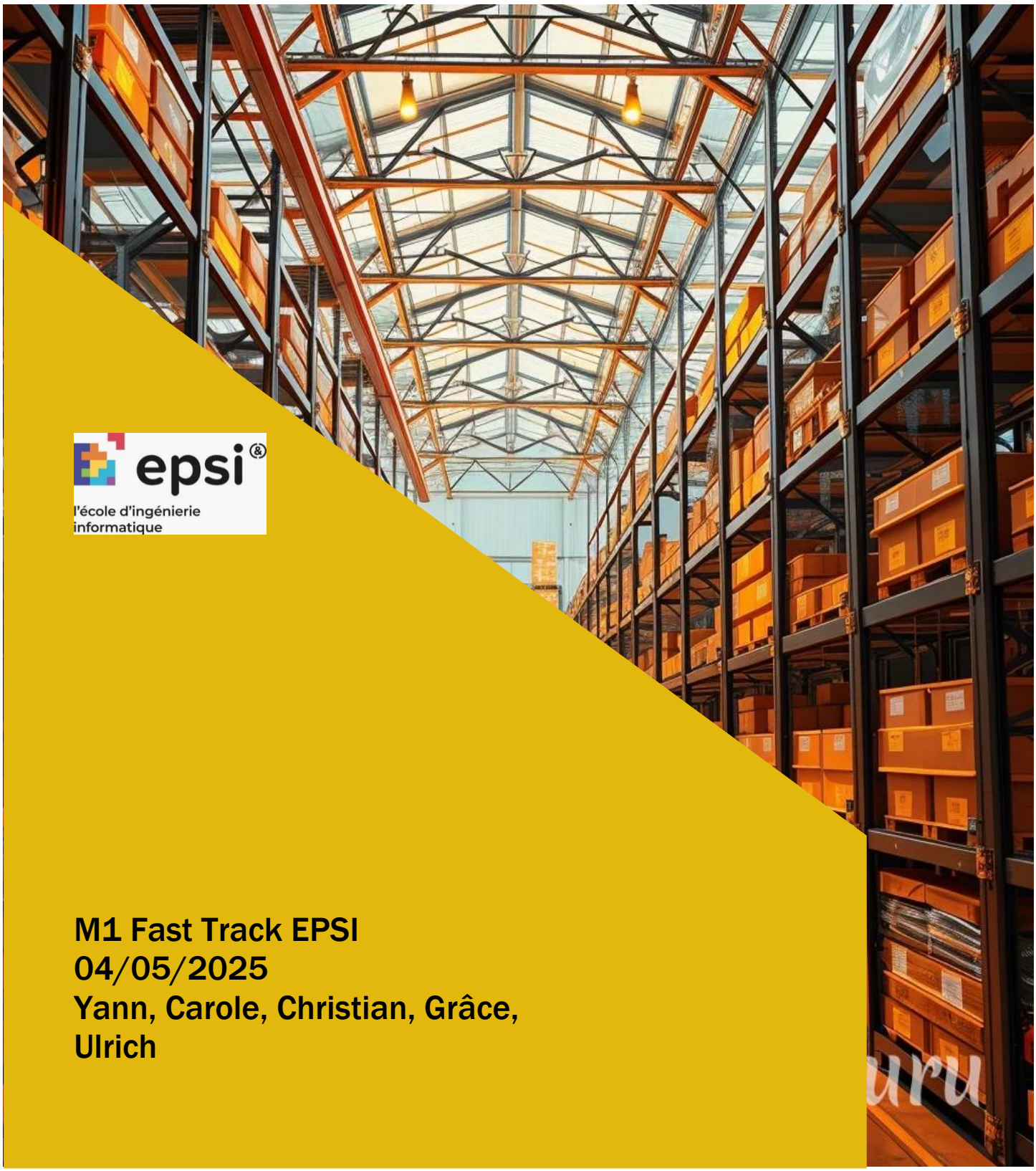
ATL-DATAMART



epsi
l'école d'ingénierie informatique

M1 Fast Track EPSI
04/05/2025
Yann, Carole, Christian, Grâce, Ulrich

Table des matières

Table des figures

Introduction

Dans un monde où la donnée est devenue le nouveau pétrole, la maîtrise des architectures décisionnelles s'impose comme une compétence clé pour tout futur data analyst ou data engineer. Ce projet, réalisé dans le cadre du cours d'Atelier Architecture Décisionnelle Datamart, nous a plongés au cœur d'une problématique moderne : construire une pipeline analytique complète pour exploiter les données des taxis new-yorkais.

Inspiré des standards du CAC40, ce travail a reproduit à échelle réduite l'écosystème complet d'une entreprise data-driven, depuis la collecte de données brutes jusqu'à leur valorisation via des tableaux de bord interactifs. À travers cinq travaux pratiques progressifs, nous avons implémenté les briques fondamentales d'une architecture Big Data : datalake (MinIO), data warehouse (PostgreSQL), datamart multidimensionnel et outils de visualisation (Power BI).

