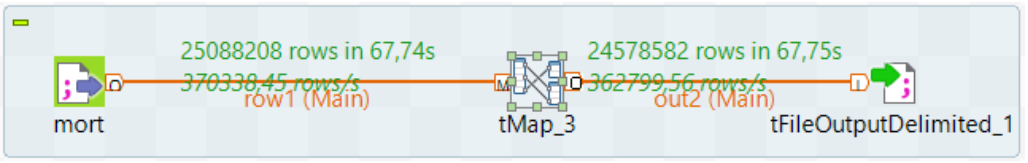


Figure 6 : Capture d'écran de Talend du Jobs pour la table Décès

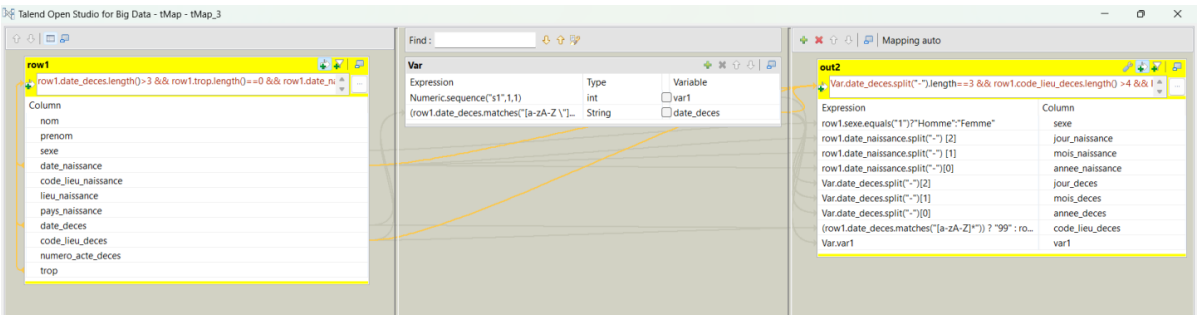


Figure 7 : Capture d'écran du tMap du Jobs pour la table Deces

Ce job permet de transformer les données des décès et d'ajouter un identifiant décès. On vérifie le nombre de colonnes dans chaque ligne (il y a parfois des virgules dans les adresses, ce qui ajoute des colonnes, donc nous ne prenons pas en compte ces lignes car leur quantité est négligeable). Ensuite, nous prenons en compte tous les décès compris entre 1972 et 2019 car en dehors de cette période, les données des décès ne sont pas complètes. Nous transformons le sexe de 1 et 2 en h et f, puis nous testons si les dates sont conformes (format, problème de mois...).

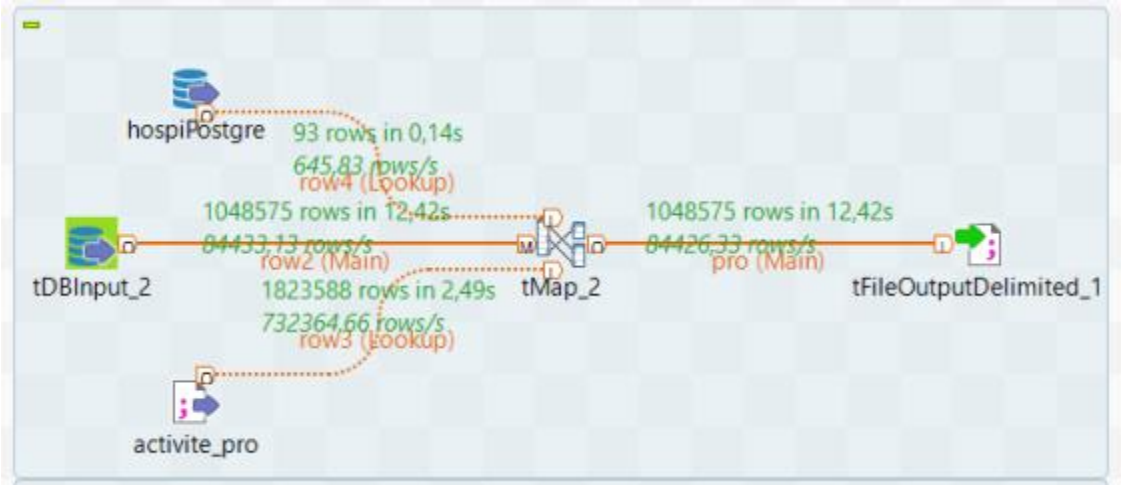


Figure 8 : Capture d'écran de Talend du Jobs pour la table Professionnel

Ce job permet de lier Professionnel avec sa spécialité (dans la Base de donnée Postgres) ainsi qu'a Etablissement (provient d'un fichier CSV). Ce JOB permet d'avoir une approche plus précise en liant un professionnel avec sa spécialité, sa profession et sa categorie_pro.

