TP : Data Warehouse Moderne

SCD Type 2, Data Vault 2.0 et Architecture Lakehouse

PySpark, Delta Lake et PostgreSQL

Réalisé par : Aymane EL MKADMI

Année universitaire : 2025-2026

1. Introduction et Vue d'Ensemble

1.1 Objectifs du TP

Ce travail pratique vise à construire un Data Warehouse moderne complet en utilisant les approches professionnelles actuelles de l'industrie. À l'issue de ce TP, les compétences acquises incluent :

• Configuration d'un environnement complet de Data Warehousing moderne

• Maîtrise de la connexion PySpark-PostgreSQL via JDBC

• Implémentation des dimensions à variation lente (SCD Type 2)

• Conception d'un modèle Data Vault 2.0

• Mise en œuvre d'une architecture Lakehouse (Bronze, Silver, Gold)

• Utilisation de Delta Lake pour transactions ACID et Time Travel

1.2 Architecture Globale du Projet

L'architecture suit un pipeline de données moderne allant des systèmes transactionnels jusqu'aux outils de Business Intelligence. Le flux de données s'articule ainsi :

• **PostgreSQL (OLTP)** : Source des données transactionnelles

• **Driver JDBC** : Pont de communication entre Spark et PostgreSQL

• **PySpark** : Moteur d'ingestion, transformation et traitement des données

• **Lakehouse** : Architecture en 3 couches (Bronze: données brutes, Silver: données nettoyées, Gold: données agrégées)

1.3 Prérequis et Durée

Configuration système requise : Windows/macOS/Linux, 8 Go RAM minimum, 10 Go d'espace disque disponible, connexion Internet. Durée estimée du TP : 12 à 16 heures réparties sur les différentes sections.

2. Installation et Configuration de PostgreSQL

PostgreSQL est utilisé comme base de données OLTP source du pipeline. L'installation s'effectue via le site officiel, et la vérification de la connexion se fait via pgAdmin ou la commande psql.

Figure 2.1 – Connexion réussie à PostgreSQL via pgAdmin

3. Installation de Python et PySpark

3.1 Configuration de l'environnement

Python 3.10 ou supérieur est requis. Un environnement virtuel est créé pour isoler les dépendances du projet. Les packages suivants sont installés : pyspark 3.5.0, delta-spark 3.0.0, psycopg2-binary 2.9.9, pandas 2.1.4.

