TP : Data Warehouse Moderne

SCD Type 2, Data Vault 2.0 et Architecture Lakehouse

PySpark, Delta Lake et PostgreSQL

Réalisé par :

Khadija Nachid Idrissi

Aya Hamim

Rajae Fdili

Année universitaire : 2024–2025

Table des matières

Chapitre 1

Introduction et Vue d'Ensemble

1.1 Objectifs du TP

Ce travail pratique a pour objectif de construire un **Data Warehouse moderne complet**, en suivant une approche professionnelle utilisée dans les environnements industriels actuels.

À l'issue de ce TP, l'étudiant sera capable de :

— Installer et configurer un environnement de Data Warehousing moderne
— Comprendre en détail la connexion entre PySpark et PostgreSQL via JDBC
— Implémenter des **Slowly Changing Dimensions (SCD Type 2)**
— Concevoir un modèle **Data Vault 2.0**
— Mettre en place une architecture **Lakehouse** (Bronze, Silver, Gold)
— Exploiter **Delta Lake** pour les transactions ACID et le Time Travel

1.2 Architecture Globale du Projet

L'architecture globale du projet suit un pipeline de données moderne allant des systèmes transactionnels jusqu'aux outils de Business Intelligence.

1.2.1 Flux de Données

— **PostgreSQL (OLTP)** : source des données transactionnelles
— **Driver JDBC** : pont de communication entre Spark et PostgreSQL
— **PySpark** : ingestion, transformation et traitement des données
— **Lakehouse** :
  — Bronze : données brutes
  — Silver : données nettoyées et enrichies
  — Gold : données agrégées pour l'analyse

1.3 Prérequis

Catégorie	**Requis**	**Vérification**
Système	Windows / macOS / Linux	N/A
RAM	8 Go minimum	Gestionnaire de tâches

Espace disque	10 Go libres	Explorateur / df -h
Internet	Requis	N/A

1.4 Durée Estimée

La durée totale du TP est estimée entre **12 et 16 heures**, réparties sur les différentes sections d'installation, de connexion et de modélisation.

Chapitre 2

Installation et Configuration de PostgreSQL

2.1 Téléchargement de PostgreSQL

PostgreSQL est utilisé comme **base OLTP source** du pipeline.

2.2 Vérification de l'Installation

```
psql -U postgres
```

Listing 2.1 – Vérification de PostgreSQL

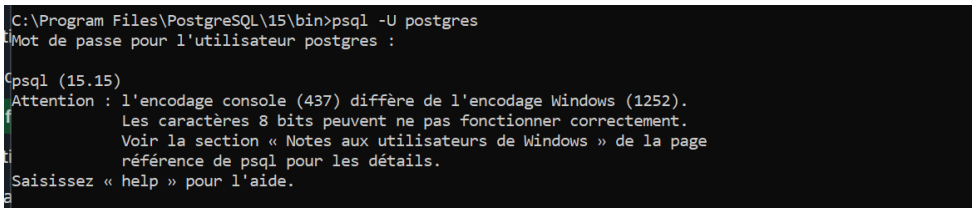


FIGURE 2.1 – Connexion réussie à PostgreSQL via pgAdmin

Chapitre 3

Installation de Python et PySpark

3.1 Installation de Python

Python 3.10 ou supérieur est requis.

```
python --version
```

Listing 3.1 – Vérification de Python

3.2 Création d'un Environnement Virtuel

```
python -m venv venv
venv\Scripts\activate
```

Listing 3.2 – Création et activation de l'environnement virtuel

3.3 Installation des Dépendances

```
pip install pyspark==3.5.0
pip install delta-spark==3.0.0
pip install psycopg2-binary==2.9.9
pip install pandas==2.1.4
```

Listing 3.3 – Installation des packages

3.4 Vérification de l'Installation

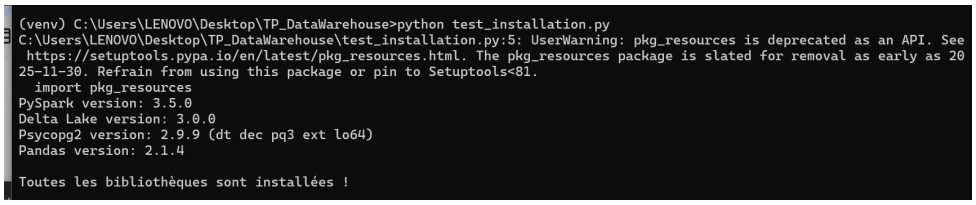


FIGURE 3.1 – Vérification de l'installation

Chapitre 4

Connexion PySpark à PostgreSQL

4.1 Principe de la Connexion JDBC

PySpark utilise un **driver JDBC** pour communiquer avec PostgreSQL, car Spark fonctionne sur la JVM Java.

PySpark → JDBC → PostgreSQL

4.2 Téléchargement du Driver JDBC

— Télécharger depuis `https://jdbc.postgresql.org/download/`
— Placer le fichier `.jar` dans un dossier `drivers`

4.3 Test de Connexion PySpark

Un script de test permet de valider la connexion JDBC et la lecture des données PostgreSQL.

Screenshot de succès attendu :