3. Gestion des données dans les tables

a) Création et le chargement de données

Pour charger les données dans HIVE nous avons utilisé cette structure de scripte pour toutes les tables. Dans un premier temps on copie le fichier qu'on utilisera comme source (celui-ci provient des jobs Talend) depuis le répertoire local vers le système de fichiers Hadoop (HDFS) dans le répertoire. Cela nous permet de rendre le fichier accessible aux opérations Hive. Ensuite on crée une table externe dans Hive pour accéder à ces données. Les colonnes de la table sont définies en fonction du contenu du fichier et les paramètres de délimitation sont spécifiés pour traiter correctement les données et avoir les bonnes données dans les bonnes colonnes. Les données sont stocké dans des .txt dans le répertoire « /user/hive/data » et chargé via l'interface graphique HUE.

Voici un exemple de la procédure pour effectuer le chargement des données :

```
hadoop fs -copyFromLocal /home/cloudera/Patient.txt /user/hive/data
CREATE EXTERNAL TABLE IF NOT EXISTS log_patient
(
 id_patient INT,
 genre STRING,
 jour_naissance INT,
 mois_naissance INT,
 annee_naissance INT,
 taille INT,
 poids FLOAT,
 groupe_sanguin STRING
)
COMMENT 'table intermediaire'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/data';
```

