# Rapport : Atelier Architecture Décisionnelle (DataMart)

**Participants : Axel Depoitre, Paul-Louis Mignotte, Sadish Senthilkumar, Ethan Bensaid**

Encadré par : Rakib SHEIKH
Durée : 14 heures
Cours dispensé pour : I1 EISI, I1 ECDPIA, I1 ESI Cyber
Date : [Insérez la date ici]

# Rapport : Atelier Architecture Décisionnelle (DataMart)

**Participants :** Axel Depoitre, Paul-Louis Mignotte, Sadish Senthilkumar, Ethan Bensaid

**Nom de l'atelier :** Architecture Décisionnelle (DataMart)
**Encadré par :** Rakib SHEIKH
**Durée :** 14 heures
**Cours dispensé pour :** I1 EISI, I1 ECDPIA, I1 ESI Cyber

Cet atelier avait pour objectif principal de mettre en pratique les concepts d'architecture décisionnelle en réalisant un projet complet, allant de la collecte des données à leur visualisation en passant par leur traitement et stockage. Ce document présente le déroulement des travaux pratiques ainsi que les résultats obtenus.

## 2. Objectifs du Projet

Le projet portait sur l'analyse de données issues du secteur des VTC à New York. Les objectifs principaux étaient :
- Automatiser la récupération des données publiques depuis le site de l'État de New York vers un Data Lake.
- Mettre en place des processus ETL pour transformer et nettoyer les données.
- Stocker les données dans un Data Warehouse puis un Data Mart.
- Créer un tableau de bord connecté pour visualiser les KPI (indicateurs de performance).
- Automatiser certaines tâches avec Apache Airflow.

## 3. Déroulement des Travaux Pratiques

Les différentes étapes des travaux pratiques, réparties en cinq TPs, sont détaillées ci-dessous.

### TP1 : Téléchargement des fichiers Parquet

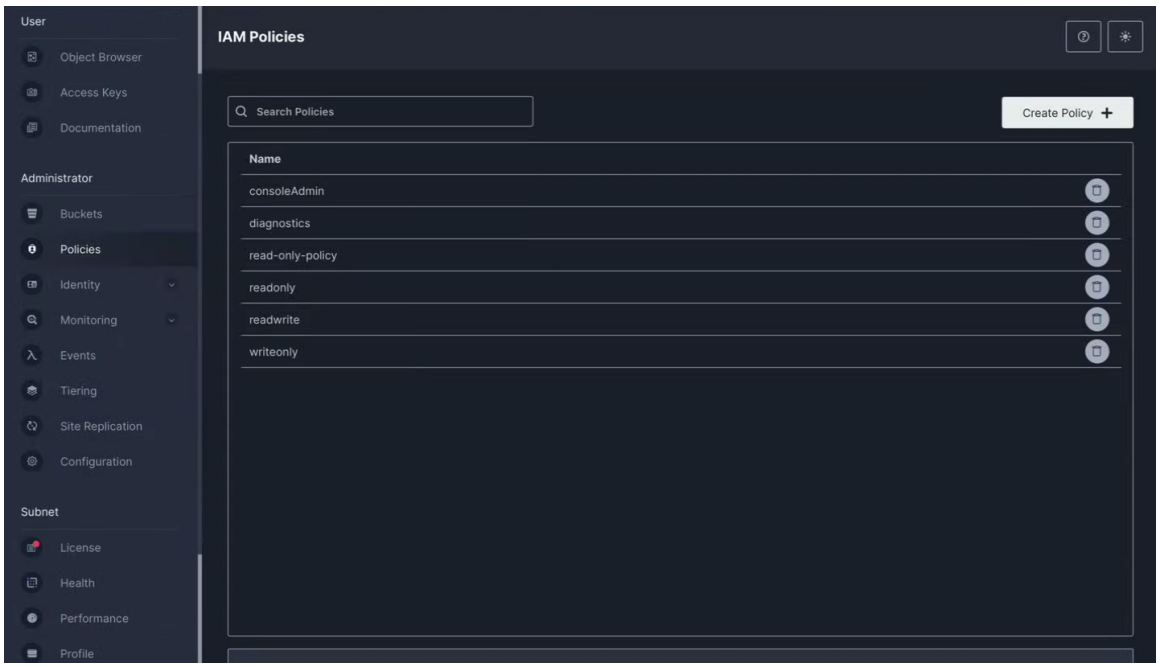
**Objectif :** Télécharger en local tous les fichiers parquet des taxis du site du gouvernement de New York en Python, puis uniquement le dernier fichier en date.

**Étapes réalisées :**

1. Utilisation de Docker pour démarrer les services :
    - Commande exécutée : docker compose up pour lancer l'ensemble des services définis dans le fichier docker-compose.yml.