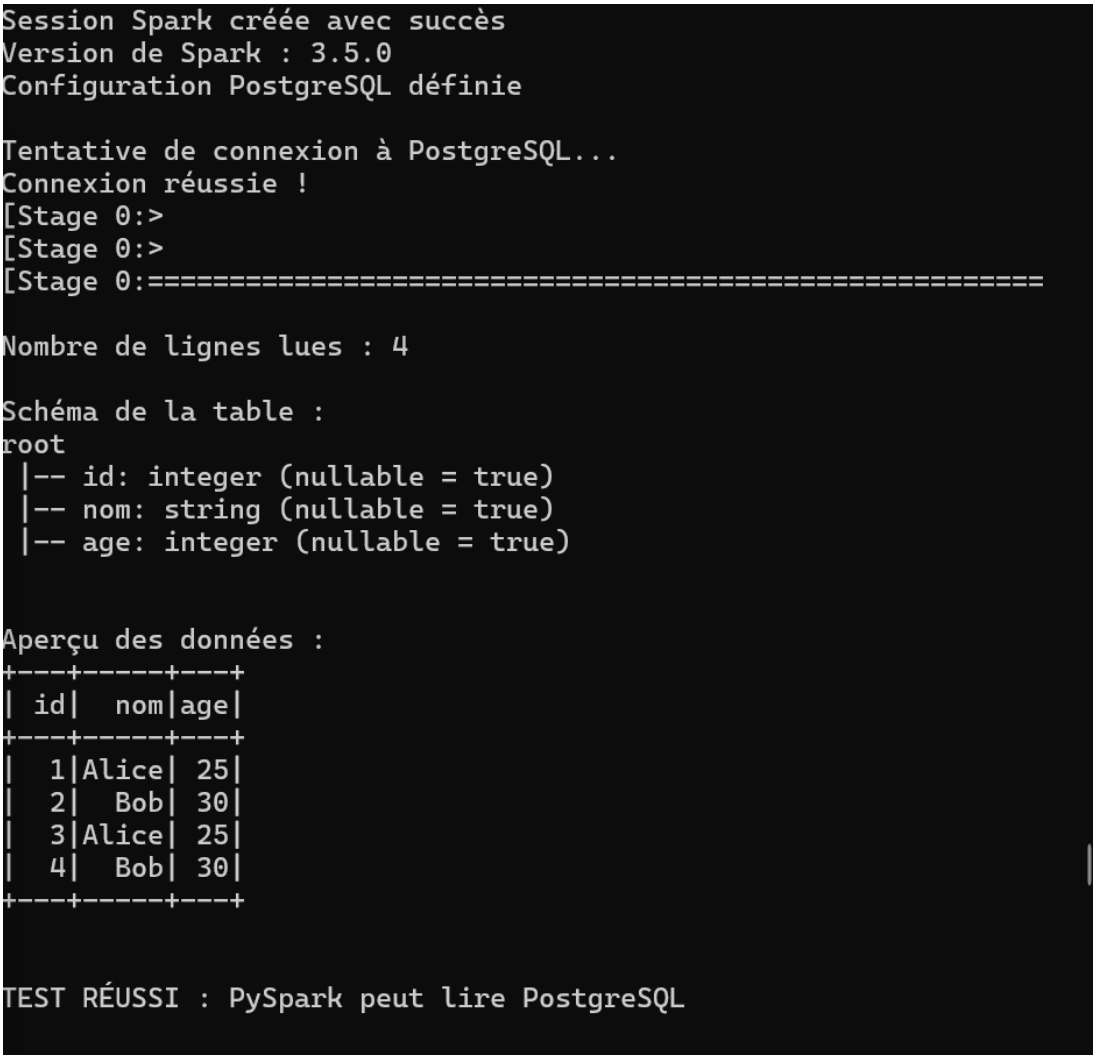


FIGURE 4.1 – Connexion PySpark - PostgreSQL réussie

Chapitre 5

Slowly Changing Dimensions (SCD Type 2)

5.1 Comprendre les SCD

Une **Slowly Changing Dimension (SCD)** est une dimension dont les données changent lentement dans le temps.

Exemple : Un client nommé *Jean Dupont* habite à Paris :

client_id	nom	ville
1	Jean Dupont	Paris

Si Jean déménage à Lyon, nous avons deux options :
— **SCD Type 1** : Remplacer Paris par Lyon → Problème : l'historique est perdu.
— **SCD Type 2** : Créer une nouvelle ligne → Solution : l'historique est conservé.
Résultat avec SCD Type 2 :

client_key	client_id	nom	ville	date_debut	date_fin	est_courant
1	1	Jean Dupont	Paris	2023-01-01	2024-12-31	FALSE
2	1	Jean Dupont	Lyon	2025-01-01	NULL	TRUE

Ainsi, il est possible de savoir que Jean habitait à Paris jusqu'au 31/12/2024 et à Lyon depuis le 01/01/2025.

5.2 Créer la base de données et les tables sources

5.2.1 Création de la base de données

Le script SQL a été exécuté dans PostgreSQL pour créer la base `retailpro_dwh`.

5.2.2 Création des tables sources

Les tables sources (`clients_source`, `produits_source`, `ventes_source`) ont été créées et peuplées avec des données test.

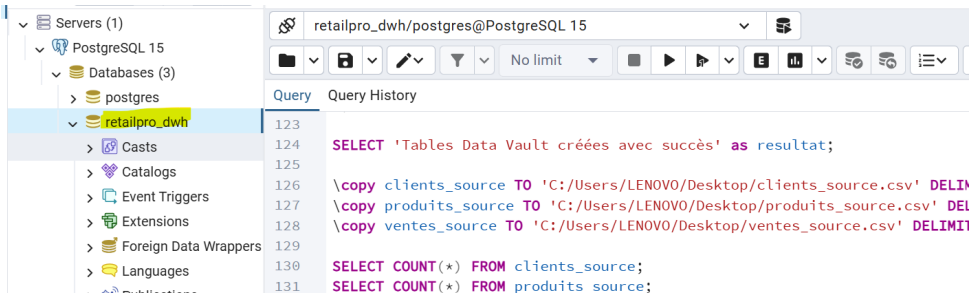
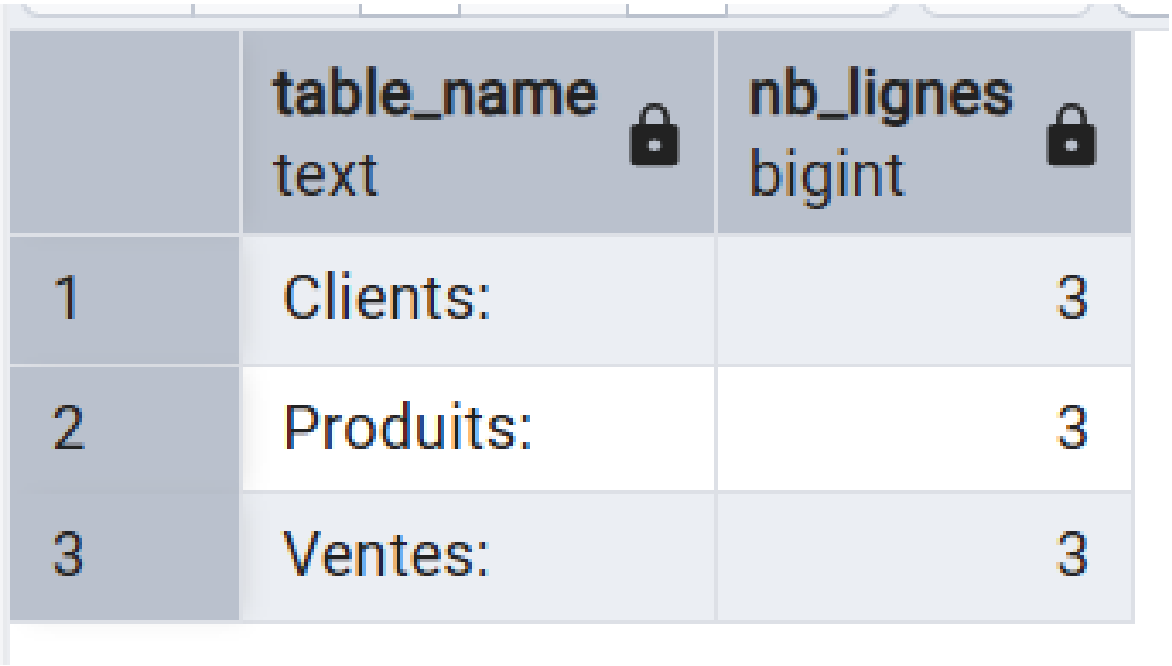