Ce parcours nous permettra d'expérimenter concrètement les défis techniques et méthodologiques rencontrés en entreprise : intégration de sources hétérogènes, modélisation dimensionnelle, automatisation des flux ETL, et création de dashboards métiers.

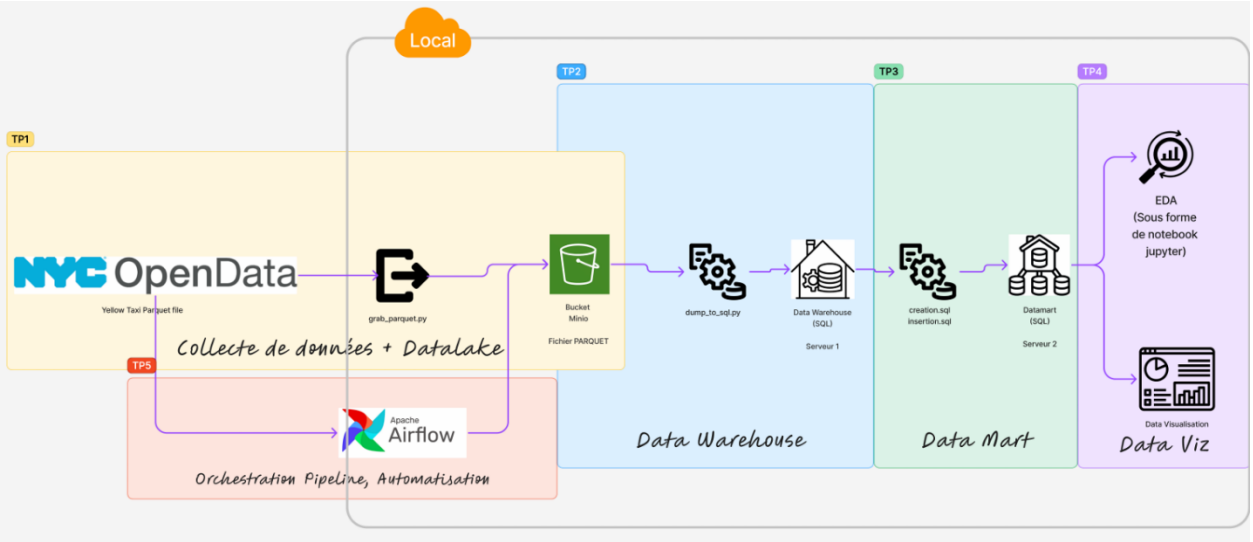


Figure 1: Schéma global de l'architecture

1. TP1 – La collecte des données vers un datalake

La première étape a porté sur la mise en place de l'infrastructure de collecte. L'environnement Docker a été configuré pour héberger deux services essentiels : un serveur MinIO pour le stockage objet et un serveur PostgreSQL destiné au data warehouse (nyc_warehouse, exposé sur le port 15432).

Un script Python (grab_parquet.py) a ensuite été développé pour automatiser le téléchargement des fichiers yellow_tripdata_2024.parquet correspondant aux mois d'octobre, novembre et décembre. Ces fichiers ont été stockés dans un bucket MinIO nommé nyc-taxi.

Difficultés rencontrées :

- La première difficulté a concerné la connexion entre le script Python et MinIO, notamment l'accès aux fichiers via l'API. Cette erreur était liée à une mauvaise configuration des identifiants d'accès (clé et secret).
- Une autre erreur récurrente était liée au format des fichiers stockés. Certains fichiers Parquet présentaient des incohérences de schéma lors du premier test de chargement.

Solutions apportées :

- L'ensemble des paramètres de connexion à MinIO a été revérifié et externalisé dans un fichier de configuration .env pour éviter les erreurs manuelles.
- Des vérifications préliminaires ont été ajoutées dans le script pour contrôler la structure de chaque fichier avant traitement.