2. Accès à l'interface web MinIO via l'URL http://127.0.0.1:9001/pour vérifier son bon fonctionnement.



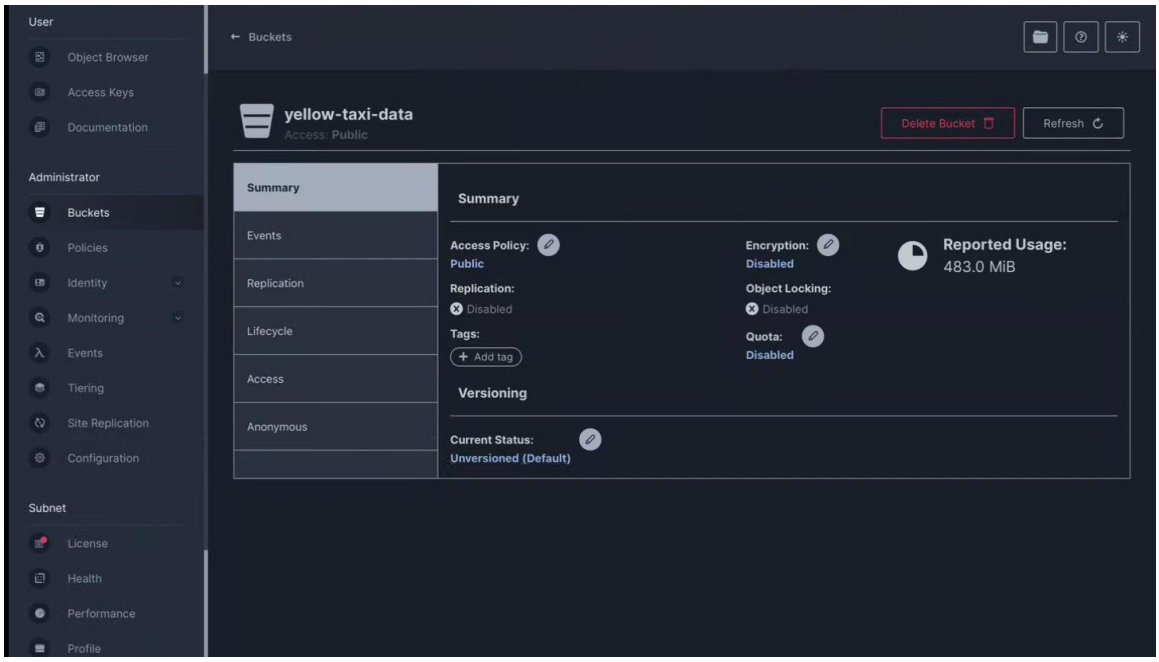

3. Écriture des scripts Python dans VSCode :
    - Commande utilisée : pip install -r requirements.txt.
4. Premier script : Téléchargement du fichier le plus récent (décembre 2024).

**Code : Scraping et Upload MinIO**

```
import ssl
import urllib.request
import requests
from bs4 import BeautifulSoup
from minio import Minio

# URL de la page à scraper
page_url = "https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page"

# Configuration MinIO
minio_client = Minio(
    "127.0.0.1:9000",
    secure=False,
    access_key="minio",
    secret_key="minio123"
)
bucket_name = "yellow-taxi-data"
if not minio_client.bucket_exists(bucket_name):
    minio_client.make_bucket(bucket_name)
    print(f"Bucket '{bucket_name}' créé.")
else:
    print(f"Bucket '{bucket_name}' déjà existant.")

# Fonction pour scraper tous les liens Parquet de Yellow Taxi Trip Records pour
2024
def get_yellow_taxi_links(url):
    response = requests.get(url)
    response.raise_for_status()  # Vérifie que la requête a réussi
    soup = BeautifulSoup(response.text, 'html.parser')

    # Trouver tous les liens pour les fichiers Yellow Taxi de 2024
    links = []
    for link in soup.find_all("a", href=True):
        href = link['href'].strip()
        if "yellow_tripdata_2024" in href and href.endswith(".parquet"):
            if href.startswith("http"):
                links.append(href)
            else:
                links.append(f"https://www.nyc.gov{href}")

    # Vérifier si tous les mois de janvier à décembre sont présents
    expected_months = {f"yellow_tripdata_2024-{str(month).zfill(2)}" for month
in range(1, 13)}
```

```
    available_months = {link.split('/')[-1].split('.')[0] for link in links}
    missing_months = expected_months - available_months

    if missing_months:
        print(f"Attention : les mois suivants sont manquants dans les liens extraits :
{missing_months}")
    else:
        print("Tous les mois de 2024 sont présents.")
    return links

# Fonction pour télécharger et uploader dans MinIO
def download_and_upload_to_minio(parquet_url):
    file_name = parquet_url.split("/")[-1]
    local_file_path = f"/tmp/{file_name}"

    context = ssl._create_unverified_context()

    try:
        print(f"Téléchargement de {file_name} depuis {parquet_url}...")
        with urllib.request.urlopen(parquet_url, context=context) as response,
open(local_file_path, 'wb') as out_file:
            out_file.write(response.read())
        print(f"{file_name} téléchargé avec succès.")
    except Exception as e:
        print(f"Erreur lors du téléchargement de {file_name} : {e}")
        return

    try:
        minio_client.fput_object(bucket_name, file_name, local_file_path)
        print(f"Fichier {file_name} uploadé avec succès dans le bucket
'{bucket_name}' sur MinIO.")
    except Exception as e:
        print(f"Erreur lors de l'upload de {file_name} dans MinIO : {e}")

if __name__ == "__main__":
    yellow_taxi_links = get_yellow_taxi_links(page_url)
    print(f"Liens Yellow Taxi récupérés : {yellow_taxi_links}")

    for parquet_link in yellow_taxi_links:
        download_and_upload_to_minio(parquet_link)
```

Une fois exécuté, le script crée bien le bucket sur MinIO :