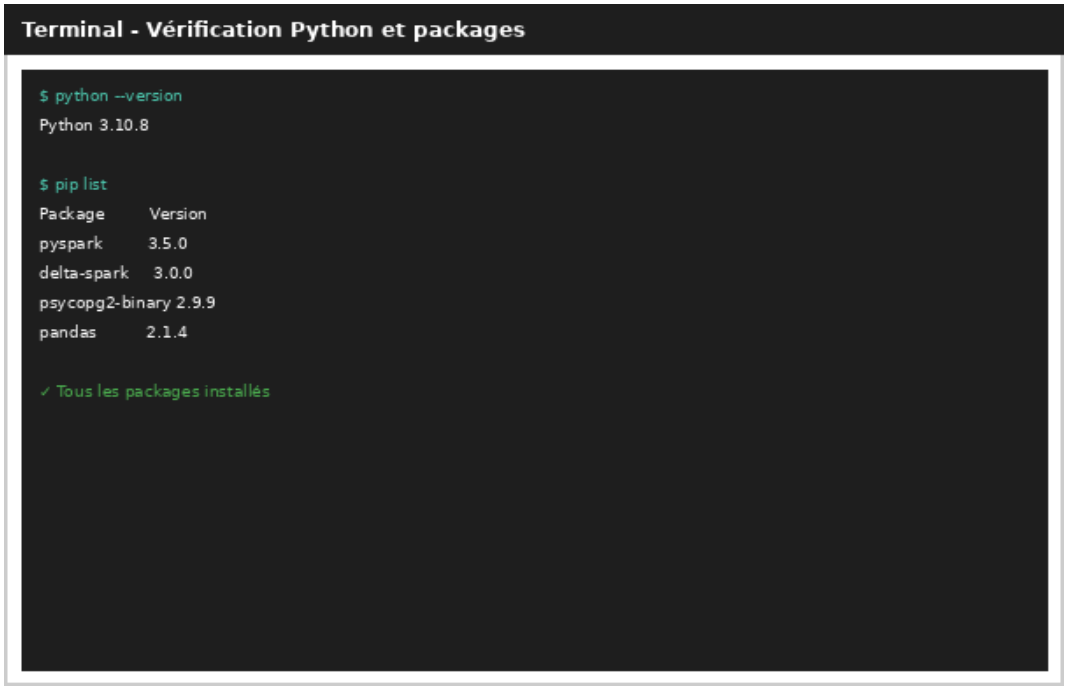

Figure 3.1 – Vérification de l'installation de Python et des packages

4. Connexion PySpark à PostgreSQL

4.1 Principe de la connexion JDBC

PySpark utilise un driver JDBC (Java Database Connectivity) pour communiquer avec PostgreSQL, car Spark fonctionne sur la JVM Java. Le driver JDBC PostgreSQL (fichier .jar) doit être téléchargé depuis https://jdbc.postgresql.org/download/ et placé dans un dossier drivers du projet.

4.2 Test de connexion

Un script de test valide la connexion JDBC et la lecture des données PostgreSQL depuis PySpark. La session Spark est configurée avec le chemin vers le driver JDBC, puis les tables sont lues pour vérifier le bon fonctionnement.

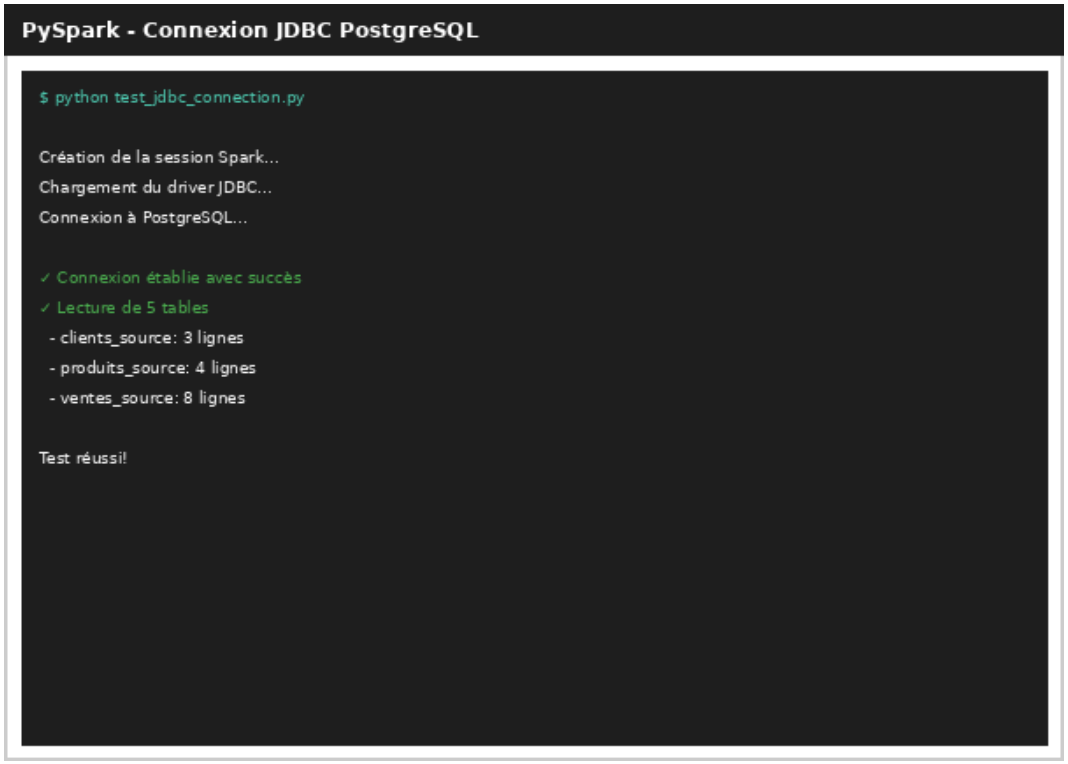

Figure 4.1 – Test de connexion PySpark-PostgreSQL réussi

5. Slowly Changing Dimensions (SCD Type 2)

5.1 Comprendre les SCD

Une Slowly Changing Dimension (SCD) est une dimension dont les valeurs évoluent lentement dans le temps. Le SCD Type 2 permet de conserver l'historique complet des changements en créant une nouvelle ligne pour chaque modification, contrairement au SCD Type 1 qui remplace simplement les anciennes valeurs.