b) Vérification et accès aux données

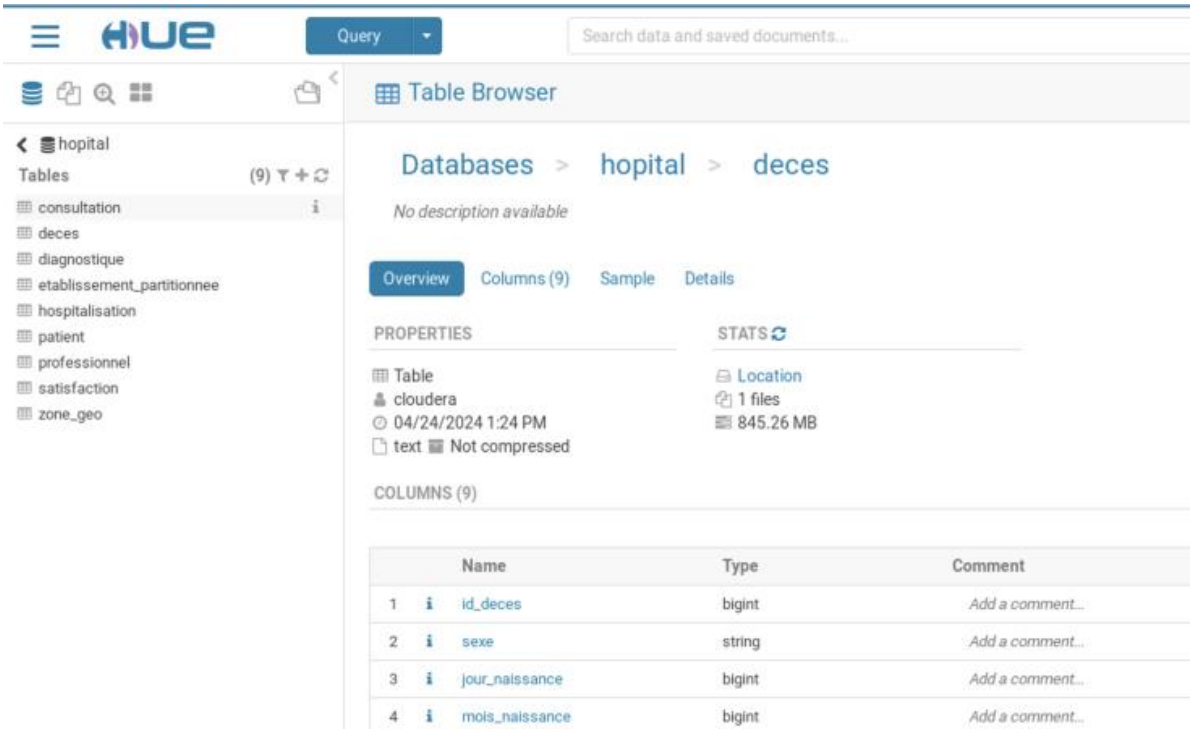


Figure 9 : Capture d'écran de la table deces chargé la database

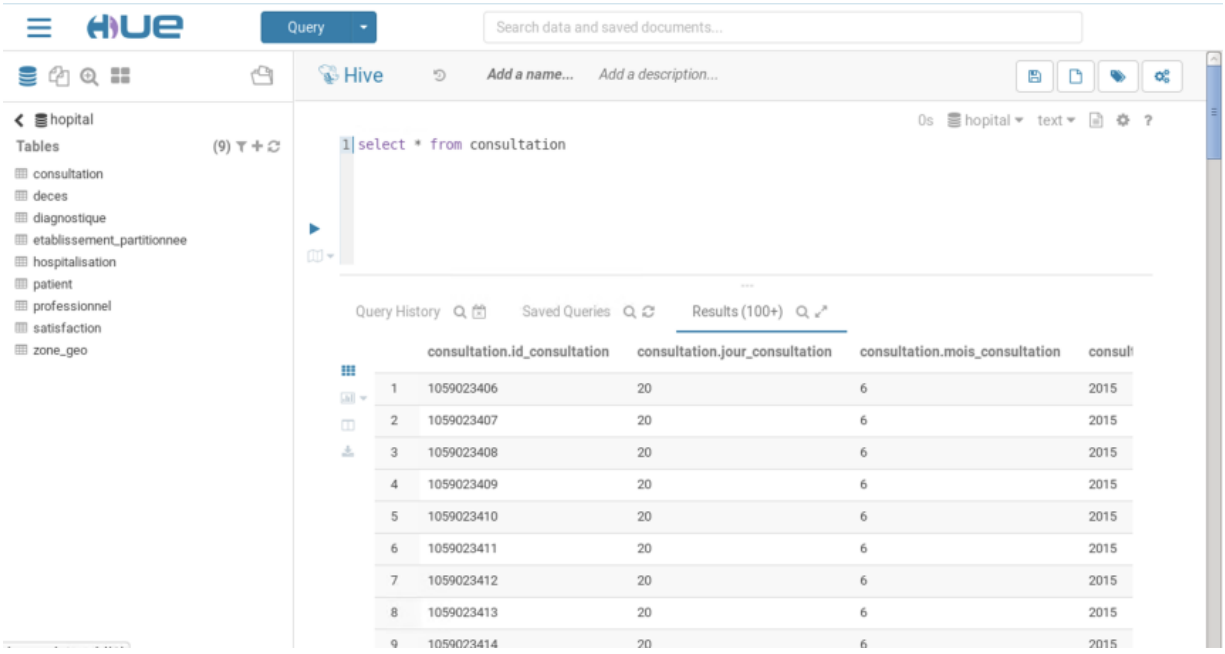
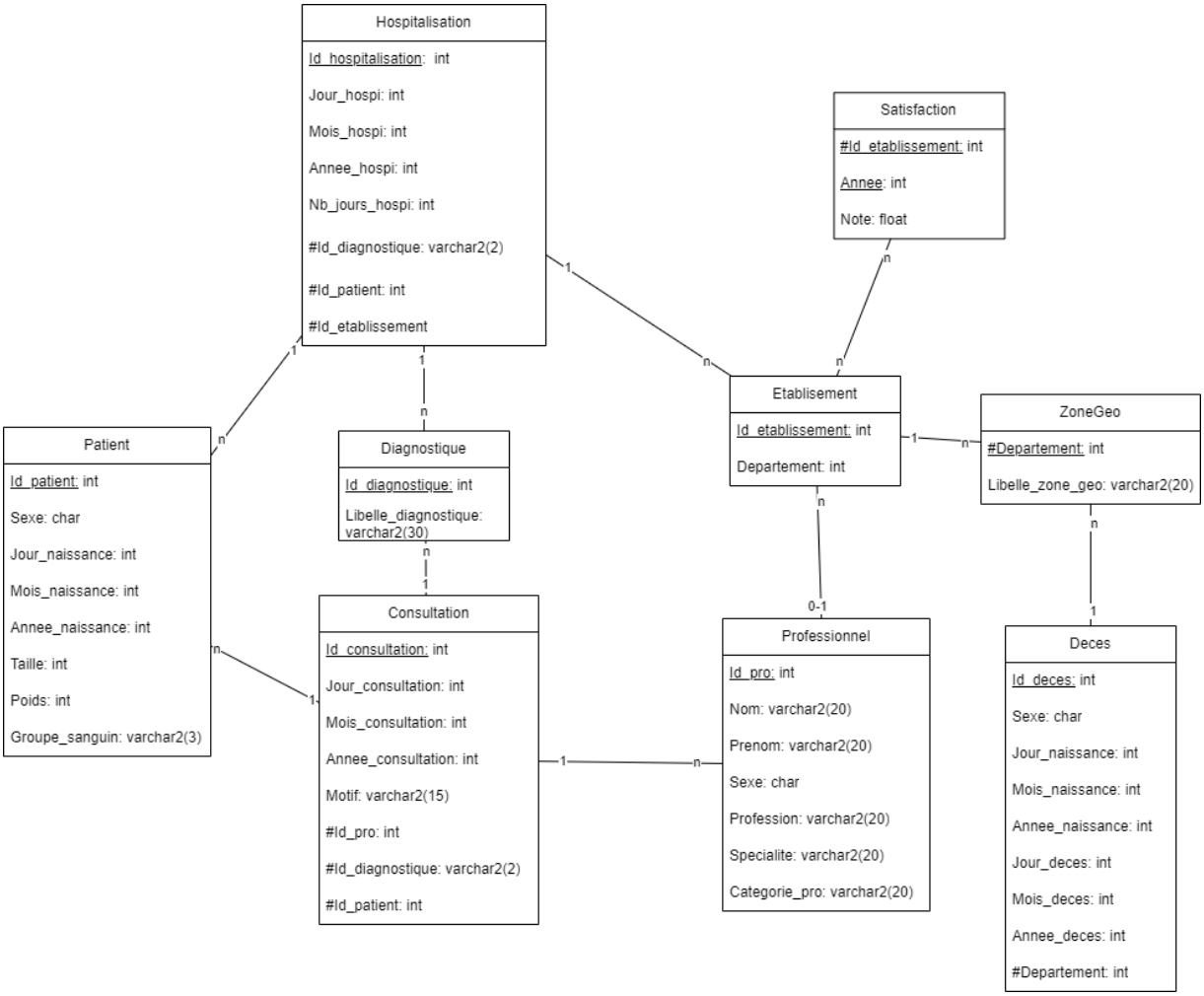