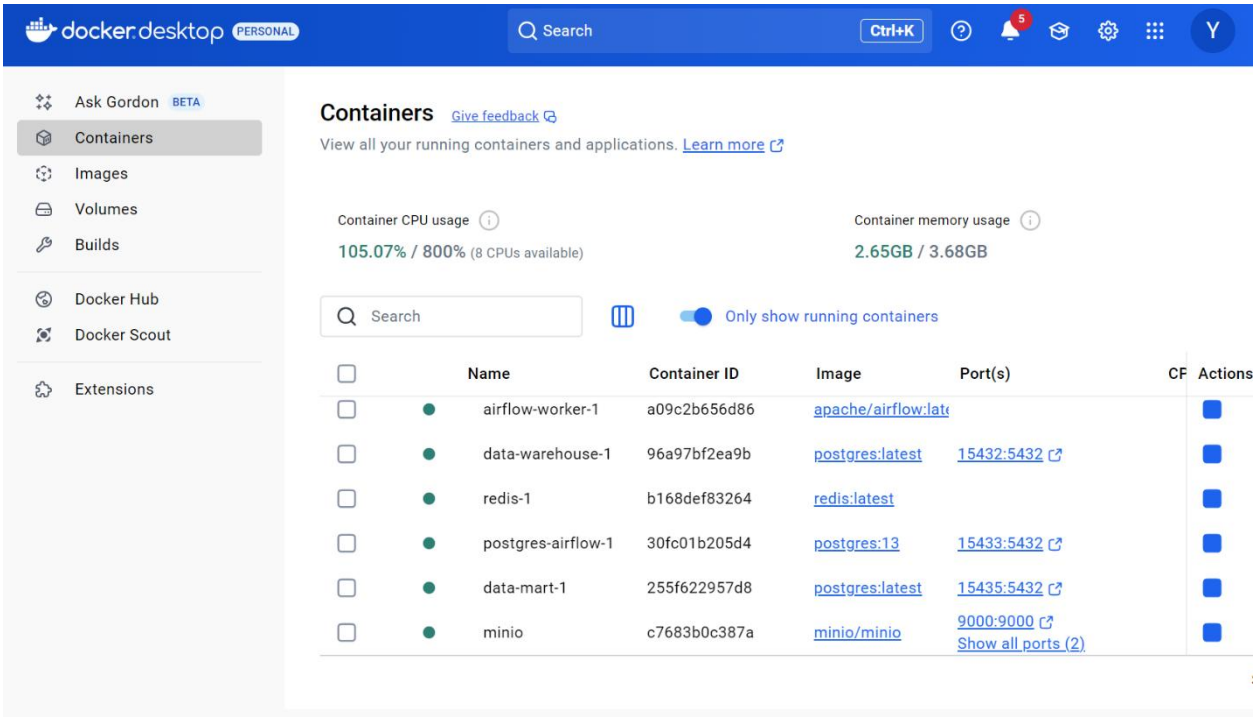


Figure 2: containers actifs

dump_to_sql.py | insertion.sql | creation.sql | grab_parquet.py

src > data > grab_parquet.py > ...

```python
from minio import Minio
import urllib.request
import pandas as pd
import sys
import os

def main():
    grab_data()
    write_data_minio()

def grab_data() -> None:
    """Grab the data from New York Yellow Taxi
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\33758\Desktop\YANN\ATL-Datamart> & C:/Users/33758/Desktop/YANN/ATL
 c:/Users/33758/Desktop/YANN/ATL-Datamart/src/data/grab_parquet.py
⏭ yellow_tripdata_2024-10.parquet déjà présent, téléchargement ignoré.
⏭ yellow_tripdata_2024-11.parquet déjà présent, téléchargement ignoré.
⏭ yellow_tripdata_2024-12.parquet déjà présent, téléchargement ignoré.
ℹ Bucket 'nyc-taxi' existe déjà.
📤 Upload de yellow_tripdata_2024-10.parquet vers le bucket nyc-taxi...
✅ yellow_tripdata_2024-10.parquet uploadé avec succès.
📤 Upload de yellow_tripdata_2024-11.parquet vers le bucket nyc-taxi...
✅ yellow_tripdata_2024-11.parquet uploadé avec succès.
📤 Upload de yellow_tripdata_2024-12.parquet vers le bucket nyc-taxi...
✅ yellow_tripdata_2024-12.parquet uploadé avec succès.
PS C:\Users\33758\Desktop\YANN\ATL-Datamart>
```