FIGURE 5.1 – Vérification de la création de la base de données dans pgAdmin



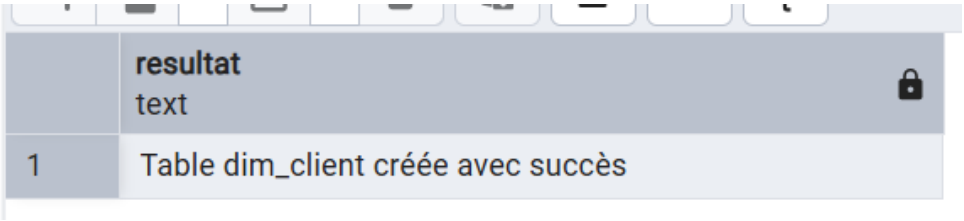
	table_name text	nb_lignes bigint
1	Clients:	3
2	Produits:	3
3	Ventes:	3

FIGURE 5.2 – Vérification des tables sources dans pgAdmin

5.3 Créer la table dimension avec SCD Type 2

Cette étape consiste à créer la **dimension client avec SCD Type 2**, permettant de conserver l'historique complet des modifications des clients.

Le script SQL a été exécuté dans PostgreSQL pour créer la table `dim_client`.



	resultat text
1	Table dim_client créée avec succès

FIGURE 5.3 – Vérification de la création de la table dim_client avec SCD Type 2

5.4 Script Python de chargement SCD Type 2

Cette étape consiste à exécuter le **script Python de chargement SCD Type 2**, permettant de charger les clients depuis la source vers la table `dim_client` en conservant l'historique complet.

Le script `03_load_scd_type2.py` a été exécuté dans Python. Chaque étape est expliquée ligne par ligne dans le script pour comprendre le fonctionnement :

- Import des bibliothèques et configuration de la connexion PostgreSQL
- Lecture des clients depuis la table source `clients_source`
- Vérification de l'existence d'un client dans la dimension
- Détection des changements (email, ville, segment)
- Insertion de nouveaux clients ou création de nouvelles versions
- Affichage des statistiques du chargement

Exécution et vérification : Après avoir exécuté le script avec la commande :

```
python 03_load_scd_type2.py
```

Listing 5.1 – Exécution du script SCD Type 2

Le résultat attendu est similaire à celui-ci :

```
(venv) C:\Users\LENOVO\Desktop\TP_DataWarehouse>python 03_load_s
cd_type_2.py
Bibliothèques importées
Configuration : connexion à retailpro_dwh sur localhost

======================================================================
DÉBUT DU PROCESSUS SCD TYPE 2
======================================================================
✓ Connexion à PostgreSQL réussie
✓ 3 clients lus depuis la source

Traitement du client 1
✓ Nouveau client inséré : client_id=1

Traitement du client 2
✓ Nouveau client inséré : client_id=2

Traitement du client 3
✓ Nouveau client inséré : client_id=3

======================================================================
RÉSUMÉ DU CHARGEMENT
======================================================================
Nouveaux clients : 3
Nouvelles versions : 0
Inchangés : 0
======================================================================
```

FIGURE 5.4 – Vérification du chargement SCD Type 2 dans la table dim_client

On peut observer que :
— Les clients ont été lus depuis la source
— Les nouveaux clients ont été insérés avec la version 1
— Les statistiques de traitement confirment le nombre de clients chargés et de nouvelles versions créées

Cette étape permet de s'assurer que le processus SCD Type 2 fonctionne correctement et que l'historique des clients est bien conservé.

5.5 Tester le SCD Type 2 avec des changements

Cette étape consiste à **simuler des changements dans la table source** et à **relancer le script de chargement SCD Type 2** pour vérifier que le processus gère correctement les nouvelles versions.

5.5.1 Modification des données source

Dans pgAdmin ou psql, les modifications suivantes ont été effectuées :
— Changement de la ville du client 1 : Paris → Lyon
— Changement du segment du client 2 : Silver → Platinum

5.5.2 Relancer le script de chargement SCD Type 2

Le script Python `03_load_scd_type2.py` a été relancé pour détecter les changements et créer de nouvelles versions dans la dimension.

```
python 03_load_scd_type2.py
```

Listing 5.2 – Exécution du script après modifications

5.5.3 Résultat attendu et vérification

Le résultat attendu montre que :
— La ville du client 1 a été modifiée → une nouvelle version a été créée
— Le segment du client 2 a été modifié → une nouvelle version a été créée
— Le client 3 est inchangé → aucune nouvelle version

Statistiques de chargement :
— Nouveaux clients insérés : 0
— Nouvelles versions créées : 2
— Clients inchangés : 1
— Total traité : 3

Cette étape permet de confirmer que le **SCD Type 2 fonctionne correctement** : les changements dans la source sont historisés avec de nouvelles versions, et l'historique des clients est conservé.