Figure 10 : Capture d'écran de l'interface HUE montrant les données de la table Consultation

c) Peuplement des tables



Schema du Modèle Conceptuel des Données du projet

4. Gestion des performances

a) Partitionnement et buckets

```
CREATE TABLE etablissement_partitionnee
(
    id_etablissement STRING,
    nom_etablissement STRING
)
PARTITIONED BY (departement BIGINT);

INSERT OVERWRITE TABLE etablissement_partitionnee
PARTITION (departement)
SELECT id_etablissement, nom_etablissement, departement
FROM etablissement;
```

Figure 11 : Requête pour le partitionnement de la table Etablissement

```
CREATE TABLE consultation_partitionnee
(
    id_consultation BIGINT,
    jour_consultation BIGINT,
    mois_consultation BIGINT,
    annee_consultation BIGINT,
    motif STRING,
    id_pro BIGINT,
    id_patient BIGINT
)
PARTITIONED BY (id_diagnostique STRING)
CLUSTERED BY (annee_consultation) INTO X BUCKETS;
-- X = nombre d'année de consultation dans la base de donnée,
-- il faudra penser a l'augmenter chaque année.
-- (certains logiciels le font de maniere automatique)

INSERT OVERWRITE TABLE consultation_partitionnee
PARTITION (id_diagnostique)
SELECT id_consultation, jour_consultation, mois_consultation, motif, id_pro, id_diagnostique, id_patient, annee_consultation
FROM consultation;
```