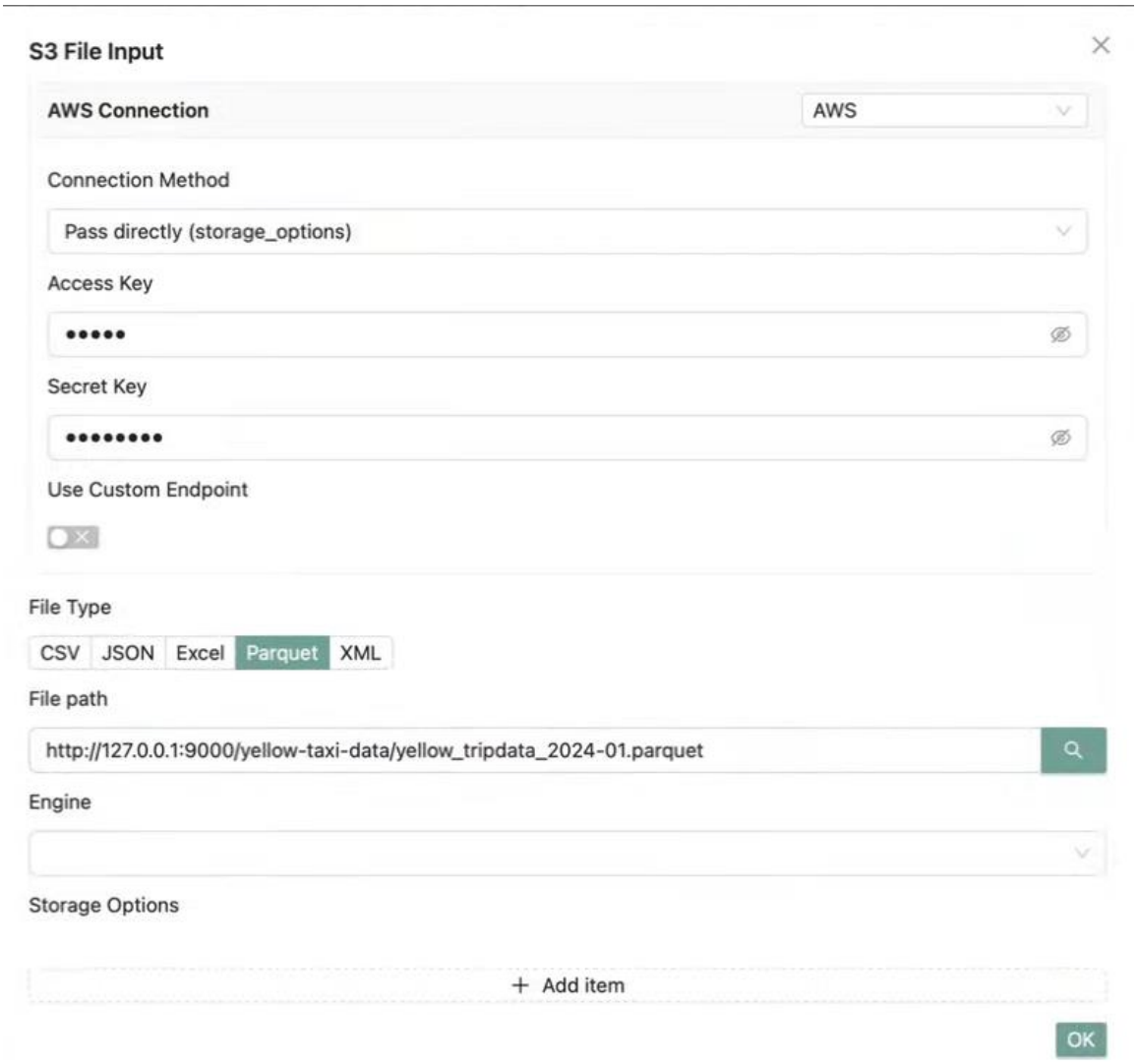

---

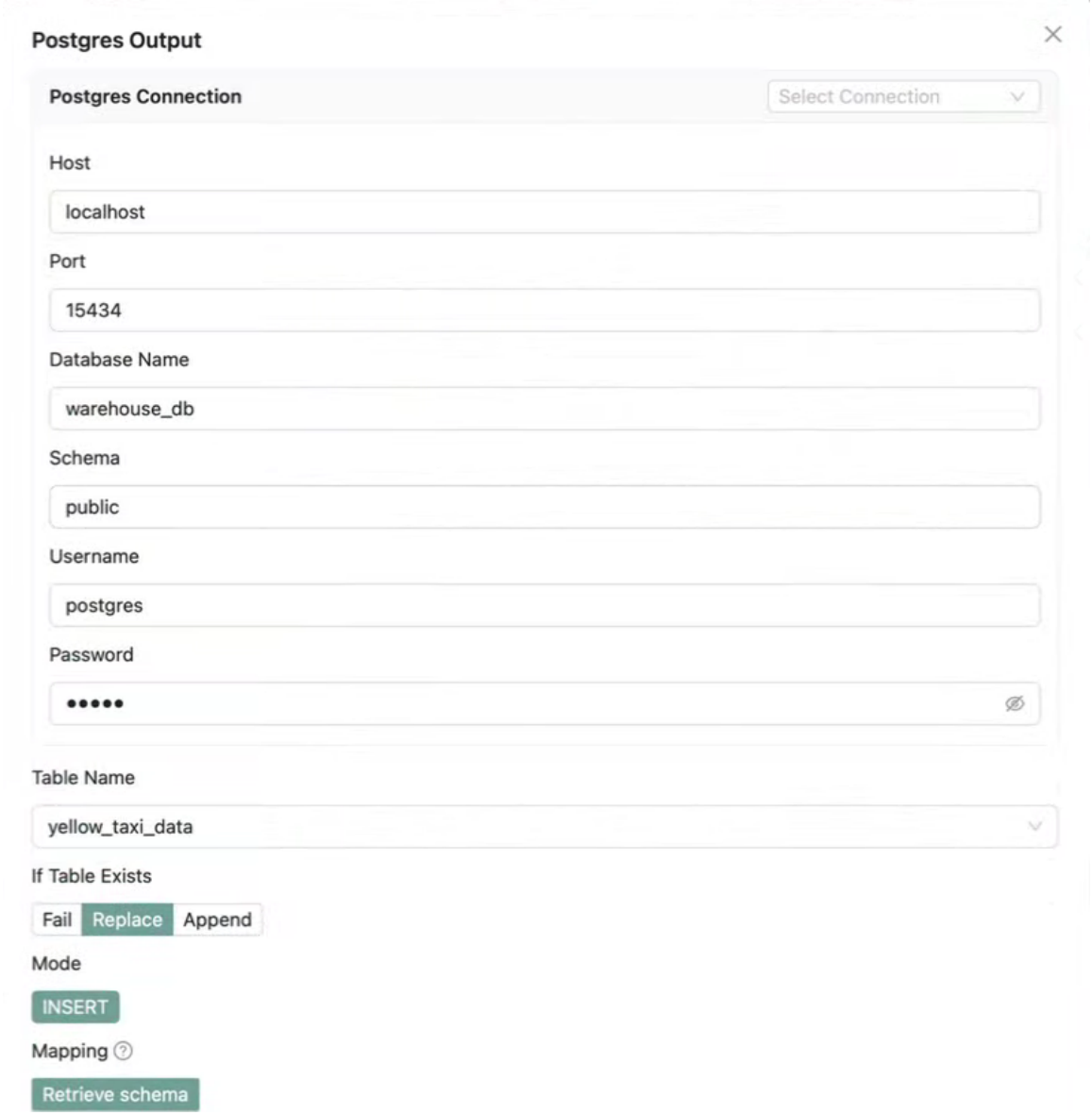
**TP2 : Envoi des fichiers vers MinIO et PostgreSQL**