Figure 3: Logs du script Python grab_parquet.py montrant l'upload réussi

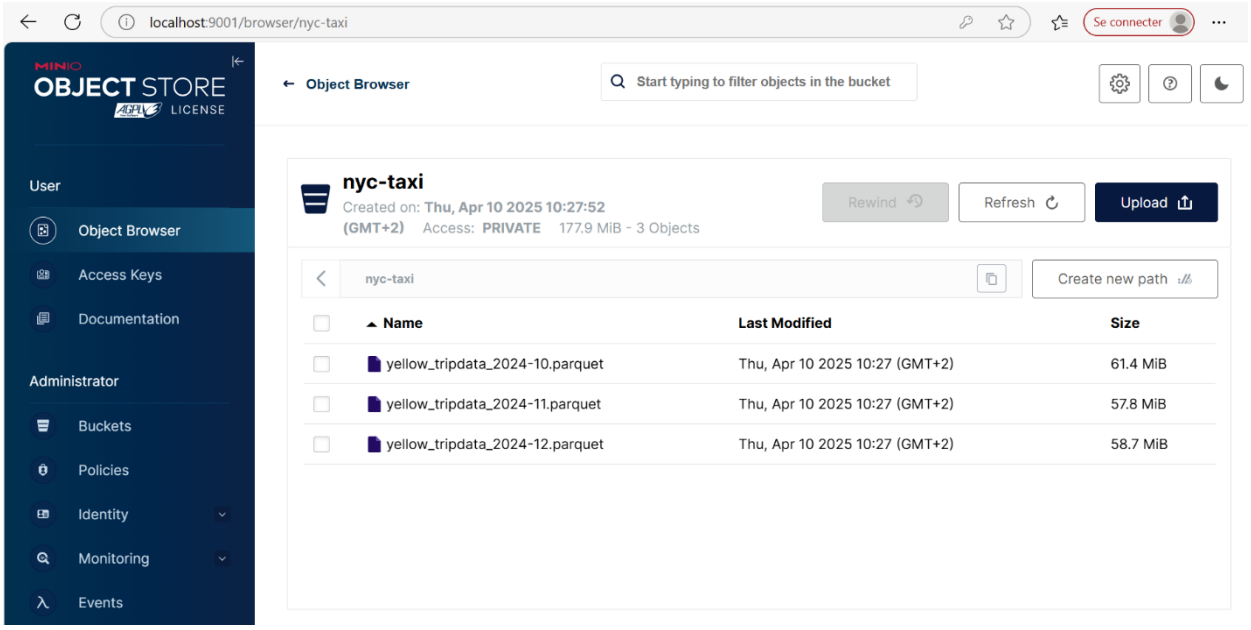


Figure 4: Bucket MinIO nyc-taxi avec les fichiers Parquet listés (preuve du stockage réussi)

2. TP2 – Du datalake vers un datawarehouse

La deuxième phase a consisté à transférer les données depuis MinIO vers PostgreSQL. Un script Python (dump_to_sql.py) a été mis en place pour lire les fichiers Parquet, les nettoyer si nécessaire, puis les insérer dans une table appelée nyc_raw au sein de la base nyc_warehouse.

Le script vérifie également l'existence de la base et la crée si besoin. Des logs et une gestion des erreurs ont été ajoutés pour assurer une exécution fiable.

Difficultés rencontrées :

- L'un des premiers blocages était lié à la non-existence de la base de données nyc_warehouse. Le script échouait si la base n'était pas déjà créée manuellement.
- Une autre difficulté technique provenait de conflits de types de données lors de l'insertion : certains champs attendus en INTEGER contenaient en réalité des valeurs flottantes ou nulles.
- Enfin, quelques erreurs d'encodage ou de noms de colonnes ont été rencontrées lors du mapping automatique du DataFrame vers PostgreSQL.

Solutions apportées :

- Le script a été modifié pour inclure un mécanisme de création automatique de la base si elle n'existe pas, avec fallback manuel via pgAdmin si besoin.
- Un nettoyage plus rigoureux des données a été introduit en amont de l'insertion, notamment via des fonctions fillna() et astype() dans Pandas.
- La définition manuelle du schéma a été préférée au mapping automatique afin d'avoir un meilleur contrôle sur les types et la structure des colonnes.