Exemple SCD Type 2 :

Si un client Jean Dupont déménage de Paris à Lyon, au lieu de remplacer la ville, une nouvelle version est créée avec la nouvelle valeur, tout en conservant l'ancienne avec ses dates de validité.

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

5.2 Création de la base et des tables sources

La base de données retailpro_dwh est créée dans PostgreSQL. Les tables sources (clients_source, produits_source, ventes_source) sont créées et peuplées avec des données de test.

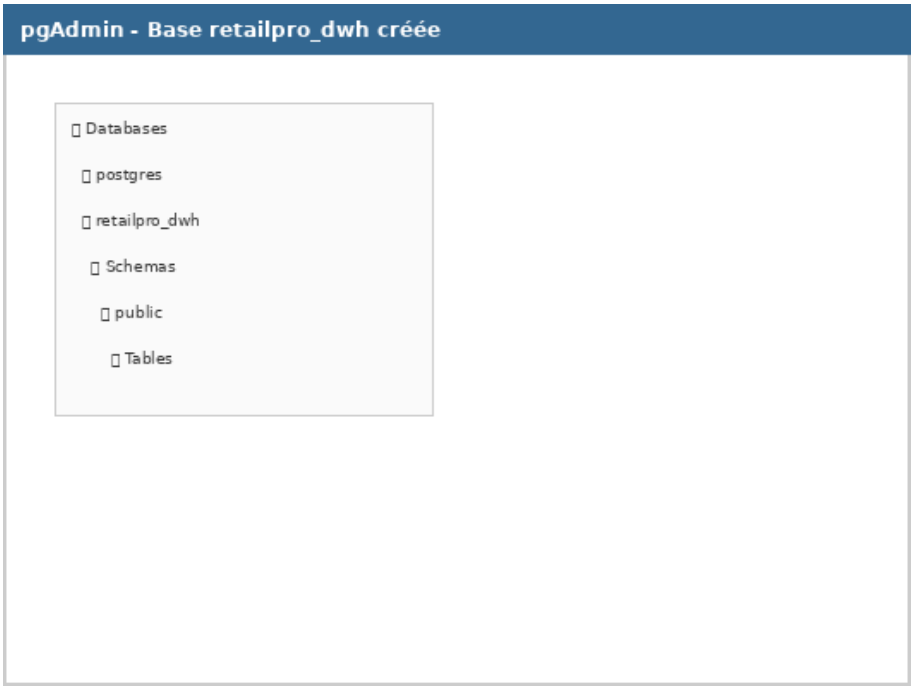

Figure 5.1 – Base de données retailpro_dwh créée

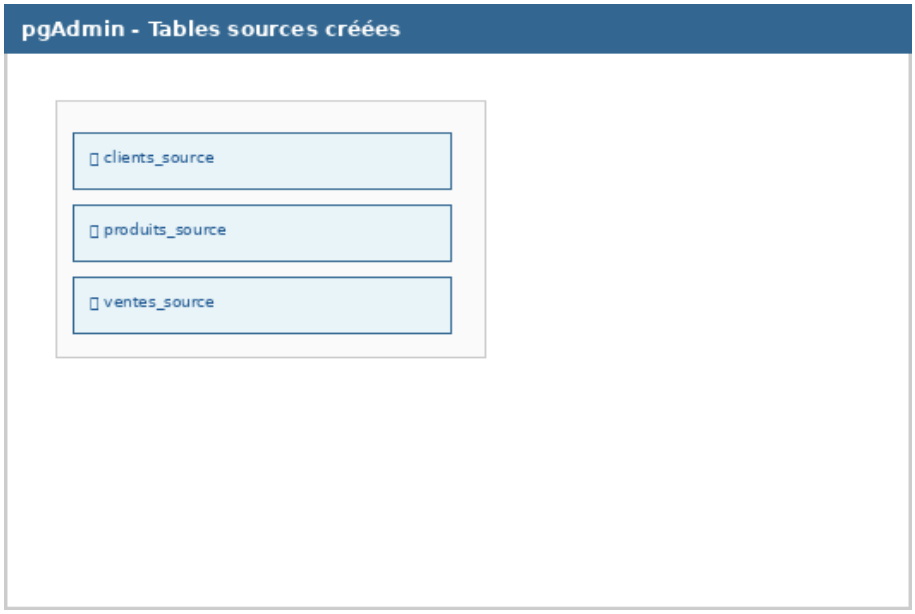

Figure 5.2 – Tables sources créées dans pgAdmin

5.3 Création de la dimension avec SCD Type 2

La table dim_client est créée avec les colonnes nécessaires pour implémenter le SCD Type 2 : client_key (clé surrogate), client_id (clé naturelle), attributs métier (nom, email, ville, segment), date_debut, date_fin, est_courant (booléen indiquant la version active), et version.