**Objectif :** Envoyer les fichiers Parquet en local vers MinIO, puis transférer les données de MinIO vers PostgreSQL.

**MinIO vers PostgreSQL (ETL)**

1. Utilisation de l'outil ETL "Amphi AI" :
    - Installation des packages :
        - pip install jupyterlab-amphi
        - pip install --upgrade amphi
    - Suppression des exclusions "Minio" et "JupyterLab" dans le .gitignore.
2. Accès à la base PostgreSQL via DBeaver :
    - Modification des ports et des utilisateurs pour établir la connexion.
    - Capture d'écran : Test de connexion et vue de la base de données.
3. Mise en place du pipeline dans JupyterLab avec Amphi AI :
    - Ajout d'un "AWS S3 File Input" pour lire les fichiers Parquet depuis MinIO.
    - Ajout d'un "Postgres Output" pour charger les données dans PostgreSQL.


S3 File Input
AWS Connection
AWS
Connection Method
Pass directly (storage_options)
Access Key
•••••
Secret Key
••••••••
Use Custom Endpoint
File Type
CSV
JSON
Excel
Parquet
XML
File path
http://127.0.0.1:9000/yellow-taxi-data/yellow_tripdata_2024-01.parquet
Engine
Storage Options
+ Add item
OK




4. Création de la table suivante dans PostgreSQL :

```
CREATE TABLE yellow_taxis_data (
    VendorID INT,
    tpep_pickup_datetime TIMESTAMP,
    tpep_dropoff_datetime TIMESTAMP,
    passenger_count INT,
    trip_distance FLOAT,
    RatecodeID INT,
```