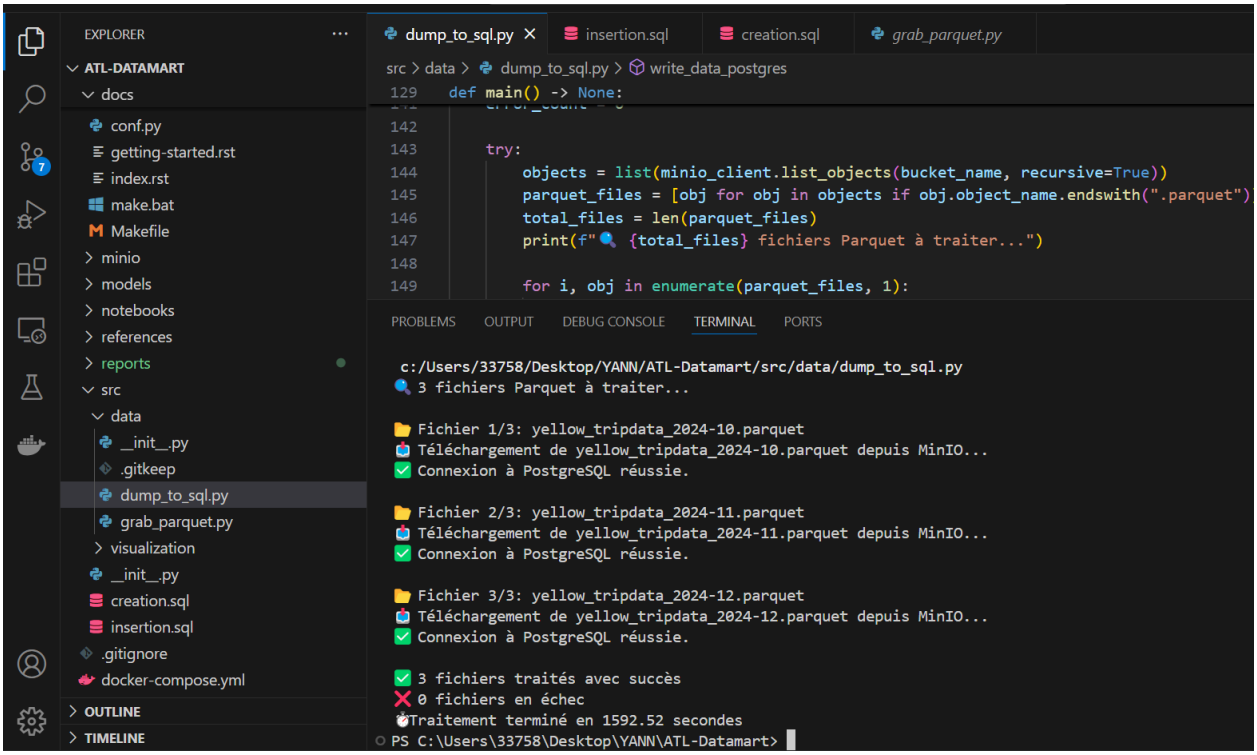


Figure 5: Script Python dump_to_sql.py avec logs de succès

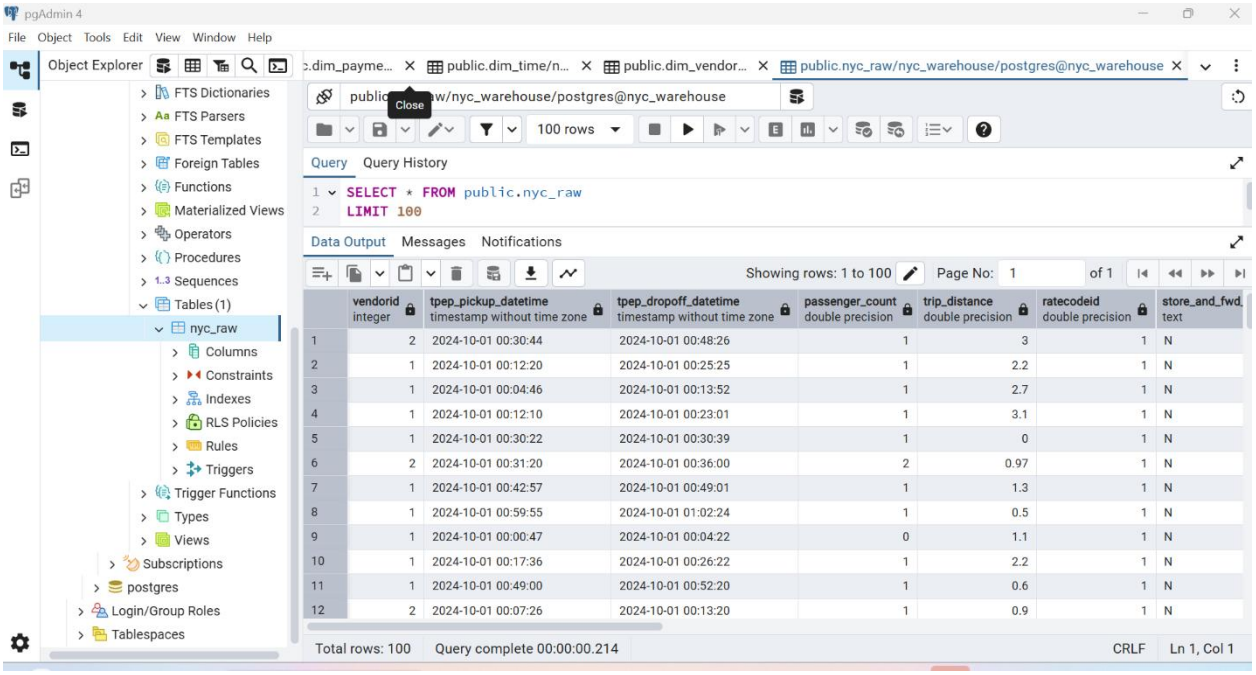


Figure 6: Extrait de la table nyc_raw du data warehouse

3. TP3 – Datamart et création d'un modèle multi-dimensionnelle

Dans cette troisième phase, un second serveur PostgreSQL a été déployé (sur le port 15435) pour accueillir le datamart nyc_datamart. Un modèle en étoile y a été conçu, avec une table de faits fact_trips et plusieurs dimensions : dim_time, dim_location, dim_vendor et dim_payment.

Les scripts SQL creation.sql et insertion.sql ont été utilisés respectivement pour créer les structures de tables (avec les contraintes d'intégrité) et pour transférer les données depuis nyc_warehouse en utilisant l'extension Foreign Data Wrapper (FDW). Ce transfert s'est accompagné de transformations comme le formatage des dates, le mapping d'identifiants, et le nettoyage des valeurs manquantes.