Figure 12 : Requête pour le partitionnement de la table Consultation

Le partitionnement par diagnostic est utile quand on doit filtrer ou analyser les données en se concentrant sur certains diagnostics. En organisant les données de cette manière cela facilite l'extraction des informations nécessaires et leur traitement pour des analyses précises et optimisées dans PowerBI.

```
CREATE TABLE deces_partitionnee (
    id_deces BIGINT,
    sexe STRING,
    jour_naissance BIGINT,
    mois_naissance BIGINT,
    annee_naissance BIGINT,
    jour_deces BIGINT,
    mois_deces BIGINT,
    code_lieu_deces BIGINT
)
PARTITIONED BY (annee_deces BIGINT)
CLUSTERED BY (code_lieu_deces) INTO 10 BUCKETS;


INSERT OVERWRITE TABLE deces_partitionnee
PARTITION (annee_deces)
SELECT id_deces, sexe, jour_naissance, mois_naissance, annee_naissance, jour_deces, mois_deces, annee_deces, code_lieu_deces, annee_deces
FROM deces;
```

Figure 13 : Requête pour le partitionnement de la table Deces

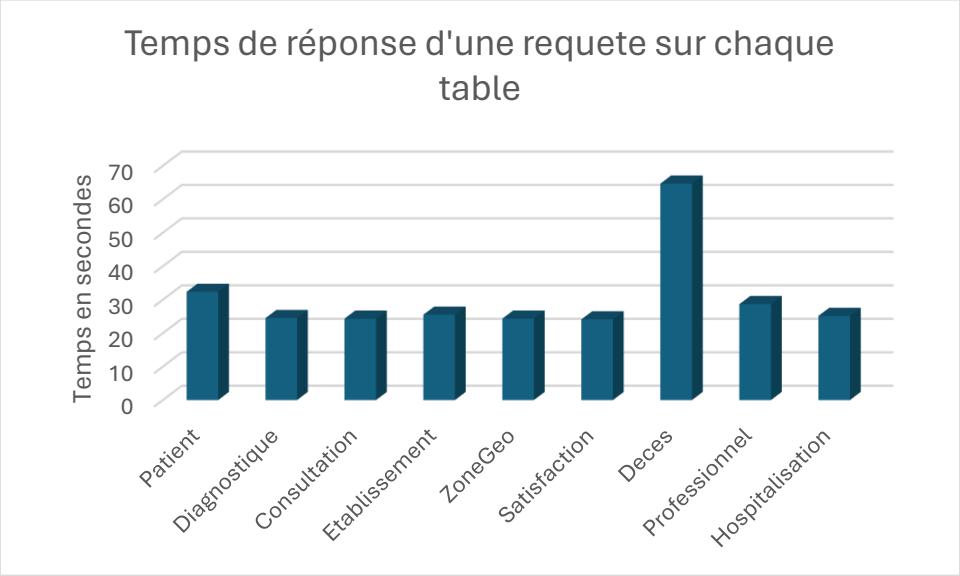
Pour la table décès, le partitionnement par année permettra de limiter le traitement aux données de cette année spécifique, tandis que le clustering par région aidera à accélérer l'agrégation des données par région.

b) Evaluation du temps de réponse

Temps de réponse pour des requêtes SQL (sans optimisation avec le partitionnement et l'utilisation de cluster) sous la forme de **SELECT * from « Table » WHERE « critère de la recherche ».** Cette requête a été effectué 3 fois pour éviter d'obtenir des données incohérentes.

Table	Temps de réponse 1 (en sec)	Temps de réponse 2 (en sec)	Temps de réponse 3 (en sec)	Moyenne (en secondes)	Nombre de résultats	critère de la recherche
Patient	48	25	24	32,333333	800	annee = 2000
Diagnostique	25	24	24.05	24,5	409	un code diagnostique = 'A%'
Consultation	23,81	25,1	24	24,303333	33896	annee = 2015
Etablissement	26,56	24,96	25	25,506667	410	departement = 54
ZoneGeo	25,77	24	23,2	24,323333	1	departement = 54
Satisfaction	24	23,3	25	24,1	27	departement = 54
Deces	68	62	64	64,666667	549000	annee = 2000
Professionnel	30	31	25	28,666667	18000	profession = osteopathe
Hospitalisation	25	26	24,4	25,133333	410	annee = 2000

Tableau de résultats des temps de réponse



5. Conclusion

Après l'exécution réussie des tâches, ayant ainsi alimenté nos tables avec les données pertinentes et constaté les bénéfices du partitionnement pour accélérer nos requêtes, il est désormais temps de donner vie à ces données en créant un tableau de bord sur PowerBI.