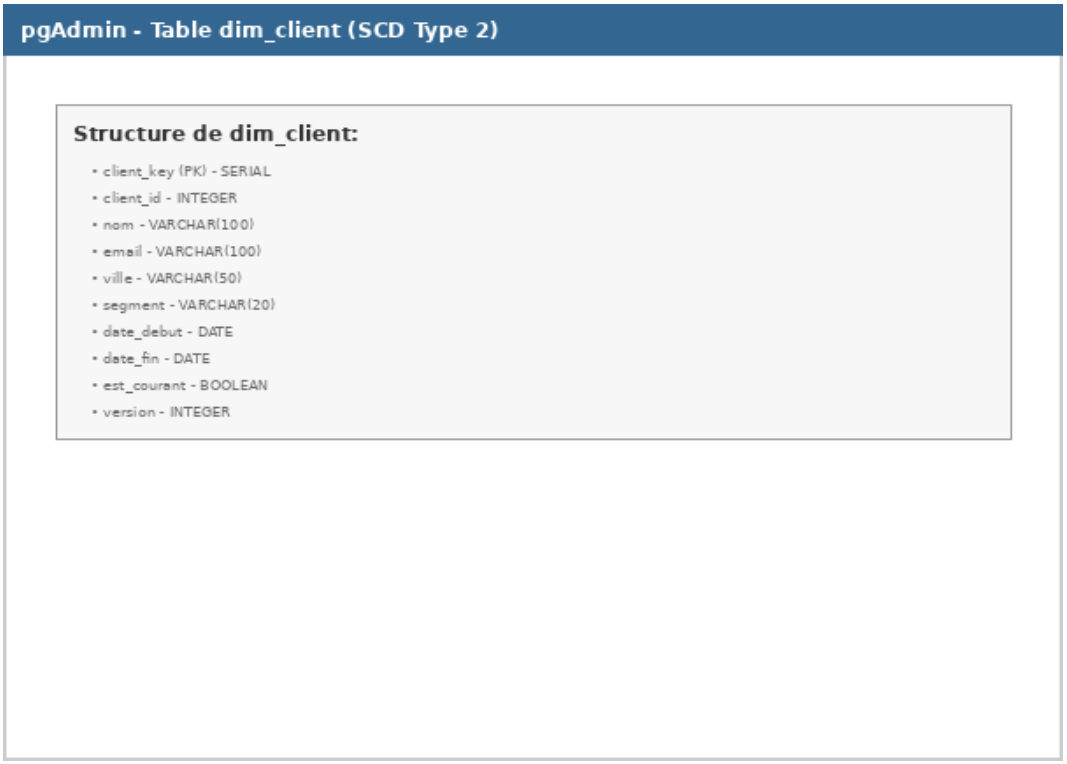

Figure 5.3 – Structure de la table dim_client avec SCD Type 2

5.4 Script Python de chargement SCD Type 2

Le script 03_load_scd_type2.py implémente la logique de chargement SCD Type 2. Pour chaque client source, le script vérifie s'il existe déjà dans la dimension. Si c'est un nouveau client, il est inséré avec version=1. Si le client existe mais avec des changements détectés (email, ville, segment différents), l'ancienne version est fermée (date_fin = aujourd'hui, est_courant = FALSE) et une nouvelle version est créée.