store_and_fwd_flag VARCHAR(10),

PULocationID INT,

DOLocationID INT,

payment_type INT,

fare_amount FLOAT,

extra FLOAT,

mta_tax FLOAT,

tip_amount FLOAT,

tolls_amount FLOAT,

improvement_surcharge FLOAT,

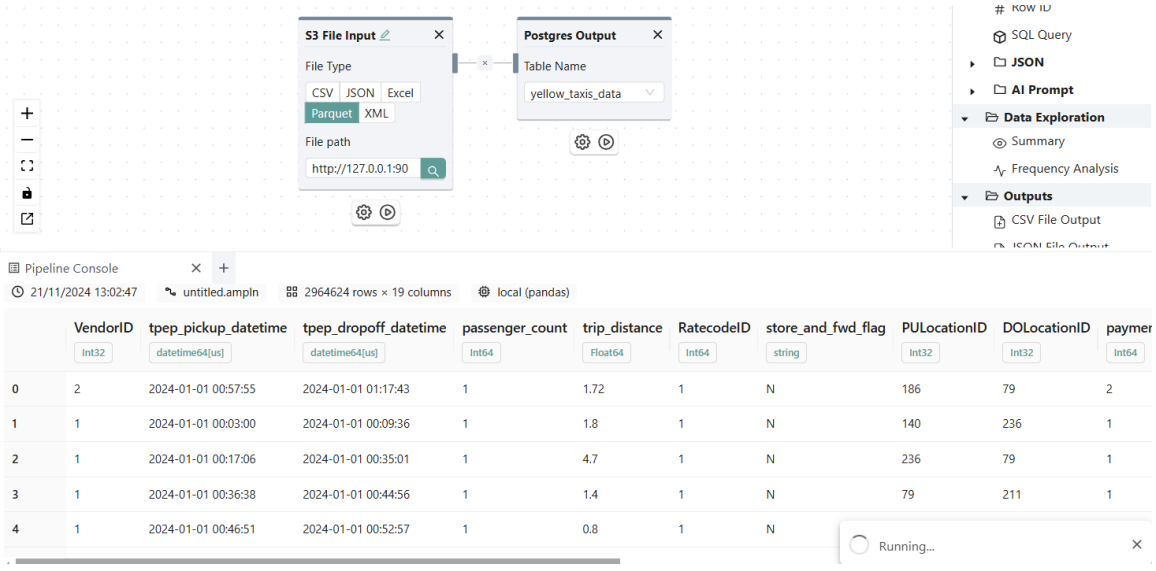
total_amount FLOAT,

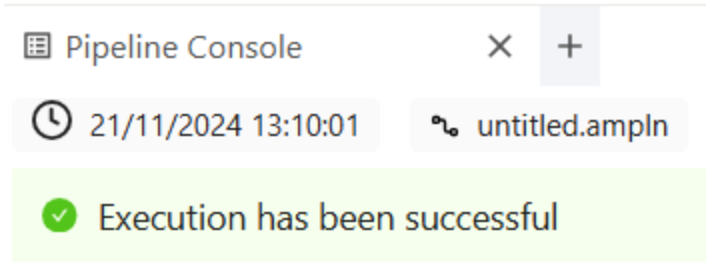
congestion_surcharge FLOAT,

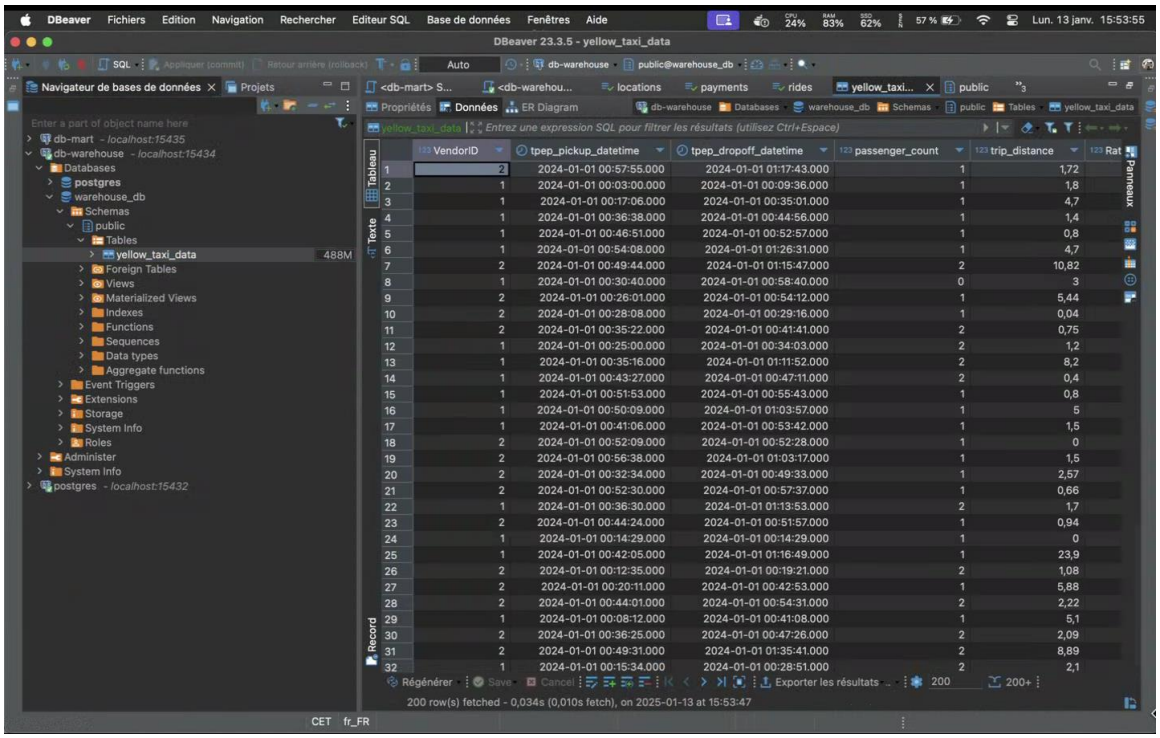
Airport_fee FLOAT

);

