



Atelier Architecture décisionnel Datamart

Encadré par : SHEIKH Rakib

Réalisé par :

- **BCHIRI Imane**
- **BOUAYAD Zineb**
- **BARHAMI Mohamed**
- **BENKIRANE AbdelKarim**

Table des matières

1.	Prérequis et installation	4
1.1.	Installation nécessaire	4
1.1.1.	Installation de Git	4
1.1.2.	Installation de Docker Desktop	4
1.1.3.	Installation de Python	4
1.2.	Cloner le projet	4
1.2.1.	Récupérer le dépôt :	4
1.3.	Lancement de l'infrastructure :	5
2.	TP1 : Téléchargement et Chargement de Données sur MinIO	6
2.1.	Objectif	6
2.2.	Étapes Réalisées	6
2.2.1.	Création d'un Script Python pour l'Extraction des Donnée	6
2.2.2.	Organisation des Données Localement	7
2.2.3.	Chargement des Données sur MinIO	7
2.3.	Architecture ETL Mise en Place	8
2.4.	Exécution	8
2.4.1.	Accéder à MinIO (interface web) :	8
2.4.2.	Exécutons le script <i>grab_parquet.py</i>	9
3.	TP2 : Transfert des données depuis le Data Lake (MinIO) vers le Data Warehouse (PostgreSQL)	9
3.1.	Objectif	9
3.2.	Contexte technique	9
3.3.	Architecture et étapes du TP	10
3.3.1.	Configuration de MinIO :	10
3.3.2.	Lecture des fichiers Parquet depuis MinIO :	10
3.3.3.	Création de l'engine PostgreSQL :	10
3.3.4.	Injection des données dans PostgreSQL :	11
3.4.	Exécution	11
4.	TP3 : Conception d'un Data Mart avec PostgreSQL : Implémentation d'un Modèle Flocon	13
4.1.	Objectif	13
4.2.	Détails des étapes réalisées	13
4.2.1.	Création des tables (creation.sql)	13

4.2.2.	Insertion des données (insertion.sql)	14
4.3.	Tests et Validation	15
5.	TP4 : Visualisation	22
6.	TP5 : Automatisation du TP1 avec Airflow	24
6.1.	Téléchargement des données	24
6.2.	Chargement vers MinIO	25
6.3.	Test et Validation	25

1. Prérequis et installation

1.1. Installation nécessaire

1.1.1. Installation de Git

```
C:\Users\hp>git --version  
git version 2.47.1.windows.2
```

1.1.2. Installation de Docker Desktop

```
C:\Users\hp>docker --version  
Docker version 27.4.0, build bde2b89
```

```
C:\Users\hp>docker compose version  
Docker Compose version v2.31.0-desktop.2
```

1.1.3. Installation de Python

```
C:\Users\hp>python --version  
Python 3.12.0
```

```
C:\Users\hp>pip --version  
pip 24.3.1 from C:\Users\hp\AppData\Local\Programs\Python\Python312\Lib\site-packages\pip (python 3.12)
```

1.2. Cloner le projet

1.2.1. Récupérer le dépôt :

```
PS C:\Users\DELL\Downloads> git clone https://github.com/Noobzik/ATL-Datamart/  
Cloning into 'ATL-Datamart'...  
remote: Enumerating objects: 91, done.  
remote: Counting objects: 100% (44/44), done.  
remote: Compressing objects: 100% (27/27), done.  
remote: Total 91 (delta 32), reused 17 (delta 17), pack-reused 47 (from 3)  
Receiving objects: 100% (91/91), 475.85 KiB | 1.25 MiB/s, done.  
Resolving deltas: 100% (36/36), done.
```


Vérifions que tous les services sont bien lancés avec : docker ps

```
C:\Users\DELL\Downloads\ATL-Datamart-main\ATL-Datamart-main>docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
61955dd4299b	apache/airflow:latest-python3.11	"/usr/bin/dumb-init -"	14 minutes ago	Up 14 minutes (healthy)	8080/tcp	atl-datamart-main-airflow-triggerer-1
6c55c17e343	apache/airflow:latest-python3.11	"/usr/bin/dumb-init -"	14 minutes ago	Up 14 minutes (healthy)	8080/tcp	atl-datamart-main-airflow-worker-1
e0494916982b	apache/airflow:latest-python3.11	"/usr/bin/dumb-init -"	14 minutes ago	Up 14 minutes (healthy)	8080/tcp	atl-datamart-main-airflow-scheduler-1
f9bec7002c01	apache/airflow:latest-python3.11	"/usr/bin/dumb-init -"	14 minutes ago	Up 14 minutes (healthy)	0.0.0.0:8080->8080/tcp	atl-datamart-main-airflow-webserver-1
309569cd35e5	postgres:latest	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	0.0.0.0:15433->5432/tcp	atl-datamart-main-data-mart-1
e2a88eb20d7a	postgres:13	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes (healthy)	0.0.0.0:15433->5432/tcp	atl-datamart-main-postgres-airflow-1
3d80a6f81d0d	postgres:latest	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	0.0.0.0:15432->5432/tcp	atl-datamart-main-data-warehouse-1
5981e5c2a3ad	redis:latest	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes (healthy)	6379/tcp	atl-datamart-main-redis-1
82ee28270d7d	minio/minio	"/usr/bin/docker-ent..."	14 minutes ago	Up 14 minutes	0.0.0.0:9000-9001->9000-9001/tcp	minio

2. TP1 : Téléchargement et Chargement de Données sur MinIO

2.1. Objectif

Ce TP avait pour objectif de mettre en place un processus d'extraction et de chargement des données (ETL) depuis une source externe jusqu'à un stockage local, suivi de leur transfert sur un serveur MinIO. Il constitue la première étape pour la construction d'une architecture décisionnelle complète.

2.2. Étapes Réalisées

2.2.1. Création d'un Script Python pour l'Extraction des Données

La fonction 'grab_data_2023_to_2024' a été développée pour télécharger les fichiers de données au format .parquet depuis une URL donnée.

```
# Fonction pour télécharger les données pour la période 2023 à 2024
def grab_data_2023_to_2024() -> None:
    """Supprime les fichiers existants et télécharge les fichiers de janvier 2018 à août 2023."""
    base_url = "https://d37ci6vzurnyck.cloudfront.net/trip-data/" # URL de base des fichiers de données
    years = range(2023, 2024) # Années à traiter
    months = range(1, 3) # Mois à traiter
    data_dir = "C:/Users/DELL/Downloads/ATL-Datamart-main/ATL-Datamart-main/data/raw" # Répertoire local pour stocker les fichiers

    # Supprimer les fichiers existants dans le répertoire cible
    if os.path.exists(data_dir):
        shutil.rmtree(data_dir) # Suppression du répertoire existant et de son contenu
    os.makedirs(data_dir, exist_ok=True) # Création du répertoire cible

    # Télécharger les fichiers pour chaque année et mois spécifié
    for year in years:
        for month in months:
            if year == 2024 and month > 2: # Limiter les téléchargements jusqu'en février 2024
                break
            filename = f"yellow_tripdata_{year}-{month:02d}.parquet" # Nom du fichier à télécharger
            file_url = base_url + filename # URL complète du fichier
            output_path = os.path.join(data_dir, filename) # Chemin local pour enregistrer le fichier
            try:
                # Téléchargement du fichier depuis l'URL et sauvegarde locale
                urllib.request.urlretrieve(file_url, output_path)
                print(f"Downloaded {filename}") # Confirmation du téléchargement réussi
            except Exception as e:
                # Gestion des échecs de téléchargement
                print(f"Failed to download {filename}: {e}")
```

- **Base URL** : Lien source des données.
- **Années et Mois** : La boucle itère sur une plage spécifique (2023 et début 2024).
- **Téléchargement des fichiers** : Les fichiers sont téléchargés et sauvegardés localement à l'aide de urllib.request.urlretrieve.

- **Gestion des erreurs** : Si un fichier ne peut pas être téléchargé, une exception est gérée pour éviter l'interruption du programme.

2.2.2. Organisation des Données Localement

Avant de télécharger de nouvelles données, tous les fichiers existants dans le répertoire cible sont supprimés à l'aide de `shutil.rmtree`.

Le répertoire est recréé avec `os.makedirs`, garantissant un environnement propre pour chaque exécution.