Difficultés rencontrées :

- La configuration initiale du FDW contenait des erreurs, ce qui empêchait l'accès à la table distante nyc_raw.
- Des valeurs invalides dans la dimension temporelle provoquaient des violations des contraintes CHECK.
- Les identifiants de localisation ne correspondaient pas toujours à des zones connues.
- Des erreurs de typage ont été rencontrées lors de la tentative de transformation d'identifiants en UUID.

Solutions apportées :

- La configuration du FDW a été revue : les imports, les options de connexion et les schémas ont été corrigés.
- Des fonctions de nettoyage spécifiques ont été ajoutées pour garantir la conformité des données temporelles (jours valides, formats normalisés).
- Un mapping additionnel a été créé manuellement pour enrichir les zones manquantes.
- Une nouvelle colonne UUID a été générée à partir de chaînes uniques, ce qui a permis une transition progressive sans casser les dépendances existantes.

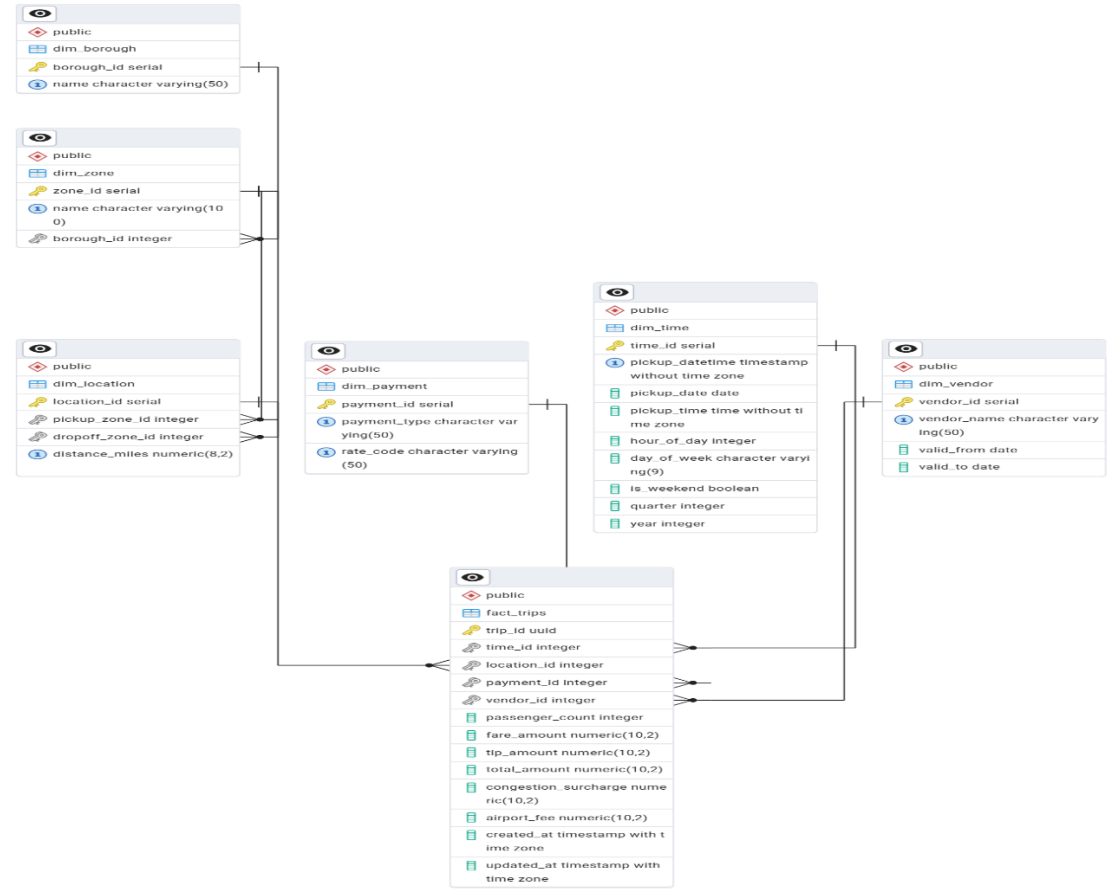


Figure 7: Diagramme du modèle montrant les tables du datamart avec leurs relations