5. Exécution du pipeline dans JupyterLab pour charger les données dans PostgreSQL.





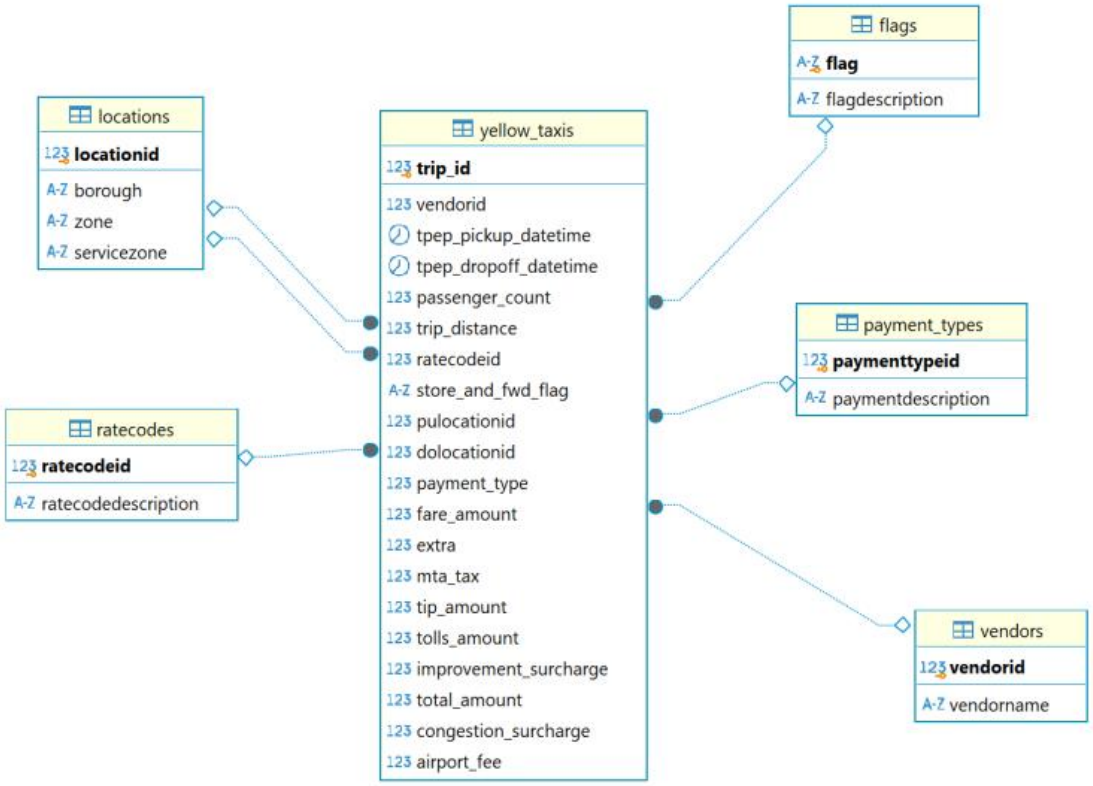

---

**TP3 : Modélisation et stockage des données retravaillées**

**Objectif :** Concevoir un modèle en flocon avec les données obtenues afin d'avoir des données retravaillées utilisables pour le tableau de bord, puis stocker de nouveau les résultats dans PostgreSQL OLAP.

**Modèle en flocon**

1. Conception d'un schéma en flocon basé sur les données disponibles :




 **Création de l'extension DBLink** :
Pour permettre la communication entre les bases de données OLTP (Warehouse) et OLAP (Datamart), l'extension DBLink a été configurée dans la base OLAP. Cela permet d'effectuer des requêtes distantes entre les deux instances PostgreSQL.

sql

CopierModifier

CREATE EXTENSION IF NOT EXISTS dblink;

 **Configuration de la connexion distante** :
La connexion entre les bases a été établie en précisant les paramètres d'accès (hôte, port, nom de base, utilisateur et mot de passe).

sql

CopierModifier

SELECT dblink_connect(

  'warehouse_conn',

'host=localhost port=15434 dbname=warehouse_db user=admin password=admin'

);

 **Validation de la connexion** :
Une simple requête de test a été effectuée pour valider la connexion et récupérer les données depuis la base OLTP (Warehouse).

sql

CopierModifier

```
SELECT *

FROM dblink('warehouse_conn', 'SELECT * FROM yellow_taxi_data LIMIT 1')

AS source_data(

  VendorID INT,

  tpep_pickup_datetime TIMESTAMP,

  tpep_dropoff_datetime TIMESTAMP,

  passenger_count INT,

  trip_distance FLOAT,

  RatecodeID INT,

  store_and_fwd_flag VARCHAR(10),

  PULocationID INT,

  DOLocationID INT,

  payment_type INT,

  fare_amount FLOAT,

  extra FLOAT,

  mta_tax FLOAT,

  tip_amount FLOAT,

  tolls_amount FLOAT,

  improvement_surcharge FLOAT,
```

```
    total_amount FLOAT,
    congestion_surcharge FLOAT,
    Airport_fee FLOAT
);
```

 **Transfert des données** :
Les données de la table *yellow_taxi_data* de la base Warehouse ont été transférées vers la table équivalente de la base OLAP.

```
sql
CopierModifier
INSERT INTO yellow_taxi_data
SELECT *
FROM dblink(
    'warehouse_conn',
    'SELECT * FROM yellow_taxi_data'
) AS source_data(
    VendorID INT,
    tpep_pickup_datetime TIMESTAMP,
    tpep_dropoff_datetime TIMESTAMP,
    passenger_count INT,
    trip_distance FLOAT,
    RatecodeID INT,
    store_and_fwd_flag VARCHAR(10),
    PULocationID INT,
    DOLocationID INT,
    payment_type INT,
    fare_amount FLOAT,
    extra FLOAT,
```

mta_tax FLOAT,

tip_amount FLOAT,

tolls_amount FLOAT,

improvement_surcharge FLOAT,

total_amount FLOAT,

congestion_surcharge FLOAT,

Airport_fee FLOAT

);

 **Résultats** :
Grâce à DBLink, la synchronisation des données entre les deux bases a été assurée, permettant ainsi un stockage optimisé dans le Datamart pour des analyses plus performantes.