```
# Supprimer les fichiers existants dans le répertoire cible
if os.path.exists(data_dir):
    shutil.rmtree(data_dir) # Suppression du répertoire existant et de son contenu
os.makedirs(data_dir, exist_ok=True) # Création du répertoire cible
```

2.2.3. Chargement des Données sur MinIO

La fonction `write_data_minio` gère le transfert des fichiers `.parquet` locaux vers un serveur MinIO.

```
def write_data_minio(bucket_name: str, folder_path: str):
    """Charge les fichiers .parquet téléchargés sur un bucket MinIO."""
    client = Minio(
        "localhost:9000",      # Adresse du serveur MinIO
        secure=False,          # Connexion non sécurisée (HTTP)
        access_key="minio",    # Clé d'accès MinIO
        secret_key="minio123"  # Clé secrète MinIO
    )
    bucket: str = "newyork-data-bucket" # Nom du bucket MinIO cible

    # Vérifier l'existence du bucket, le créer s'il n'existe pas
    found = client.bucket_exists(bucket)
    if not found:
        client.make_bucket(bucket) # Création du bucket
    else:
        print("Bucket " + bucket + " existe déjà")

    # Parcours des fichiers dans le dossier et upload sur MinIO
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name) # Chemin complet du fichier

        # Vérification que le fichier est bien un fichier .parquet
        if os.path.isfile(file_path) and file_name.endswith(".parquet"):
            try:
                # Upload du fichier sur MinIO
                client.fput_object(
                    bucket_name=bucket_name, # Nom du bucket cible
                    object_name=file_name,   # Nom de l'objet dans MinIO
                    file_path=file_path      # Chemin local du fichier
                )
                print(f"Fichier '{file_name}' téléchargé dans le bucket '{bucket_name}'.")
            except Exception as e:
                # Gestion des échecs d'upload
                print(f"Erreur lors du téléchargement de {file_name}: {e}")
```

Étapes principales :

- **Connexion au serveur MinIO** : Création d'un client utilisant des clés d'accès (access_key et secret_key).
- **Vérification et création du bucket** : Si le bucket n'existe pas, il est créé.
- **Parcours des fichiers locaux** : Les fichiers .parquet sont détectés et chargés dans le bucket.

Gestion des erreurs : Si un upload échoue, une exception est gérée avec un message explicatif.

2.3. Architecture ETL Mise en Place

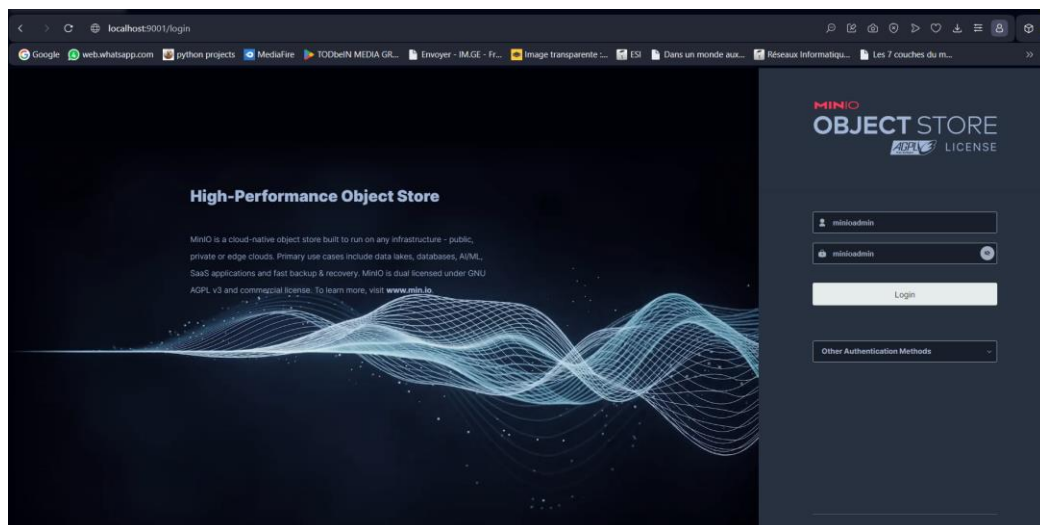
- **Extraction** :
 - ❖ Téléchargement des fichiers depuis une source web.
 - ❖ Suppression des fichiers précédents pour garantir une nouvelle extraction à chaque exécution.
- **Transformation** :
 - ❖ Bien que ce TP ne traite pas encore la transformation, les fichiers téléchargés sont organisés localement.
- **Chargement** :
 - ❖ Les fichiers '.parquet' sont transférés sur un serveur MinIO, un service de stockage cloud local.

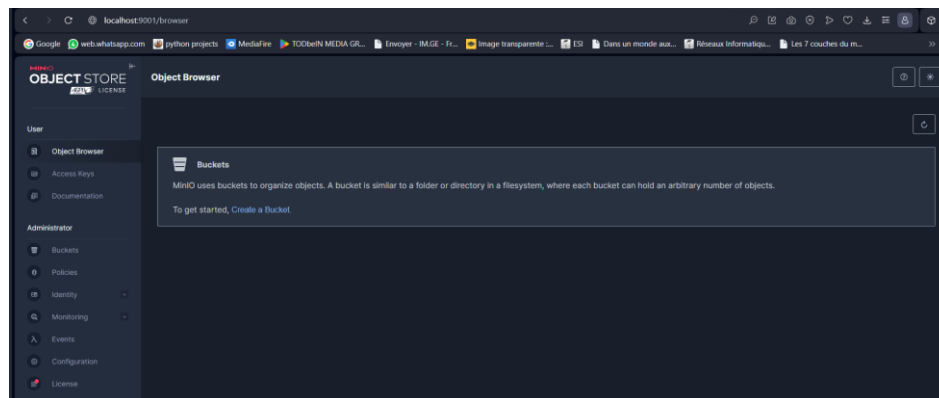
2.4. Exécution

2.4.1. Accéder à MinIO (interface web) :

Ouvrons un navigateur et accédez à l'interface MinIO :
<http://localhost:9000> et Connectons-nous avec les identifiants :

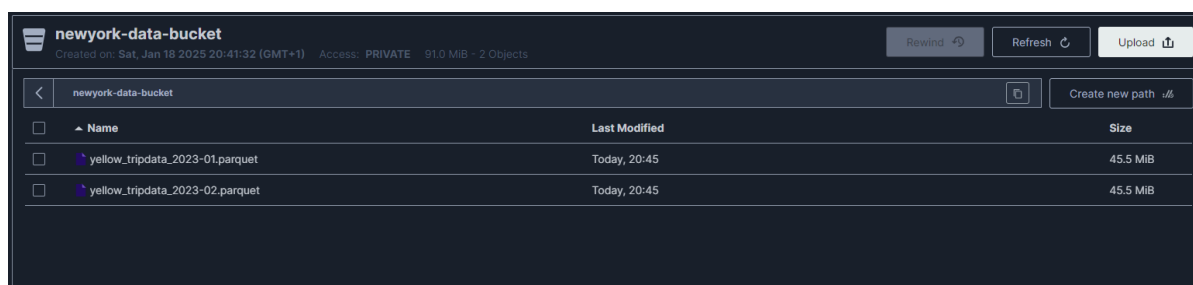
Identifiant : minio | **Mot de passe** : minio123





2.4.2. Exécutons le script *grab_parquet.py*