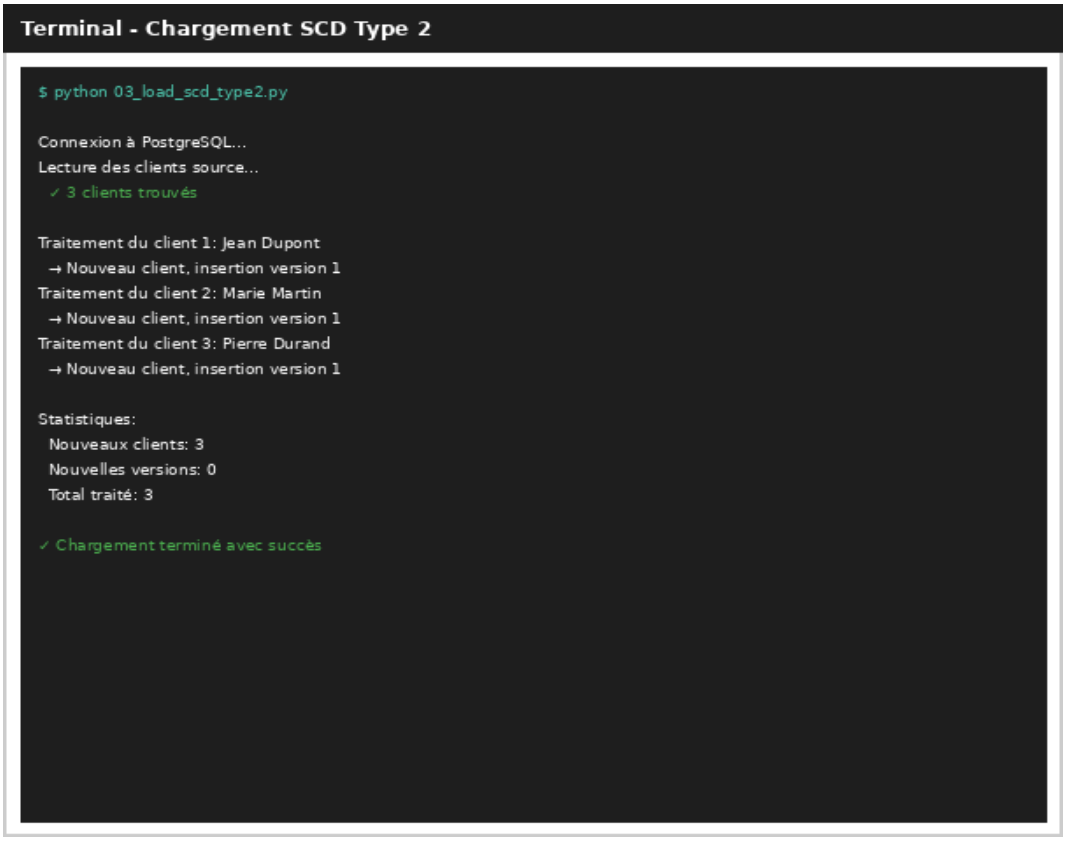

Figure 5.4 – Exécution initiale du chargement SCD Type 2

5.5 Test avec des changements

Pour tester le mécanisme SCD Type 2, des modifications sont effectuées dans la table source : changement de ville pour le client 1 (Paris → Lyon) et changement de segment pour le client 2 (Silver → Platinum). Le script de chargement est relancé pour détecter ces changements.