---

**TP4 : Visualisation des données**

**Objectif :** Créer un tableau de bord dynamique et interactif pour visualiser les KPI.

**Étapes réalisées :**

1. Connexion à la DBMS Datamart avec JupyterLAB
2. Réalisation d'une analyse exploratoire des données (EDA) avec Jupyter Notebook pour identifier les KPI clés.
3. Conception des visualisations avec l'outil BI choisi.

```python
# Importer les bibliothèques nécessaires
from sqlalchemy import create_engine
import pandas as pd

# Définir les paramètres de connexion
db_user = "postgres"        # Remplacez par votre utilisateur
db_password = "admin"       # Mot de passe de la base
db_host = "localhost"       # Hôte (localhost ou IP de Docker)
db_port = "15435"           # Port exposé par le container
db_name = "mart_db"         # Nom de la base de données

# Connexion à la base de données
try:
    engine = create_engine(f"postgresql+psycopg2://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}")
    print("Connexion réussie à la base de données")
except Exception as e:
    print(f"Erreur de connexion : {e}")
```

```
Connexion réussie à la base de données
```

```python
# Lister les tables disponibles
query = """
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public';
"""
tables = pd.read_sql(query, engine)

# Afficher les tables disponibles
print(tables)
```

```
  table_name
0  locations
1   payments
2      rides
```

[3]:
```python
# Charger les premières lignes de la table locations
query = "SELECT * FROM locations LIMIT 10"
locations_data = pd.read_sql(query, engine)

# Afficher les données
print(locations_data)
```

```
   location_id location_name location_type
0            1            75        Pickup
1            2           247        Pickup
2            3            13        Pickup
3            4           144        Pickup
4            5           121        Pickup
5            6            93        Pickup
6            7            94        Pickup
7            8            34        Pickup
8            9           182        Pickup
9           10           115        Pickup
```

[6]:
```
# Statistiques descriptives pour rides
print(rides_data.describe())

# Vérifier les valeurs manquantes
print(rides_data.isnull().sum())
```

```
            ride_id           tpep_pickup_datetime  \
count  1.000000e+01                             10
mean   2.965184e+06     2024-01-01 06:42:02.700000
min    2.964625e+06              2024-01-01 00:34:51
25%    2.964930e+06  2024-01-01 01:21:44.750000128
50%    2.965214e+06              2024-01-01 04:53:43
75%    2.965451e+06  2024-01-01 11:45:27.249999872
max    2.965595e+06              2024-01-01 15:51:04
std    3.355533e+02                              NaN

            tpep_dropoff_datetime  trip_distance  passenger_count  \
count                          10      10.000000        10.000000
mean   2024-01-01 06:53:51.300000       2.017000         1.100000
min            2024-01-01 01:05:04       0.600000         1.000000
25%            2024-01-01 01:32:36       1.015000         1.000000
50%            2024-01-01 05:02:30       1.920000         1.000000
75%            2024-01-01 11:52:53       2.440000         1.000000
max            2024-01-01 15:56:25       5.240000         2.000000
std                            NaN       1.342875         0.316228

       total_amount  pickup_location_id  dropoff_location_id  payment_type
count     10.000000           10.000000            10.000000          10.0
mean      20.547000          185.000000           163.400000           1.0
min       12.100000           65.000000            49.000000           1.0
25%       14.700000          148.250000           119.000000           1.0
50%       17.850000          206.000000           143.000000           1.0
75%       21.562500          230.500000           233.500000           1.0
max       49.080000          249.000000           237.000000           1.0
std       10.868556           61.777378            67.348513           0.0
ride_id                   0
vendorid                 10
tpep_pickup_datetime      0
tpep_dropoff_datetime     0
```

[7]:
```
query = """
SELECT location_type, COUNT(*) AS total_locations
FROM locations
GROUP BY location_type
"""
location_summary = pd.read_sql(query, engine)
print(location_summary)
```

```
  location_type  total_locations
0       Dropoff              261
1        Pickup              520
```

[8]:
```
query = """
SELECT payment_type, COUNT(*) AS total_payments
FROM payments
GROUP BY payment_type
"""
payment_summary = pd.read_sql(query, engine)
print(payment_summary)
```

```
   payment_type  total_payments
0             0            3393
1             1           20876
2             3             190
3             4             484
4             2             672
```

[13]:
```
query = """
SELECT
    DATE_TRUNC('month', tpep_pickup_datetime) AS month,
    AVG(trip_distance) AS avg_distance,
    AVG(total_amount) AS avg_revenue_per_ride
FROM rides
WHERE
    tpep_pickup_datetime BETWEEN '2024-01-01' AND '2024-12-31' -- Filtrer uniquement sur l'année 2024
    AND total_amount > 0                                       -- Exclure les trajets avec un montant de 0 ou négatif
    AND trip_distance > 0                                      -- Exclure les trajets sans distance
GROUP BY month
ORDER BY month;
"""
filtered_monthly_analysis = pd.read_sql(query, engine)
print(filtered_monthly_analysis)
```

```
       month  avg_distance  avg_revenue_per_ride
0 2024-01-01      3.732297             27.329516
1 2024-02-01      4.083333             30.156667
```

**TP5 : Automatisation des tâches**

**Objectif :** Automatiser certaines tâches avec Apache Airflow pour optimiser les processus de gestion des données.

**Étapes réalisées :**

1. Création d'une DAG (Directed Acyclic Graph) pour la récupération mensuelle des données.

```
from urllib.request import urlopen, Request

from airflow.utils.dates import days_ago

from airflow import DAG

from airflow.operators.python import PythonOperator

import pendulum

import os

import tempfile

import ssl

import urllib.error


def download_parquet(**kwargs):

    """Download the most recent available Parquet file from a URL."""

    base_url = "https://d37ci6vzurychx.cloudfront.net/trip-data/"

    filename = "yellow_tripdata"

    extension = ".parquet"

    local_dir = tempfile.gettempdir()  # Use a temporary directory


    # Disable SSL verification (TEMPORARY, fix for production)

    ssl_context = ssl._create_unverified_context()


    # Search for the available file by checking past months
```