```
PS C:\Users\DELL\Downloads\ATL-Datamart-main\ATL-Datamart-main\src\data> python grab_parquet.py
Downloaded yellow_tripdata_2023-01.parquet
Downloaded yellow_tripdata_2023-02.parquet
Bucket newyork-data-bucket existe déjà
Fichier 'yellow_tripdata_2023-01.parquet' téléchargé dans le bucket 'newyork-data-bucket'.
Fichier 'yellow_tripdata_2023-02.parquet' téléchargé dans le bucket 'newyork-data-bucket'.
```



3. TP2 : Transfert des données depuis le Data Lake (MinIO) vers le Data Warehouse (PostgreSQL)

3.1. Objectif

L'objectif de ce TP est de récupérer des fichiers au format Parquet depuis un Data Lake (MinIO) et de les injecter dans un Data Warehouse (PostgreSQL) en optimisant le processus pour réduire le temps d'injection et s'assurer que les données sont bien formatées.

3.2. Contexte technique

Data Lake : MinIO (Object Storage).

Data Warehouse : PostgreSQL.

Format des fichiers : Parquet (efficace pour l'analytique).

Bibliothèques utilisées :

- pandas : Manipulation et transformation des données.

- sqlalchemy : Connexion et injection des données dans PostgreSQL.
- pyarrow : Lecture des fichiers Parquet.

3.3. Architecture et étapes du TP

3.3.1. Configuration de MinIO :

- Création d'un client MinIO pour interagir avec le Data Lake.
- Configuration des accès (adresse, clés d'accès).

```
def get_minio_client():  
    """Create and return a MinIO client instance"""  
    return Minio(  
        "localhost:9000",  
        access_key="minio",  
        secret_key="minio123",  
        secure=False  
    )
```

3.3.2. Lecture des fichiers Parquet depuis MinIO :

- Récupération des fichiers depuis le bucket du Data Lake.
- Utilisation de io.BytesIO pour manipuler les données en mémoire.
- Conversion des données Parquet en DataFrame avec pandas.

```
def read_parquet_from_minio(client, bucket_name, object_name):  
    """Read a parquet file from MinIO into a pandas DataFrame"""  
    try:  
        # Get the object data  
        data = client.get_object(bucket_name, object_name).read()  
        # Create a buffer from the data  
        buffer = io.BytesIO(data)  
        # Read the parquet file using pandas  
        df = pd.read_parquet(buffer, engine='pyarrow')  
        # Convert column names to lowercase  
        df.columns = df.columns.str.lower()  
        return df  
    except Exception as e:  
        print(f"Error reading parquet file {object_name}: {e}")  
        return None
```

3.3.3. Création de l'engine PostgreSQL :

- Connexion au Data Warehouse avec SQLAlchemy.
- Utilisation des paramètres de connexion sécurisés.

```
# Create PostgreSQL engine  
db_engine = create_engine(  
    "postgresql://postgres:admin@localhost:15432/nyc_warehouse"  
)
```

3.3.4. Injection des données dans PostgreSQL :

- Écriture des données dans une table nyc_raw avec pandas.to_sql.
- Optimisation de l'injection :
 - o Mode "append" pour les fichiers suivants.
 - o Utilisation du mode method='multi' et chunksize=10000 pour gérer les gros volumes de données.



```
if df is not None:
    # Write to PostgreSQL
    df.to_sql(
        name="nyc_raw",
        con=db_engine,
        if_exists="append" if i > 0 else "replace",
        index=False,
        method='multi',
        chunksize=10000
    )
    print(f"Successfully processed {parquet_file}")
```

3.4. Exécution

```
PS C:\Users\DELL\Downloads\ATL-Datamart-main\ATL-Datamart-main\src\data> python dump_to_sql.py
Processing yellow_tripdata_2023-01.parquet...
Successfully processed yellow_tripdata_2023-01.parquet
Processing yellow_tripdata_2023-02.parquet...
Successfully processed yellow_tripdata_2023-02.parquet
```

[Containers](#) / atl-datamart-main-data-warehouse-1

atl-datamart-main-data-warehouse-1

 3d80a6f81d0d  postgres:latest
[15432:5432](#)

STATUS
Running (2 hours ago)



Logs Inspect Bind mounts Exec Files Stats

```
1 s, sync=0.009 s, total=269.936 s; sync files=2, longest=0.009 s, average=0.005 s; distance=79455 kB, estimate=83950 kB; lsn=0/27919A80, redo lsn=0/23A46770
2025-01-18 22:28:18 2025-01-18 21:28:18.063 UTC [60] LOG: checkpoint starting: time
2025-01-18 22:32:48 2025-01-18 21:32:48.087 UTC [60] LOG: checkpoint complete: wrote 5022 buffers (30.7%); 0 WAL file(s) added, 0 removed, 4 recycled; write=269.95
6 s, sync=0.015 s, total=270.025 s; sync files=8, longest=0.009 s, average=0.002 s; distance=68414 kB, estimate=82396 kB; lsn=0/2C8EAD38, redo lsn=0/27D16120
2025-01-18 22:33:18 2025-01-18 21:33:18.104 UTC [60] LOG: checkpoint starting: time
2025-01-18 22:37:48 2025-01-18 21:37:48.129 UTC [60] LOG: checkpoint complete: wrote 8901 buffers (54.3%); 0 WAL file(s) added, 0 removed, 6 recycled; write=269.90
6 s, sync=0.055 s, total=270.025 s; sync files=8, longest=0.039 s, average=0.007 s; distance=85800 kB, estimate=85800 kB; lsn=0/30C91200, redo lsn=0/2D0E01B8
2025-01-18 22:38:18 2025-01-18 21:38:18.145 UTC [60] LOG: checkpoint starting: time
2025-01-18 22:42:48 2025-01-18 21:42:48.059 UTC [60] LOG: checkpoint complete: wrote 7558 buffers (46.1%); 0 WAL file(s) added, 0 removed, 4 recycled; write=269.86
9 s, sync=0.007 s, total=269.915 s; sync files=2, longest=0.006 s, average=0.004 s; distance=69273 kB, estimate=84147 kB; lsn=0/35A29158, redo lsn=0/31486680
2025-01-18 22:43:18 2025-01-18 21:43:18.083 UTC [60] LOG: checkpoint starting: time
2025-01-18 22:47:48 2025-01-18 21:47:48.023 UTC [60] LOG: checkpoint complete: wrote 8446 buffers (51.6%); 0 WAL file(s) added, 0 removed, 5 recycled; write=269.85
1 s, sync=0.014 s, total=269.941 s; sync files=2, longest=0.009 s, average=0.007 s; distance=77418 kB, estimate=83474 kB; lsn=0/3A9BF370, redo lsn=0/360210F0
2025-01-18 22:48:18 2025-01-18 21:48:18.034 UTC [60] LOG: checkpoint starting: time
2025-01-18 22:52:48 2025-01-18 21:52:48.029 UTC [60] LOG: checkpoint complete: wrote 9114 buffers (55.6%); 0 WAL file(s) added, 0 removed, 5 recycled; write=269.96
8 s, sync=0.011 s, total=269.996 s; sync files=2, longest=0.007 s, average=0.006 s; distance=83533 kB, estimate=83533 kB; lsn=0/3FB52DF8, redo lsn=0/3B1B47B8
2025-01-18 22:53:18 2025-01-18 21:53:18.056 UTC [60] LOG: checkpoint starting: time
2025-01-18 22:57:48 2025-01-18 21:57:48.131 UTC [60] LOG: checkpoint complete: wrote 9113 buffers (55.6%); 0 WAL file(s) added, 0 removed, 5 recycled; write=269.98
5 s, sync=0.016 s, total=270.075 s; sync files=2, longest=0.010 s, average=0.008 s; distance=83534 kB, estimate=83534 kB; lsn=0/44CE4C48, redo lsn=0/40348240
2025-01-18 22:58:18 2025-01-18 21:58:18.145 UTC [60] LOG: checkpoint starting: time
2025-01-18 23:02:48 2025-01-18 22:02:48.105 UTC [60] LOG: checkpoint complete: wrote 9113 buffers (55.6%); 0 WAL file(s) added, 0 removed, 5 recycled; write=269.92
3 s, sync=0.008 s, total=269.961 s; sync files=2, longest=0.005 s, average=0.004 s; distance=83527 kB, estimate=83533 kB; lsn=0/4987EF40, redo lsn=0/454DA090
2025-01-18 23:03:18 2025-01-18 22:03:18.134 UTC [60] LOG: checkpoint starting: time
```

Affichage d'un petit échantillon de la dataset en utilisant :

docker exec -it 3d80a6f81d0d8b5b88edacada54eadfe9046a266f32cfb04f2b1a0de9793ad95 psql -U postgres -d nyc_warehouse -c "SELECT * FROM nyc_raw LIMIT 10;"

avec: 3d80a6f81d0d8b5b88edacada54eadfe9046a266f32cfb04f2b1a0de9793ad95 est le <container_id>

Terminal												
vendorid	time_pickup_datetime	time_dropoff_datetime	passenger_count	trip_distance	ratecodeid	store_and_fwd_flag	pickup_locationid	dropoff_locationid	payment_type	fare_amount	extra	mta_tax
2	2023-01-01 00:32:10	2023-01-01 00:40:36	1	0.97	1	N	161					
141	2023-01-01 00:32:10	2023-01-01 00:40:36	1	0.97	1	N	161					
2.5	2023-01-01 00:32:10	2023-01-01 00:40:36	1	0.97	1	N	161					
2	2023-01-01 00:55:08	2023-01-01 01:01:27	1	1.1	1	N	43					

Avec la requête count(*) on trouve qu'on a **5980721** lignes dans notre Dataset