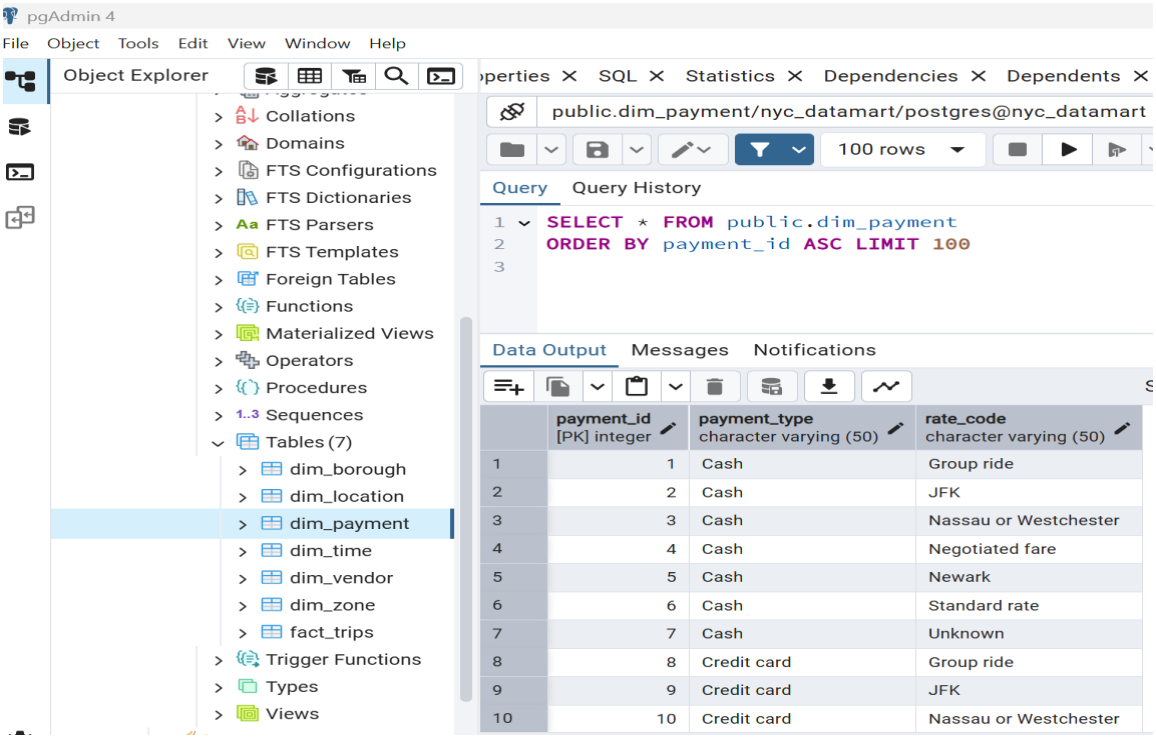


Figure 8: Extrait de la table dim_payment

4. TP4 – Visualisation des données

La dernière phase du projet a porté sur la connexion du datamart à Power BI afin de produire un tableau de bord interactif. Après avoir configuré Power BI pour accéder au serveur PostgreSQL sur le port 15435, les données ont pu être importées et visualisées.

Difficultés rencontrées :

- La connexion au datamart échouait en raison d'un nom de serveur mal saisi. Les formats localhost, 15435, ou encore nyc_datamart n'étaient pas valides.

Solution apportée :

- La syntaxe correcte a été identifiée comme étant localhost:15435, permettant ainsi la connexion de Power BI au serveur PostgreSQL.