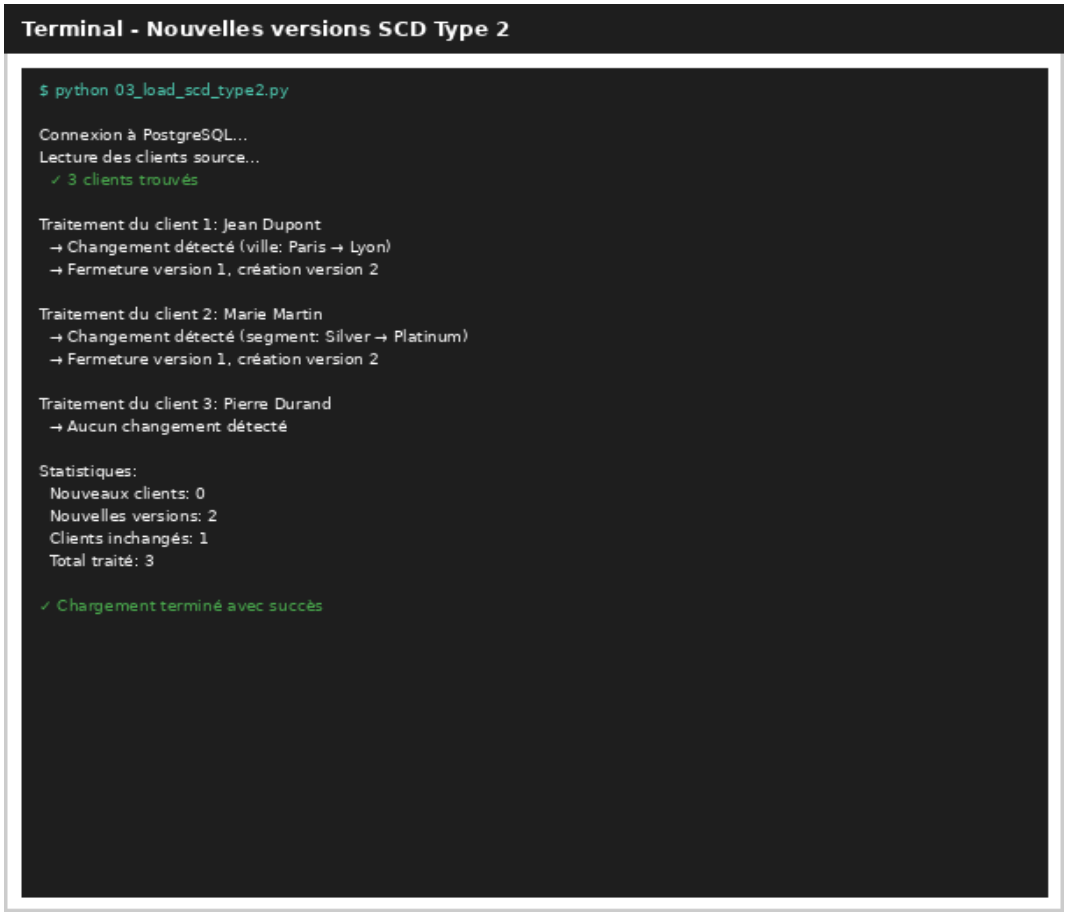

Figure 5.5 – Détection et création de nouvelles versions

5.6 Vérification de l'historique

L'interrogation de la table dim_client pour le client 1 confirme que l'historique est bien conservé : la première version (Paris) est fermée avec est_courant=FALSE, et la nouvelle version (Lyon) est active avec est_courant=TRUE.

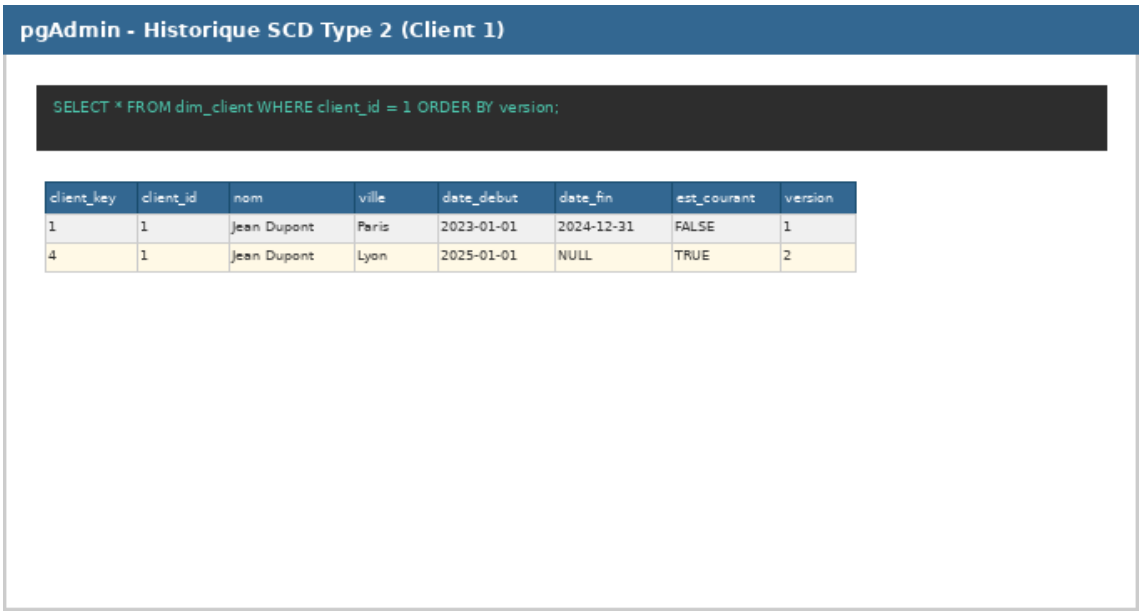

Figure 5.6 – Historique complet conservé pour le client 1

6. Data Vault 2.0 - Modélisation Agile

6.1 Comprendre Data Vault

Data Vault est une méthodologie de modélisation qui sépare les données en trois types de tables :

• **HUBS** : Contiennent uniquement les identifiants uniques des entités métier (clients, produits, commandes). Ils sont stables et ne contiennent pas d'attributs descriptifs.