5.5.4 Vérification de l'historique dans la dimension

Pour s'assurer que le SCD Type 2 conserve bien l'historique complet des clients, nous interrogeons la table `dim_client` pour le client 1 :

Résultat attendu :

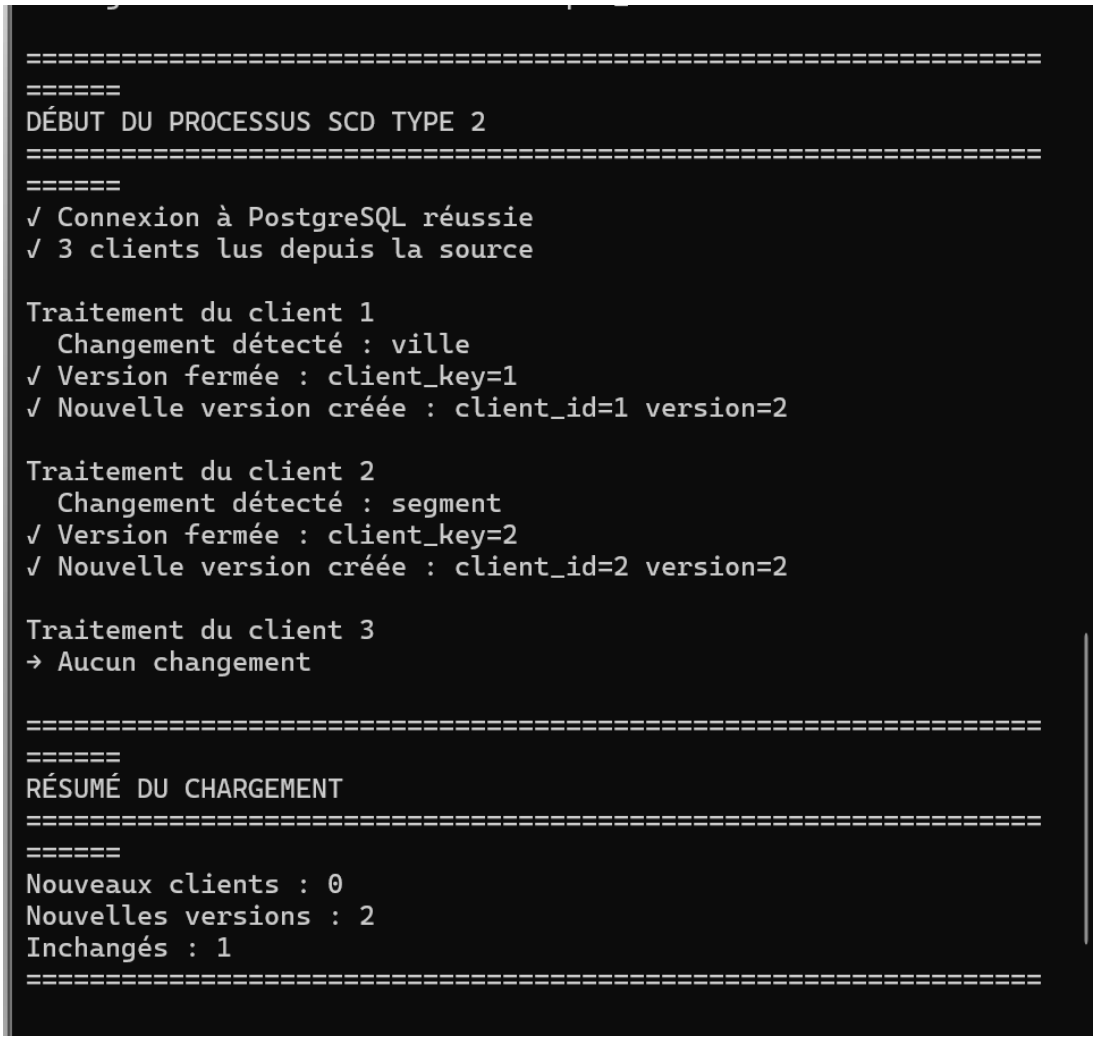


FIGURE 5.5 – Vérification des nouvelles versions créées après modifications des clients dans la source

Cette vérification confirme que :

— La première version du client (Paris) est fermée (est_courant = FALSE).
— La nouvelle version (Lyon) est active (est_courant = TRUE).
— L'historique complet est donc conservé, conforme au fonctionnement du SCD Type 2.

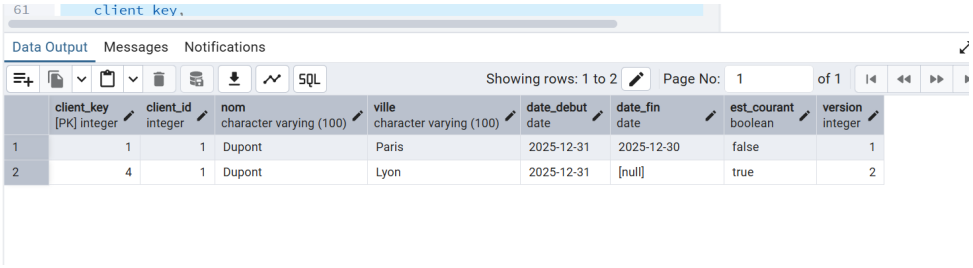


FIGURE 5.6 – Vérification de l'historique complet pour le client 1

Chapitre 6

Data Vault 2.0 - Modélisation Agile

6.1 Comprendre Data Vault

Qu'est-ce que Data Vault ?

Data Vault est une méthode de modélisation qui sépare les données en 3 types de tables :

— **HUBS** : Contiennent les identifiants uniques des entités (clients, produits, commandes)
— **LINKS** : Représentent les relations entre les hubs (ex. vente = client + produit)
— **SATELLITES** : Contiennent tous les attributs descriptifs avec historique

Analogie simple :

— **Hub** = Une carte d'identité (juste l'ID)
— **Link** = Un certificat de mariage (relie 2 personnes)
— **Satellite** = Votre CV complet avec historique

6.1.1 Architecture Data Vault

Explication du schéma :

— Chaque **Hub** contient une clé unique et identifie l'entité.
— Chaque **Link** relie les hubs pour représenter les relations entre entités.
— Chaque **Satellite** stocke les détails descriptifs et l'historique des changements pour les hubs et links.

6.1.2 Vérification finale des tables Data Vault

Après l'exécution du script `04_create_data_vault.sql`, il est important de vérifier que toutes les tables Data Vault ont été créées correctement dans PostgreSQL.

Liste des tables attendues :

— **Hubs :** `hub_client`, `hub_produit`
— **Link :** `link_vente`
— **Satellites :** `sat_client`, `sat_produit`, `sat_vente`

Vérification visuelle :

Conclusion : Toutes les tables Data Vault sont présentes et prêtes pour le chargement et l'historisation des données. La structure est conforme à la méthodologie Data Vault 2.0.