```
PS C:\Users\DELL> docker exec -it 3d80a6f81d0d8b5b88edacada54eadfe9046a266f32cfb04f2b1a0de9793ad95 psql -U postgres -d nyc_warehouse -c "SELECT count(*) FROM nyc_raw;"
count
-----
5980721
(1 row)
```

4. TP3 : Conception d'un Data Mart avec PostgreSQL : Implémentation d'un Modèle Flocon

Le TP3 du projet ATL-Datamart consiste à concevoir un modèle en flocon pour structurer et organiser les données issues de notre Data Warehouse dans une base de données Data Mart. Cette tâche inclut la création des tables nécessaires et l'insertion des données dans la base cible

4.1. Objectif

L'objectif principal était de créer des tables en modèle Flocon dans une base de données dédiée et de préparer des requêtes SQL pour l'insertion des données à partir du Data Warehouse vers le Data Mart.

Détails des étapes réalisées :

- Création des tables de dimension (dim_time, dim_location, dim_payment_type) et de la table factuelle (fact_trip).
- Insertion des données dans les tables en utilisant des requêtes SQL avec dblink pour se connecter au Data Warehouse.

4.2. Détails des étapes réalisées

4.2.1. Création des tables (creation.sql)

```
CREATE TABLE dim_time (  
    time_id SERIAL PRIMARY KEY,  
    datetime TIMESTAMPT,  
    hour INTEGER,  
    day INTEGER,  
    month INTEGER,  
    year INTEGER  
);
```

```
CREATE TABLE dim_location (  
    location_id SERIAL PRIMARY KEY,  
    location_name TEXT,  
    borough TEXT,  
    latitude FLOAT,  
    longitude FLOAT  
);
```

```
CREATE TABLE dim_time (  
    time_id SERIAL PRIMARY KEY,  
    datetime TIMESTAMPT,  
    hour INTEGER,  
    day INTEGER,  
    month INTEGER,  
    year INTEGER  
);
```

```
CREATE TABLE dim_payment_type (  
    payment_type_id SERIAL PRIMARY KEY,  
    payment_type TEXT  
);
```

```
CREATE TABLE fact_trip (  
    trip_id SERIAL PRIMARY KEY,  
    pickup_datetime TIMESTAMPT,  
    dropoff_datetime TIMESTAMPT,  
    passenger_count INTEGER,  
    trip_distance FLOAT,  
    fare_amount FLOAT,  
    total_amount FLOAT,  
    payment_type_id INTEGER REFERENCES dim_payment_type(payment_type_id),  
    pickup_location_id INTEGER REFERENCES dim_location(location_id),  
    dropoff_location_id INTEGER REFERENCES dim_location(location_id),  
    time_id INTEGER REFERENCES dim_time(time_id)  
);
```

Table dim_time : Contient les informations temporelles liées aux trajets, incluant des colonnes comme l'heure, le jour, le mois et l'année.

Table dim_location : Stocke les informations sur les lieux de prise en charge et de dépose, y compris les coordonnées géographiques (latitude et longitude).

Table dim_payment_type : Enregistre les types de paiement utilisés pour les trajets.

Table fact_trip : C'est la table factuelle reliant les autres tables de dimensions à travers des clés étrangères et contenant des informations sur chaque trajet (heure, distance, montant, nombre de passagers, etc.).

4.2.2. Insertion des données (insertion.sql)

Connexion au Data Warehouse via dblink : La connexion au serveur PostgreSQL du Data Warehouse a été établie à l'aide de la fonction dblink_connect.

```
SELECT dblink_connect('etoile_en_flocon', 'host=data-warehouse port=5432 dbname=nyc_warehouse user=postgres password=admin');
```

Insertion dans dim_location et dim_payment_type : Les données sont extraites du Data Warehouse en utilisant dblink pour remplir les tables de dimension avec les informations uniques de localisation et de type de paiement.

```

> Run
INSERT INTO dim_location (location_name, borough, latitude, longitude)
SELECT DISTINCT pulocationid, NULL, CAST(NULL AS double precision), CAST(NULL AS double precision) FROM dblink('etoile_en_flocon', 'SELECT DISTINCT pulocationid FROM nyc_raw') AS t(pulocationid INTEGER);

> Run
INSERT INTO dim_location (location_name, borough, latitude, longitude)
SELECT DISTINCT dolocationid, NULL, CAST(NULL AS double precision), CAST(NULL AS double precision) FROM dblink('etoile_en_flocon', 'SELECT DISTINCT dolocationid FROM nyc_raw') AS t(dolocationid INTEGER);

> Run
INSERT INTO dim_payment_type (payment_type)
SELECT DISTINCT payment_type FROM dblink('etoile_en_flocon', 'SELECT DISTINCT payment_type FROM nyc_raw') AS t(payment_type TEXT);

```

Insertion dans dim_time : Extraction et transformation des dates et heures des trajets pour être insérées dans la table dim_time.