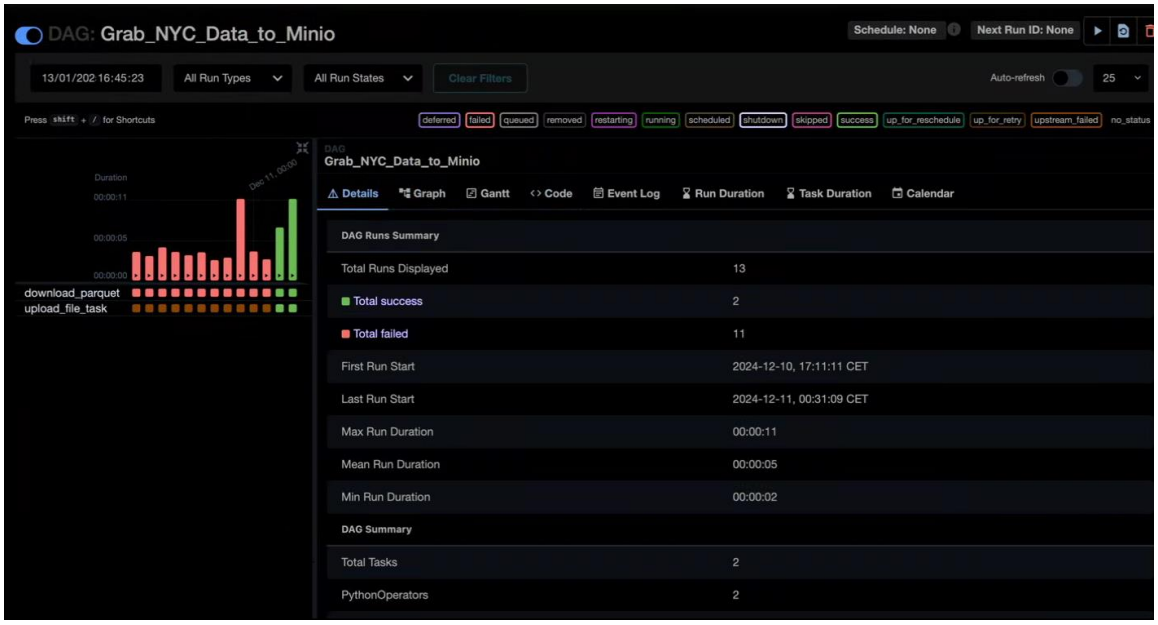
```
    for offset in range(6):  # Test up to 6 months back
        month = pendulum.now().subtract(months=offset).format('YYYY-MM')
        full_url = f"{base_url}{filename}_{month}{extension}"
        local_file_path = os.path.join(local_dir, f"{filename}_{month}{extension}")

        try:
            print(f"Testing URL: {full_url}")
            req = Request(
                full_url,
                headers={
                    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0'
                }
            )

            # Download the file if available
            with urlopen(req, context=ssl_context) as response, open(local_file_path, 'wb') as out_file:
                out_file.write(response.read())

            # Pass the local file path to the next task
            kwargs['ti'].xcom_push(key='file_path', value=local_file_path)
            return  # Stop the loop once a file is found
        except urllib.error.HTTPError as e:
            print(f"Failed to access the URL for {month}: {e}")
        except Exception as e:
            print(f"Unexpected error for {month}: {e}")
```

```
    # If no file is found, raise an error
    raise RuntimeError("No available Parquet file found in the last 6 months")

def upload_file(**kwargs):
    """Upload the downloaded file to MinIO."""
    # Retrieve the file path from XCom
    file_path = kwargs['ti'].xcom_pull(key='file_path', task_ids='download_parquet')
    if not file_path or not os.path.exists(file_path):
        raise RuntimeError(f"File not found at {file_path}")

    from minio import Minio, S3Error

    # MinIO client configuration
    client = Minio(
        "minio:9000",
        secure=False,
        access_key="minio",
        secret_key="minio123"
    )
    bucket = 'yellow-taxi-data'

    # Extract file name from the path
    object_name = os.path.basename(file_path)

    try:
```

```
        # Check if the bucket exists; if not, create it
        if not client.bucket_exists(bucket):
            client.make_bucket(bucket)

        # Upload the file
        client.fput_object(
            bucket_name=bucket,
            object_name=object_name,
            file_path=file_path
        )
        print(f"File uploaded successfully: {object_name}")
    except S3Error as e:
        raise RuntimeError(f"Failed to upload the file to MinIO: {str(e)}") from e
    finally:
        # Remove the local file after upload
        os.remove(file_path)

# Define the DAG
with DAG(
        dag_id='Grab_NYC_Data_to_Minio',
        start_date=days_ago(1),
        schedule_interval=None,
        catchup=False,
        tags=['minio', 'read', 'write']
) as dag:
    # Task to download the file
```

```
t1 = PythonOperator(
    task_id='download_parquet',
    python_callable=download_parquet
)

# Task to upload the file
t2 = PythonOperator(
    task_id='upload_file_task',
    python_callable=upload_file
)

# Task dependencies
t1 >> t2
```

2. Test et validation de la DAG dans Apache Airflow.




---

## 4. Conclusion

Cet atelier a permis d'acquérir des compétences pratiques dans la mise en place d'une architecture décisionnelle. Les différentes étapes, de la collecte des données à leur visualisation, ont été réalisées avec succès. Les outils explorés (Python, MinIO, PostgreSQL, Tableau, et Apache Airflow) ont joué un rôle essentiel dans l'atteinte des objectifs.

Ce projet constitue une base solide pour de futures applications dans le domaine de l'analyse de données et de la business intelligence.

---