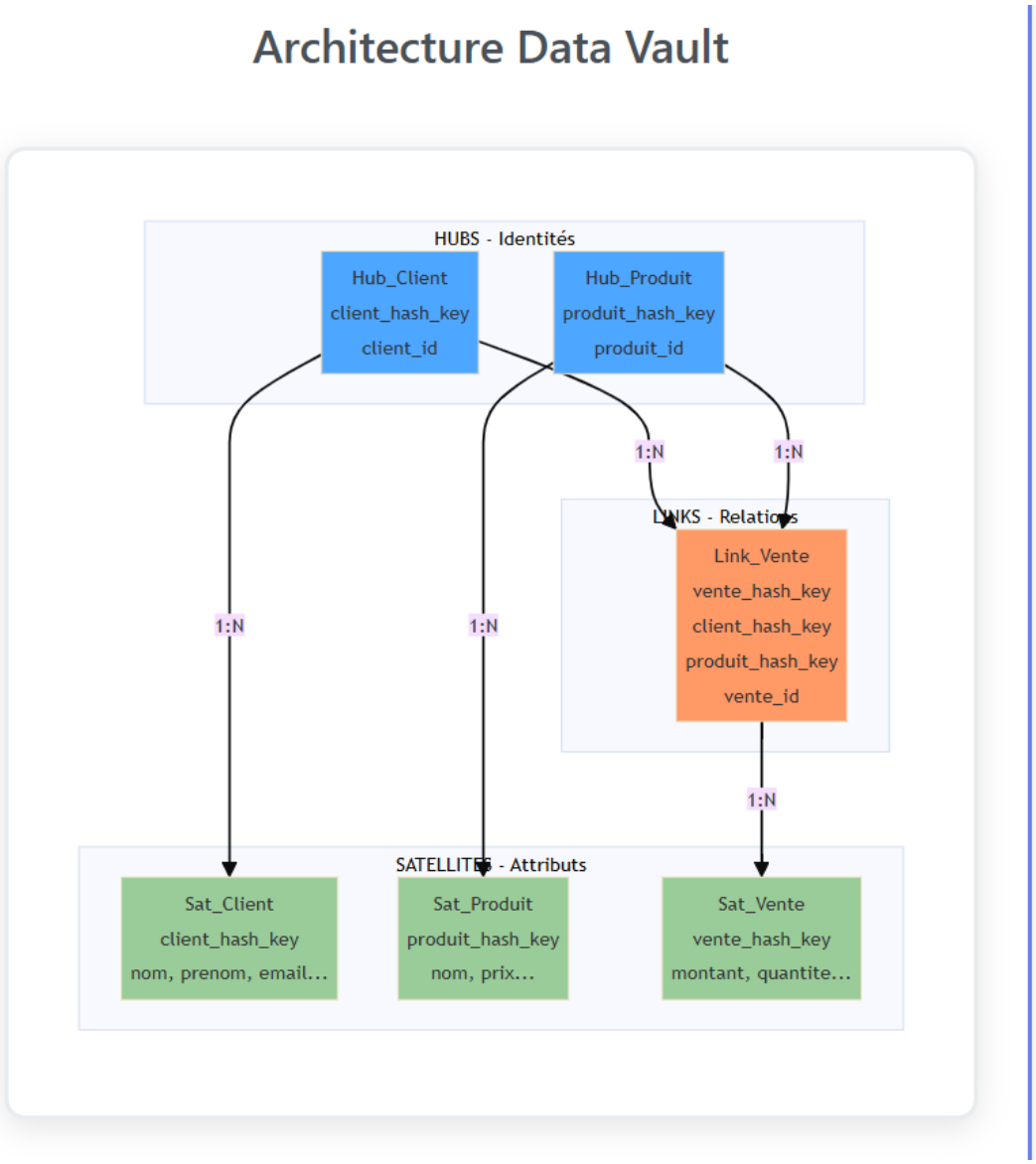


FIGURE 6.1 – Architecture simplifiée Data Vault : Hubs, Links et Satellites

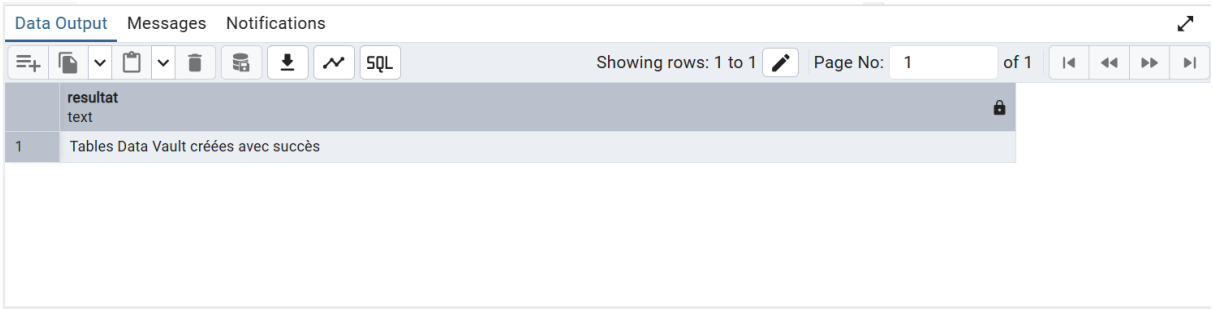


FIGURE 6.2 – Vérification que toutes les tables Data Vault (Hubs, Link, Satellites) existent dans pgAdmin

Chapitre 7

Architecture Lakehouse avec PySpark et Delta Lake

7.1 Comprendre l'Architecture Lakehouse

Le Lakehouse combine les avantages des Data Lakes et des Data Warehouses pour gérer, nettoyer et analyser les données efficacement.

7.1.1 Les 3 couches du Lakehouse

Couche	Rôle	Exemple
BRONZE	Stocker les données BRUTES telles qu'elles arrivent	Copie exacte de PostgreSQL, san
SILVER	NETTOYER et VALIDER les données	Emails validés, doublons supprimés
GOLD	AGRÉGER pour l'analyse	Ventes par jour, Top produits,

Analogie culinaire : Bronze = Ingrédients bruts du supermarché Silver = Légumes lavés, épluchés, découpés Gold = Plat préparé prêt à servir

7.2 Configuration de l'environnement Lakehouse

7.2.1 Création des répertoires

Windows :

```
mkdir C:\lakehouse
mkdir C:\lakehouse\bronze
mkdir C:\lakehouse\silver
mkdir C:\lakehouse\gold
```

macOS/Linux :

```
mkdir -p ~/lakehouse/bronze
mkdir -p ~/lakehouse/silver
mkdir -p ~/lakehouse/gold
```

Explication : Delta Lake stocke les tables sous forme de fichiers (Parquet + logs de transaction). Ces dossiers correspondent aux couches Bronze, Silver et Gold.