```

> Run | Select
INSERT INTO dim_time (datetime, hour, day, month, year)
SELECT DISTINCT tpep_pickup_datetime,
EXTRACT(HOUR FROM tpep_pickup_datetime) AS hour,
EXTRACT(DAY FROM tpep_pickup_datetime) AS day,
EXTRACT(MONTH FROM tpep_pickup_datetime) AS month,
EXTRACT(YEAR FROM tpep_pickup_datetime) AS year
FROM dblink('etoile_en_flocon', 'SELECT DISTINCT tpep_pickup_datetime FROM nyc_raw') AS t(tpep_pickup_datetime TIMESTAMP);

```

Insertion dans fact_trip : Les données des trajets, y compris les informations sur les lieux, le type de paiement et le temps, sont insérées dans la table fact_trip avec les bonnes relations de clés étrangères.

Déconnexion : Une fois l'insertion terminée, la connexion au Data Warehouse est fermée avec dblink_disconnect.

une requête SQL complexe insérant des données dans la table factuelle fact_trip à partir d'une

```
D:\Run\J>psql
INSERT INTO fact_trip (pickup_datetime, dropoff_datetime, passenger_count, trip_distance, fare_amount, total_amount, payment_type_id, pickup_location_id, dropoff_location_id, time_id)
SELECT tpep_pickup_datetime,
       tpep_dropoff_datetime,
       passenger_count,
       trip_distance,
       fare_amount,
       total_amount,
       dim_payment_type.payment_type_id,
       dim_location_pu.location_id,
       dim_location_do.location_id,
       dim_time.time_id
FROM dblink('etolie_en_flocon', 'SELECT tpep_pickup_datetime::timestamp,
       tpep_dropoff_datetime::timestamp,
       passenger_count::integer,
       trip_distance::double precision,
       fare_amount::double precision,
       total_amount::double precision,
       payment_type,
       pulocationid::integer,
       dolocationid::integer
FROM nyc_taxi')
AS t(tpep_pickup_datetime TIMESTAMPTZ, tpep_dropoff_datetime TIMESTAMPTZ, passenger_count INTEGER, trip_distance DOUBLE PRECISION, fare_amount DOUBLE PRECISION, total_amount DOUBLE PRECISION, payment_type TEXT, pulocationid INTEGER, dolocationid INTEGER)
JOIN dim_payment_type ON t.payment_type = dim_payment_type.payment_type
JOIN dim_location AS dim_location_pu ON t.pulocationid = dim_location_pu.location_id
JOIN dim_location AS dim_location_do ON t.dolocationid = dim_location_do.location_id
JOIN dim_time ON t.tpep_pickup_datetime = dim_time.datetime;

D:\Run\J>Tab\JSON
SELECT dblink_disconnect('etolie_en_flocon');
```

source externe via dblink. Les données sont transformées et jointes à plusieurs tables de dimensions (dim_payment_type, dim_location, dim_time) pour assurer l'intégrité référentielle. La requête est bien structurée, mais il serait utile d'ajouter des commentaires pour documenter le rôle de chaque jointure et valider les types de données pour garantir la compatibilité entre les colonnes.

4.3. Tests et Validation

Transfert des fichiers SQL dans le conteneur Docker : Les fichiers insertion.sql et creation.sql ont été copiés avec succès depuis la machine locale vers un conteneur Docker, prêts à être utilisés pour la création des tables et l'insertion des données.

```
PS C:\Users\DELL\Downloads\ATL-Datamart-main\ATL-Datamart-main> docker cp insertion.sql 309569cd35e575eb0a3e61ce53c7ad583bbb2329044f6f85b5d8e27fc45889f7:/insertion.sql
Successfully copied 5.12kB to 309569cd35e575eb0a3e61ce53c7ad583bbb2329044f6f85b5d8e27fc45889f7:/insertion.sql
PS C:\Users\DELL\Downloads\ATL-Datamart-main\ATL-Datamart-main> docker cp creation.sql 309569cd35e575eb0a3e61ce53c7ad583bbb2329044f6f85b5d8e27fc45889f7:/creation.sql
Successfully copied 2.56kB to 309569cd35e575eb0a3e61ce53c7ad583bbb2329044f6f85b5d8e27fc45889f7:/creation.sql
PS C:\Users\DELL\Downloads\ATL-Datamart-main\ATL-Datamart-main>
```

Connexion à PostgreSQL (version 17.2 sur Debian)

Exécution du fichier creation.sql qui crée plusieurs tables comme indiqué par les "CREATE TABLE" répétés

les données ont été chargées avec succès

```
DETAIL:  consistent recovery state was
root@309569cd35e5:/# psql -U postgres
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

postgres=# \i /creation.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

```
dblink_connect
-----
OK
(1 row)

INSERT 0 260
INSERT 0 261
INSERT 0 5
INSERT 0 3104949
INSERT 0 5980721
```


Exercice 1 : Configuration et prise en main

```
configuration.yml
1 data_source nyc_warehouse:
2   type: postgres
3   connection:
4     host: localhost
5     port: 15432
6     username: postgres
7     password: admin
8     database: mydatabase
9     schema: public
```

Creation de configuration.yml et Teste de la bonne configuration de Soda avec la ligne de commande : "soda test-connection -d nyc_warehouse -c configuration.yml"

```
PS C:\Users\VHP\Downloads\ATL-Datamart> py -3.13 -m soda test-connection -d nyc_warehouse -c configuration.yml
[23:24:16] Soda Core 3.4.3
Successfully connected to 'nyc_warehouse'.
Connection 'nyc_warehouse' is valid.
PS C:\Users\VHP\Downloads\ATL-Datamart> □
```

```
check.yml
1 checks for mytable:
2   - schema:
3     warn:
4       when required column missing:
5         - VendorID
6         - tpep_pickup_datetime
7         - tpep_dropoff_datetime
8         - passenger_count
9         - trip_distance
10        - RatecodeID
11        - store_and_fwd_flag
12        - PULocationID
13        - DOLocationID
14        - payment_type
15        - fare_amount
16        - extra
17        - mta_tax
18        - tip_amount
19        - tolls_amount
20        - improvement_surcharge
21        - total_amount
22        - congestion_surcharge
23        - airport_fee
24     fail:
25       when forbidden column present:
26         - column_name
27         - column_name2
```

Création du fichier check.yml avec les colonnes dans notre dataset

Teste de la commande : soda scan -d nyc_warehouse -c configuration.yml check.yml

Ajout du "-V" en fin de commande. Nous constatons que le résultat du scan est plus détaillé avec notamment par exemple le nom du host, le port, le nom de la base de données, ...

```
PS C:\Users\HP\Downloads\ATL-Datamart> py -3.13 -m soda scan -d nyc_warehouse -c configuration.yml check.yml
[23:39:03] Soda Core 3.4.3
[23:39:24] Scan summary:
[23:39:24] 1/1 check PASSED:
[23:39:24]     mytable in nyc_warehouse
[23:39:24]     Schema Check [PASSED]
[23:39:24] All is good. No failures. No warnings. No errors.
PS C:\Users\HP\Downloads\ATL-Datamart> 
```

La table mytable est conforme au schéma attendu, et aucun problème n'a été détecté. Cela montre que la configuration et les vérifications de qualité des données définies dans les fichiers YAML ont été correctement respectées.

```
PS C:\Users\HP\Downloads\ATL-Datamart> py -3.13 -m soda scan -d nyc_warehouse -c configuration.yml check.yml -V
[23:41:53] Soda Core 3.4.3
[23:41:53] Reading configuration file "configuration.yml"
[23:41:53] Reading SodaCL file "check.yml"
[23:41:53] Scan execution starts
[23:41:53] Postgres connection properties: host="localhost", port="15432", database="mydatabase", user="postgres", options="-c search_path=public", connection_timeout="None"
[23:41:54] Query 1.nyc_warehouse.mytable.schema[mytable]:
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE lower(table_name) = 'mytable'
      AND lower(table_catalog) = 'mydatabase'
      AND lower(table_schema) = 'public'
ORDER BY ORDINAL_POSITION
[23:41:55] Scan summary:
[23:41:55] 1/1 query OK
[23:41:55] 1.nyc_warehouse.mytable.schema[mytable] [OK] 0:00:00.635298
[23:41:55] 1/1 check PASSED:
[23:41:55]     mytable in nyc_warehouse
[23:41:55]     Schema Check [check.yml] [PASSED]
[23:41:55]     schema measured = [VendorID bigint, tpep_pickup_datetime timestamp without time zone, tpep_dropoff_datetime timestamp without time zone, passenger_count bigint, trip_distance double precision, RatecodeID bigint, store_and_fwd_flag text, PULocationID bigint, DOLocationID bigint, payment_type bigint, fare_amount double precision, extra double precision, mta_tax double precision, tip_amount double precision, tolls_amount double precision, improvement_surcharge double precision, total_amount double precision, congestion_surcharge double precision, airport_fee double precision]
[23:41:55] All is good. No failures. No warnings. No errors.
PS C:\Users\HP\Downloads\ATL-Datamart> 
```

```
YML check.yml
1 checks for mytable:
2   - row_count:
3     fail: when <= 0
4
```

```
YML check.yml
1 checks for mytable:
2   - row_count:
3     warn: when > 90
4     fail: when = 0
5
```

Changement du code du fichier check.yml par la règle suivante : Nombre de ligne supérieur à 0. (À l'aide de row_count)

- Réécriture du "row_count" sous forme de warning et failure.