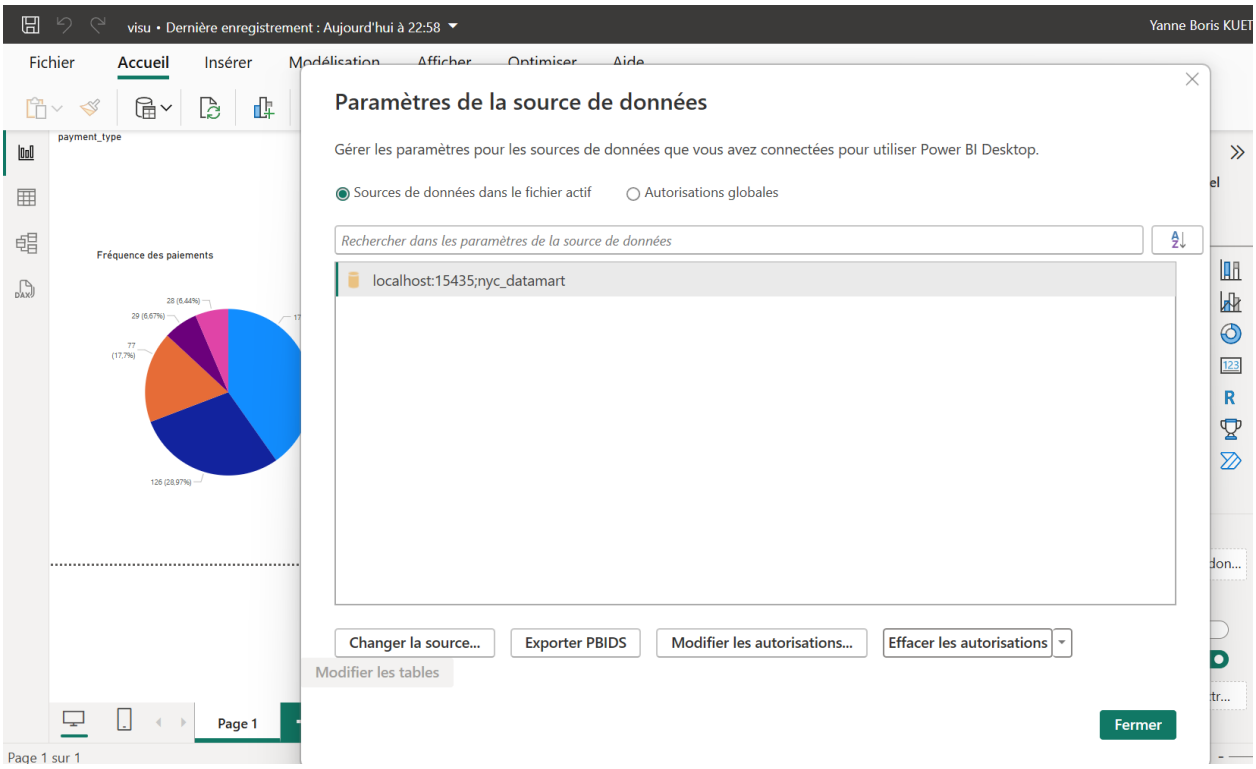


Figure 9: Connexion de Power BI au datamart PostgreSQL (source de données localhost:15435)

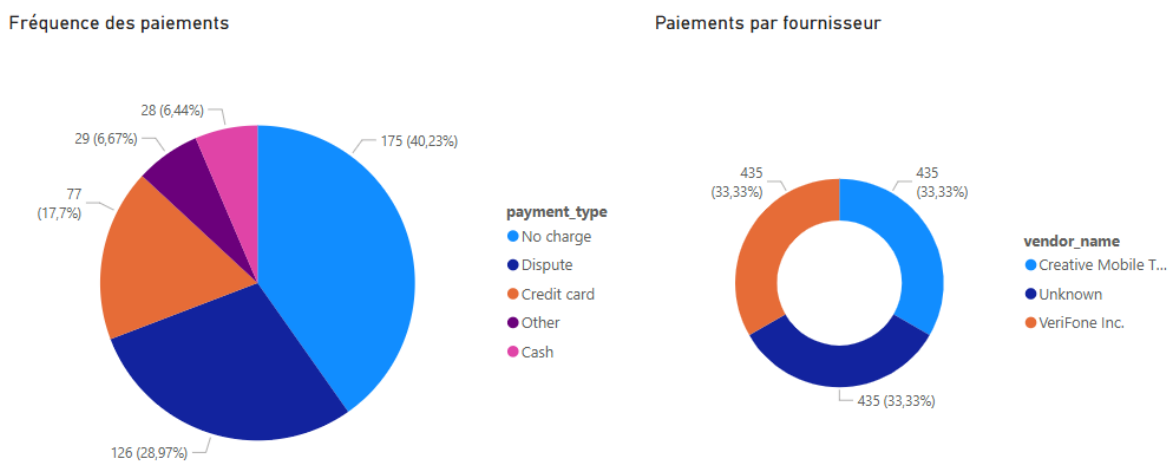


Figure 10: Dashboard final

5. Bilan et Résultats

Ce projet a permis de mettre en place une architecture analytique robuste avec une séparation claire entre le data warehouse opérationnel (nyc_warehouse) et le datamart analytique (nyc_datamart). Les scripts Python et SQL développés permettent de rejouer facilement le pipeline ETL, grâce à une automatisation complète et à une gestion rigoureuse des erreurs. La visualisation finale via Power BI valide la qualité et la cohérence des données transformées.

Conclusion

En somme, ce projet autour des données des taxis new-yorkais aura été une occasion unique de consolider nos compétences sur l'ensemble de la chaîne de valeur data. De la collecte initiale dans MinIO à la visualisation finale dans Power BI, chaque étape nous a confrontés à des défis techniques riches en enseignements. Il en ressort donc que, au-delà de la simple maîtrise d'outils (Docker, PostgreSQL, Python, Power BI), cette expérience nous a surtout appris l'importance cruciale d'une architecture bien pensée et documentée. Les difficultés rencontrées - intégration de données, gestion des schémas évolutifs, modélisation dimensionnelle - nous ont forgé une véritable rigueur dans la conception de pipelines data. Le résultat final, avec son datamart optimisé pour l'analyse OLAP et ses dashboards interactifs, valide l'efficacité de l'approche par étapes. Plus qu'une réussite technique, ce projet démontre notre capacité à transformer des données brutes en véritables insights décisionnels.

Ces compétences en architecture data, désormais éprouvées sur un cas concret, constituent un atout majeur pour notre future carrière dans l'écosystème data. Elles témoignent de notre aptitude à concevoir des solutions complètes, robustes et orientées valeur métier - la clé de voûte des projets data réussis.