7.3 BRONZE : Ingestion depuis PostgreSQL

7.3.1 Script d'ingestion Bronze (05_bronze_ingestion.py)

Le script PySpark lit les tables PostgreSQL et les écrit en Delta Lake avec 3 colonnes de métadonnées : `ingestion_timestamp`, `source_system`, `source_table`.

- Import des bibliothèques et configuration Delta Lake
- Création de la session Spark
- Lecture des tables PostgreSQL (clients, produits, ventes)
- Ajout des métadonnées
- Écriture en Delta Lake dans le dossier Bronze

7.3.2 Exécution

```
python 05_bronze_ingestion.py
```

7.3.3 Vérification

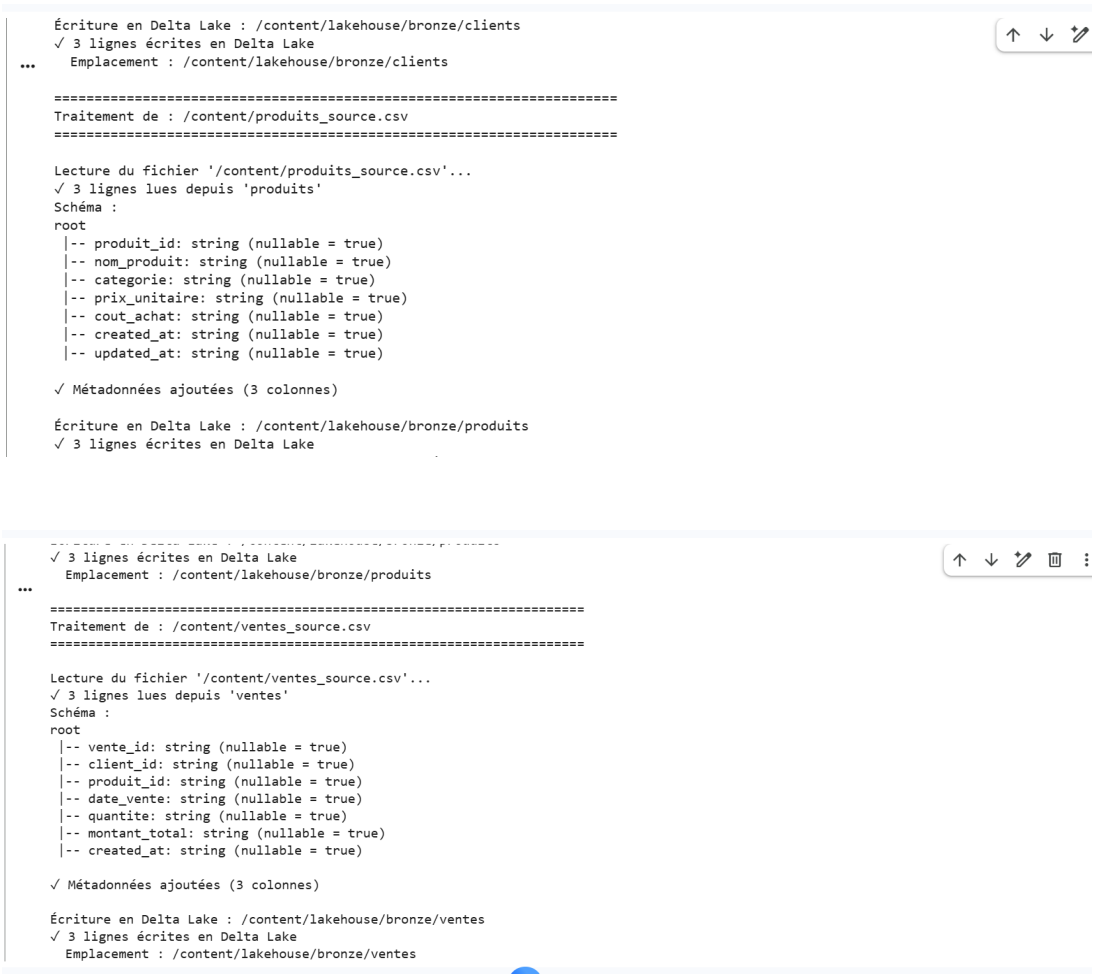


FIGURE 7.1 – Vérification des données Bronze dans Delta Lake (Windows / Linux)

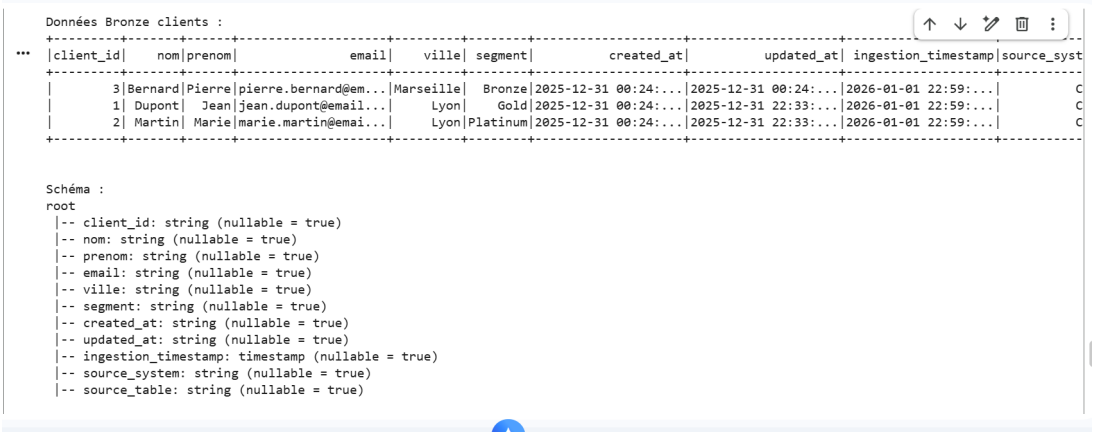
Conclusion : Toutes les tables PostgreSQL ont été ingérées correctement dans la couche Bronze, avec ajout des métadonnées et écriture en format Delta Lake. La couche Bronze est prête pour le nettoyage et l'enrichissement en Silver.