Nous constatons le déclenchement d'un avertissement car le nombre de ligne supérieur à 90.

```
PS C:\Users\HP\Downloads\ATL-Datamart> py -3.13 -m soda scan -d nyc_warehouse -c configuration.yml check.yml
[00:16:48] Soda Core 3.4.3
[00:17:30] Scan summary:
[00:17:30] 1/1 check WARNED:
[00:17:30]   mytable in nyc_warehouse
[00:17:30]     row_count warn when > 90 fail when = 0 [WARNED]
[00:17:30]     check_value: 3066766
[00:17:30] Only 1 warning. 0 failure. 0 errors. 0 pass.
PS C:\Users\HP\Downloads\ATL-Datamart>
```

- ② Un seul avertissement est détecté.
- ② Aucune erreur ou échec n'a été trouvé.
- ② Tous les autres tests (s'il y en avait) ont réussi.

Exercice 2: Configuration et prise en main

Élaboration des règles de qualité des données avant la conversion de la base de données au modèle flacons.

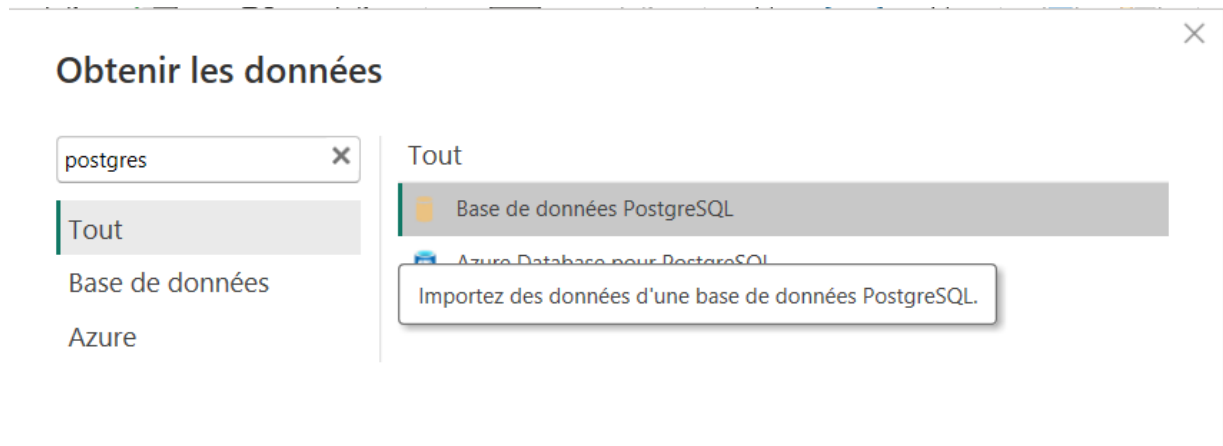
```
! fact_trip.yaml
1  checks for fact_trip:
2    - schema:
3      warn:
4        when required column missing:
5          - trip_id
6          - pickup_datetime
7          - dropoff_datetime
8          - passenger_count
9          - trip_distance
10         - fare_amount
11         - total_amount
12         - payment_type_id
13         - pickup_location_id
14         - dropoff_location_id
15         - time_id
16       fail:
17         when forbidden column present:
18           - column_name1
19           - column_name2
20       rows:
21         - passenger_count > 0
22         - trip_distance >= 0
23         - fare_amount >= 0
24         - total_amount >= fare_amount
25         - dropoff_datetime > pickup_datetime
26       relationships:
27         - ref dim_payment_type: payment_type_id
28         - ref dim_location: pickup_location_id
29         - ref dim_location: dropoff_location_id
30         - ref dim_time: time_id
```

```
! dim_time.yaml
1  # checks for dim_time.yml
2  checks for dim_time:
3    - schema:
4        warn:
5            when required column missing:
6                - time_id
7                - datetime
8                - hour
9                - day
10               - month
11               - year
12            fail:
13                when forbidden column present:
14                    - column_name1
15                    - column_name2
16            rows:
17                - hour between 0 and 23
18                - day between 1 and 31
19                - month between 1 and 12
20                - year between 2000 and 2025
```

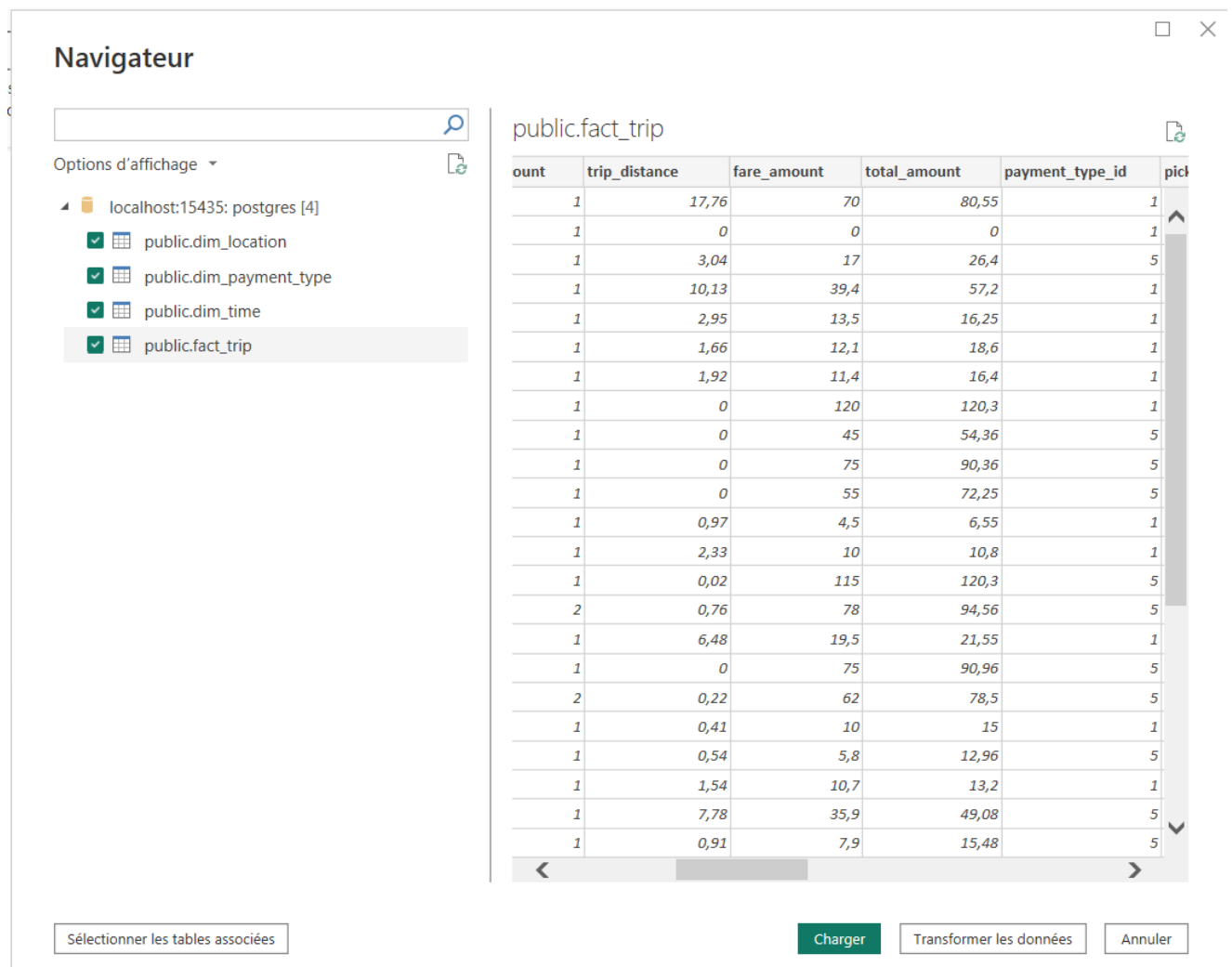
```
! dim_payment_type.yaml
1  checks for dim_payment_type:
2    - schema:
3        warn:
4            when required column missing:
5                - payment_type_id
6                - payment_type
7            fail:
8                when forbidden column present:
9                    - column_name1
10                   - column_name2
11            valid values:
12                payment_type:
13                    - 'Credit card'
14                    - 'Cash'
15                    - 'No charge'
16                    - 'Dispute'
17                    - 'Unknown'
```

```
! dim_location.yaml
1 ✓ checks for dim_location:
2 ✓   - schema:
3 ✓     warn:
4 ✓       when required column missing:
5         - location_id
6         - location_name
7         - borough
8         - latitude
9         - longitude
10 ✓     fail:
11 ✓       when forbidden column present:
12         - column_name1
13         - column_name2
14 ✓     rows:
15       - latitude between -90 and 90
16       - longitude between -180 and 180
```

5. TP4 : Visualisation



Interface de PowerBI pour importer des données de PostgreSQL



Navigateur

Options d'affichage ▾

- localhost:15435: postgres [4]
 - ☒ public.dim_location
 - ☒ public.dim_payment_type
 - ☒ public.dim_time
 - ☒ public.fact_trip

public.fact_trip

ount	trip_distance	fare_amount	total_amount	payment_type_id	picku
1	17,76	70	80,55	1	1
1	0	0	0	1	1
1	3,04	17	26,4	5	5
1	10,13	39,4	57,2	1	1
1	2,95	13,5	16,25	1	1
1	1,66	12,1	18,6	1	1
1	1,92	11,4	16,4	1	1
1	0	120	120,3	1	1
1	0	45	54,36	5	5
1	0	75	90,36	5	5
1	0	55	72,25	5	5
1	0,97	4,5	6,55	1	1
1	2,33	10	10,8	1	1
1	0,02	115	120,3	5	5
2	0,76	78	94,56	5	5
1	6,48	19,5	21,55	1	1
1	0	75	90,96	5	5
2	0,22	62	78,5	5	5
1	0,41	10	15	1	1
1	0,54	5,8	12,96	5	5
1	1,54	10,7	13,2	1	1
1	7,78	35,9	49,08	5	5
1	0,91	7,9	15,48	5	5

Sélectionner les tables associées

Charger Transformer les données Annuler

La structure de la base de données avec plusieurs tables :

- public.dim_location
- public.dim_payment_type
- public.dim_time
- public.fact_trip

Charger

- public dim_location
521 lignes depuis tcp://localhost:15435/postgres.
- public dim_payment_type
5 lignes depuis tcp://localhost:15435/postgres.
- public dim_time
610 404 lignes depuis tcp://localhost:15435/postgres.
- public fact_trip
299 333 lignes depuis tcp://localhost:15435/postgres.

Annuler

Chargeant les données sur PowerBI

-Voici la visualisation finale effectuée :

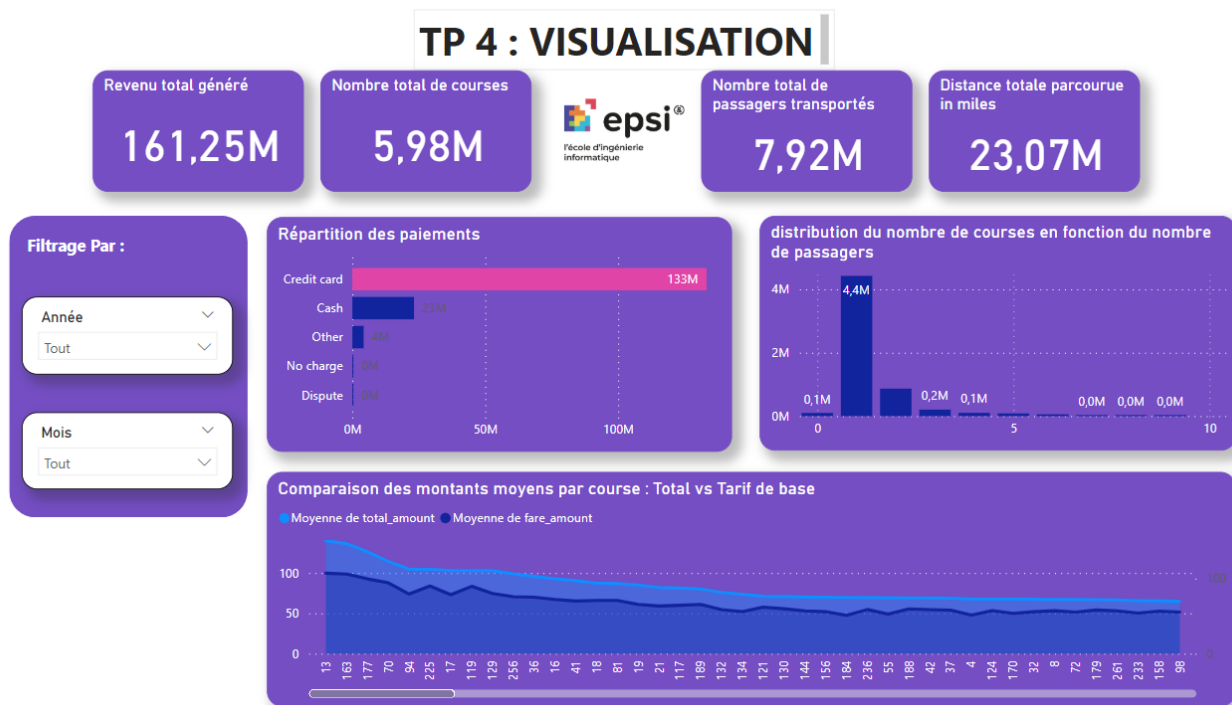


Tableau de bord de visualisation (TP4) montrant des statistiques clés :

- Revenu total : 161.25M
- Nombre total de courses : 5.98M
- Nombre de passagers : 7.92M
- Distance totale : 23.07M miles

Inclut des graphiques de répartition des paiements et d'analyse des courses avec un filtrage qui peut être effectué soit par Année ou/et par mois.