7.4 Vérification des données Bronze

Après l'ingestion des données dans la couche Bronze, il est important de vérifier que les tables ont bien été créées et que les données sont intactes.

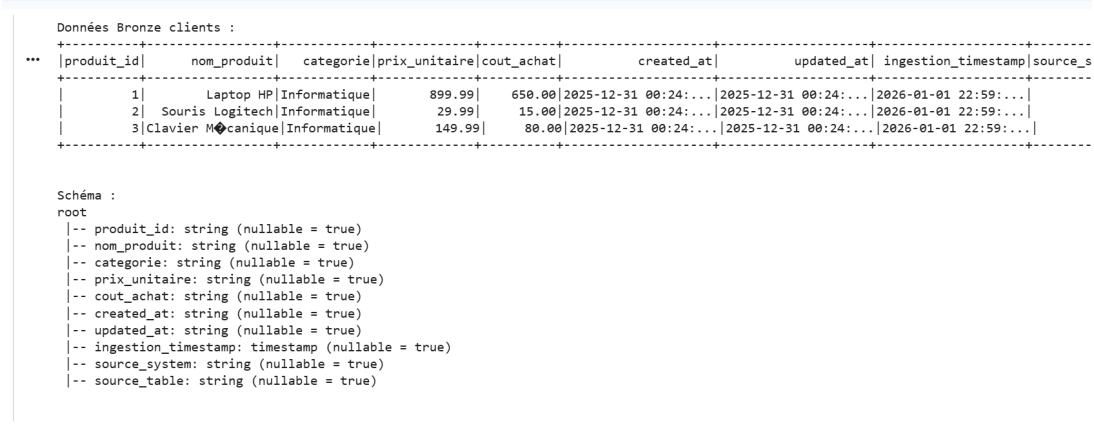
7.4.1 Script de vérification Bronze (06_verify_bronze.py)

7.4.2 Résultat attendu



```
Données Bronze clients :
+---------+------+------+--------------------+---------+-------+-------------------+-------------------+-------------------+-----------
|client_id|   nom|prenom|               email|    ville|segment|         created_at|         updated_at|ingestion_timestamp|source_syst
+---------+------+------+--------------------+---------+-------+-------------------+-------------------+-------------------+-----------
|        3|Bernard|Pierre|pierre.bernard@em...|Marseille| Bronze|2025-12-31 00:24:...|2025-12-31 00:24:...|2026-01-01 22:59:...|          C
|        1| Dupont|  Jean|jean.dupont@email...|     Lyon|   Gold|2025-12-31 00:24:...|2025-12-31 22:33:...|2026-01-01 22:59:...|          C
|        2| Martin| Marie|marie.martin@emai...|     Lyon|Platinum|2025-12-31 00:24:...|2025-12-31 22:33:...|2026-01-01 22:59:...|          C
+---------+------+------+--------------------+---------+-------+-------------------+-------------------+-------------------+-----------


Schéma :
root
 |-- client_id: string (nullable = true)
 |-- nom: string (nullable = true)
 |-- prenom: string (nullable = true)
 |-- email: string (nullable = true)
 |-- ville: string (nullable = true)
 |-- segment: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- updated_at: string (nullable = true)
 |-- ingestion_timestamp: timestamp (nullable = true)
 |-- source_system: string (nullable = true)
 |-- source_table: string (nullable = true)
```



```
Données Bronze clients :
+----------+-----------------+------------+-------------+----------+-------------------+-------------------+-------------------+--------
|produit_id|      nom_produit|   categorie|prix_unitaire|cout_achat|         created_at|         updated_at|ingestion_timestamp|source_s
+----------+-----------------+------------+-------------+----------+-------------------+-------------------+-------------------+--------
|         1|        Laptop HP|Informatique|       899.99|    650.00|2025-12-31 00:24:...|2025-12-31 00:24:...|2026-01-01 22:59:...|
|         2|  Souris Logitech|Informatique|        29.99|     15.00|2025-12-31 00:24:...|2025-12-31 00:24:...|2026-01-01 22:59:...|
|         3|Clavier M?canique|Informatique|       149.99|     80.00|2025-12-31 00:24:...|2025-12-31 00:24:...|2026-01-01 22:59:...|
+----------+-----------------+------------+-------------+----------+-------------------+-------------------+-------------------+--------


Schéma :
root
 |-- produit_id: string (nullable = true)
 |-- nom_produit: string (nullable = true)
 |-- categorie: string (nullable = true)
 |-- prix_unitaire: string (nullable = true)
 |-- cout_achat: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- updated_at: string (nullable = true)
 |-- ingestion_timestamp: timestamp (nullable = true)
 |-- source_system: string (nullable = true)
 |-- source_table: string (nullable = true)
```

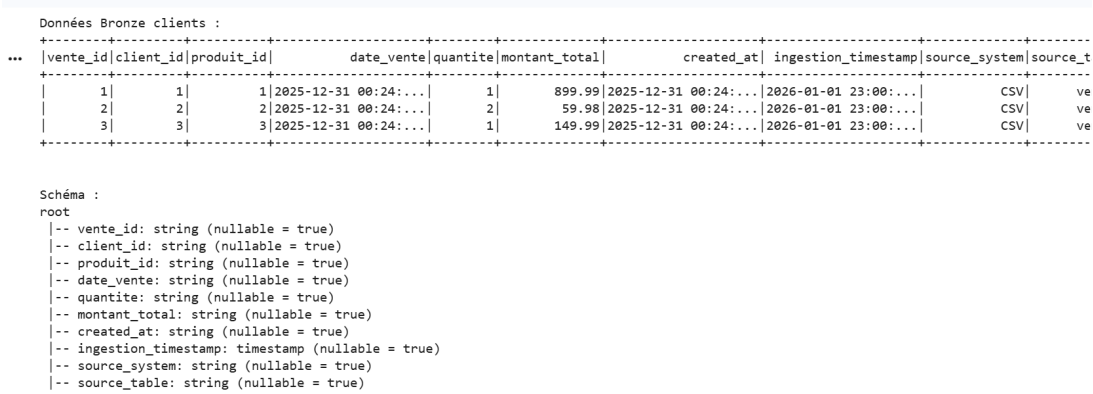


FIGURE 7.2 – Vérification des données Bronze dans Delta Lake

Chapitre 8

Projet Final - Intégration Complète

8.1 Objectif du Projet

L'objectif de ce projet est de créer un pipeline de données complet qui :
— Lit les données depuis PostgreSQL
— Les ingère dans la couche **Bronze** (Delta Lake)
— Les nettoie dans la couche **Silver**
— Crée des agrégations dans la couche **Gold**
— Génère un rapport avec les insights clés

8.2 Pipeline Complet

8.2.1 Transformation Bronze → Silver

Script : 07_silver_transformation.py

```
# Code complet comme dans la section pr c dente
```

Schéma Bronze → Silver

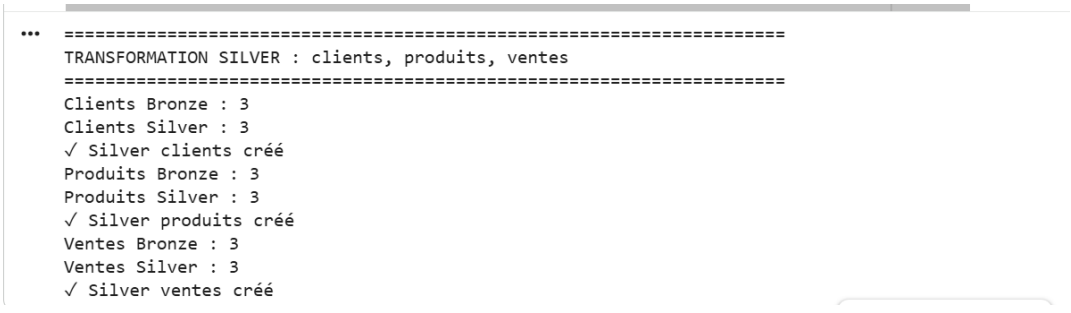


FIGURE 8.1 – Pipeline Bronze → Silver : lecture, nettoyage, ajout de métadonnées, écriture en Silver

Exécution :

```
python 07_silver_transformation.py
```

Résultat attendu :

— Suppression des doublons
— Noms standardisés et emails validés
— Table Silver `clients` créée

8.2.2 Agrégation Silver → Gold

Script : 08_gold$_{a}ggregation.py$

```
# Code complet comme dans la section pr c dente
```

Schéma Silver → Gold

```
Contenu Gold ventes_quotidiennes :
+----------+---------+--------+-----------------+
|      date|nb_ventes|ca_total|     panier_moyen|
+----------+---------+--------+-----------------+
|2025-12-31|        3| 1109.96|369.9866666666667|
+----------+---------+--------+-----------------+


Schéma :
root
 |-- date: date (nullable = true)
 |-- nb_ventes: long (nullable = true)
 |-- ca_total: double (nullable = true)
 |-- panier_moyen: double (nullable = true)


Nombre de lignes : 1
```

FIGURE 8.2 – Pipeline Silver → Gold : agrégation des ventes, calcul des KPIs et écriture en Gold

Exécution :

```
python 08_gold_aggregation.py
```

Résultat attendu :
— **Table Gold `ventes_quotidiennes` créée**
— **Agrégations : nombre de ventes, chiffre d'affaires, panier moyen**

8.2.3 Génération du Rapport Final

Script : 09_generer_rapport.py

```
# Code complet comme dans la section pr c dente
```

Schéma Gold → Rapport Final

Exécution :

```
python 09_generer_rapport.py
```

Résultat attendu :
— **Statistiques globales : total des ventes, chiffre d'affaires, panier moyen**
— **Top 5 des meilleurs jours par chiffre d'affaires**
— **Rapport final généré avec succès**

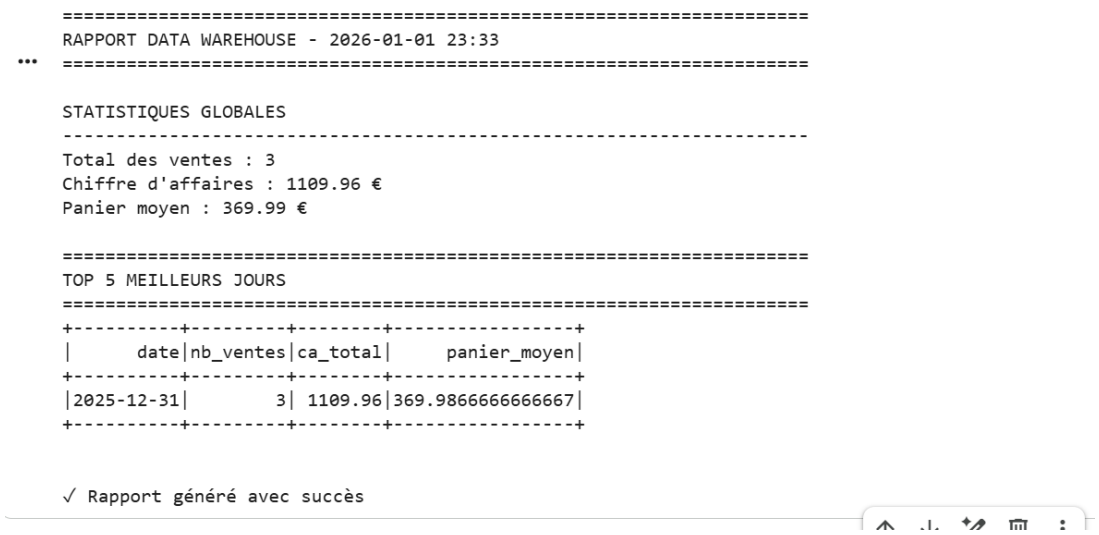


FIGURE 8.3 – Pipeline Gold → Rapport Final : lecture des données Gold et génération des statistiques et insights clés

Chapitre 9

Conclusion

Ce TP a permis de comprendre et d'expérimenter concrètement les bonnes pratiques d'ingénierie des données, notamment :

- **L'importance de conserver l'historique des données avec les SCD Type 2 et Data Vault**
- **La gestion des métadonnées et du versioning avec Delta Lake**
- **La transformation et l'agrégation des données pour produire des insights exploitables**

Ainsi, ce projet constitue une base solide pour concevoir des pipelines de données industriels, fiables, évolutifs et faciles à maintenir.