Pour télécharger le Dashboard Voici le lien (On n'a pas pu le mettre sur GitHub car il est volumineux 300Mo) :

https://drive.google.com/file/d/1eu1yH_ZxzGdmS094ZCG6SHPx6udEhsTe/view?usp=sharing

6. TP5 : Automatisation du TP1 avec Airflow

6.1. Téléchargement des données

La première étape du processus consiste à télécharger des fichiers de données à partir d'une source en ligne. Le script récupère les fichiers de données pour la période de janvier 2023 à février 2024 à partir d'un serveur HTTP, en vérifiant si les fichiers existent localement avant de les télécharger dans un répertoire dédié.

```
def grab_data_2023_to_2024():|
    base_url = "https://d37ci6vzurychx.cloudfront.net/trip-data/"
    years = range(2023, 2024)
    months = range(1, 3)
    data_dir = "C:/Users/DELL/Downloads/ATL-Datamart-main/ATL-Datamart-main/data/raw"

    # Supprimer les fichiers existants dans le dossier
    if os.path.exists(data_dir):
        shutil.rmtree(data_dir)
    os.makedirs(data_dir, exist_ok=True)

    for year in years:
        for month in months:
            filename = f"yellow_tripdata_{year}-{month:02d}.parquet"
            file_url = base_url + filename
            output_path = os.path.join(data_dir, filename)
            try:
                urllib.request.urlretrieve(file_url, output_path)
                print(f"Downloaded {filename}")
            except Exception as e:
                print(f"Failed to download {filename}: {e}")
```


6.2. Chargement vers MinIO

Une fois les données téléchargées, le script charge les fichiers .parquet dans un serveur MinIO local. MinIO est utilisé comme solution de stockage, permettant ainsi d'organiser et de gérer les fichiers dans un compartiment dédié. Le script vérifie si le compartiment existe, le crée si nécessaire, puis télécharge les fichiers locaux vers ce compartiment.

```
def write_data_minio():
    client = Minio(
        "minio:9000",
        secure=False,
        access_key="minio",
        secret_key="minio123"
    )
    bucket_name = "newyork-data-bucket"
    folder_path = "C:/Users/DELL/Downloads/ATL-Datamart-main/ATL-Datamart-main/data/raw"

    if not client.bucket_exists(bucket_name):
        client.make_bucket(bucket_name)

    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path) and file_name.endswith(".parquet"):
            try:
                client.fput_object(
                    bucket_name=bucket_name,
                    object_name=file_name,
                    file_path=file_path
                )
                print(f"Fichier '{file_name}' uploaded to bucket '{bucket_name}'")
            except Exception as e:
                print(f"Error uploading {file_name}: {e}")
```


6.3. Test et Validation

- 1- Premièrement on visite le site : localhost:8080 et on s'identifie avec les informations suivante : Username : airflow et Password : airflow

Sign In


Enter your login and password below:

Username:



airflow

Password:



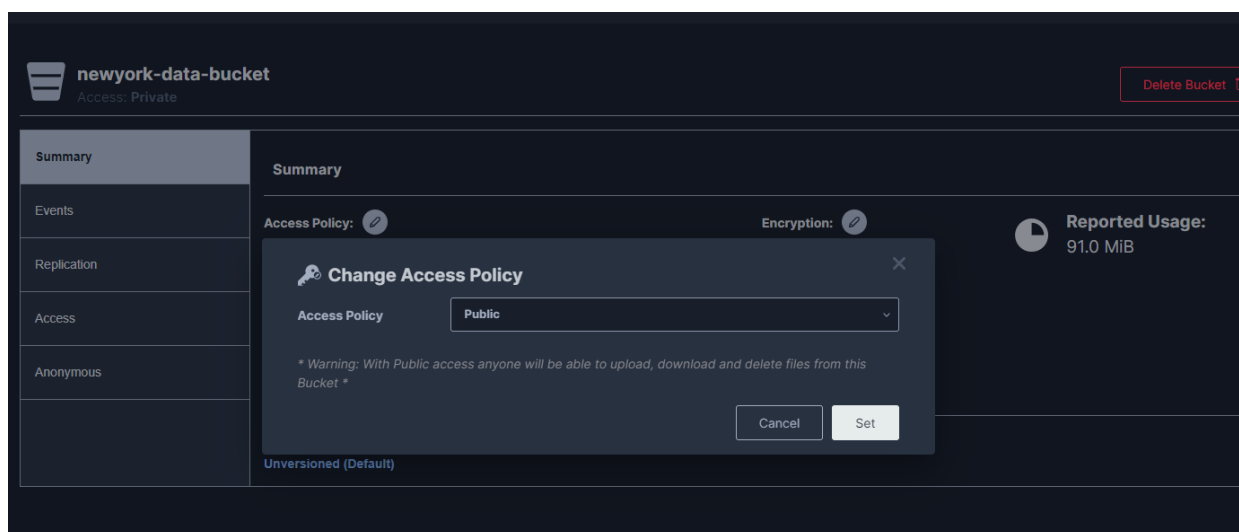
.....

Sign In

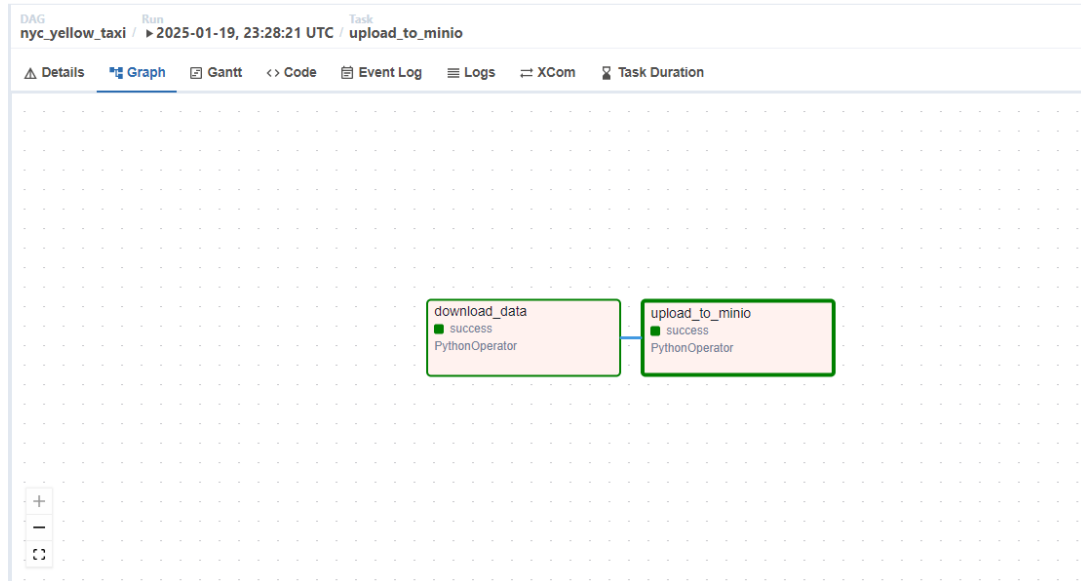
2- Après on cherche dans la barre de recherche notre DAG : nyc_yellow_taxi



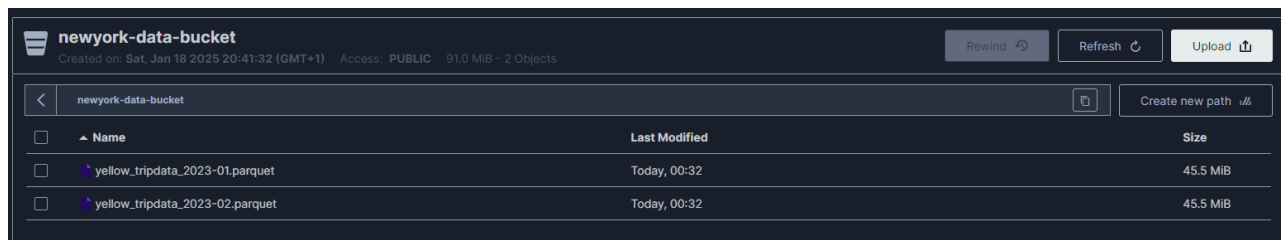
3- Avant de lancer la tâche, on doit changer le type d'accès dans minio du Privée à Public pour que Airflow peut uploader et modifier



4- On lance après la tache de nyc_yellow_taxi



5- Finalement, on vérifie dans MinIO que Upload est bien réussi :



newyork-data-bucket
Created on: Sat, Jan 18 2025 20:41:32 (GMT+1) Access: PUBLIC 91.0 MiB - 2 Objects

Rewind Refresh Upload

newyork-data-bucket Create new path

Name	Last Modified	Size
yellow_tripdata_2023-01.parquet	Today, 00:32	45.5 MiB
yellow_tripdata_2023-02.parquet	Today, 00:32	45.5 MiB

Lien Github : <https://github.com/imane25B/ATL-Datamart-Project-ArchitectureDecisionnel>

FIN