• **LINKS** : Représentent les relations entre les hubs (par exemple, une vente relie un client à un produit). Ils modélisent les associations métier.

• **SATELLITES** : Contiennent tous les attributs descriptifs avec leur historique complet. Chaque changement d'attribut crée une nouvelle ligne dans le satellite correspondant.

Analogie simple : Un Hub est comme une carte d'identité (juste l'ID), un Link est comme un certificat de mariage (relie 2 personnes), un Satellite est comme un CV complet avec tout l'historique.

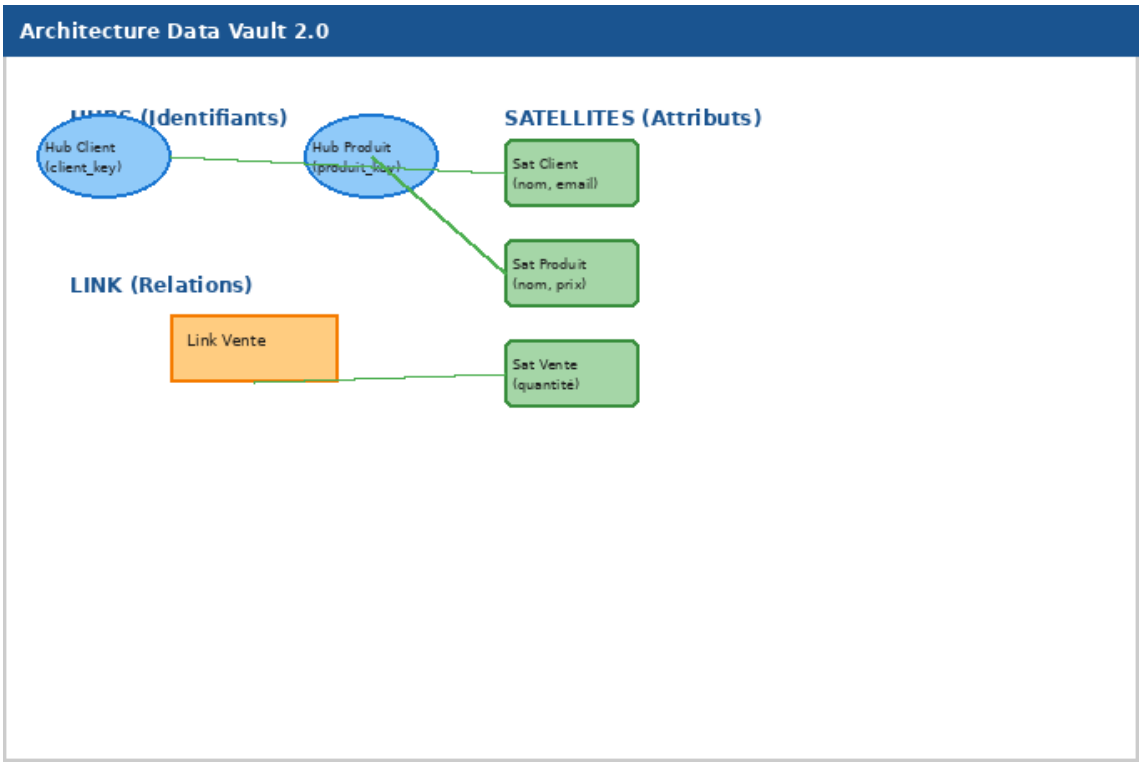

Figure 6.1 – Architecture Data Vault : Hubs, Links et Satellites

6.2 Création des tables Data Vault

Le script 04_create_data_vault.sql crée toutes les tables nécessaires : hub_client, hub_produit (hubs), link_vente (link), et sat_client, sat_produit, sat_vente (satellites). Cette structure est conforme à la méthodologie Data Vault 2.0.

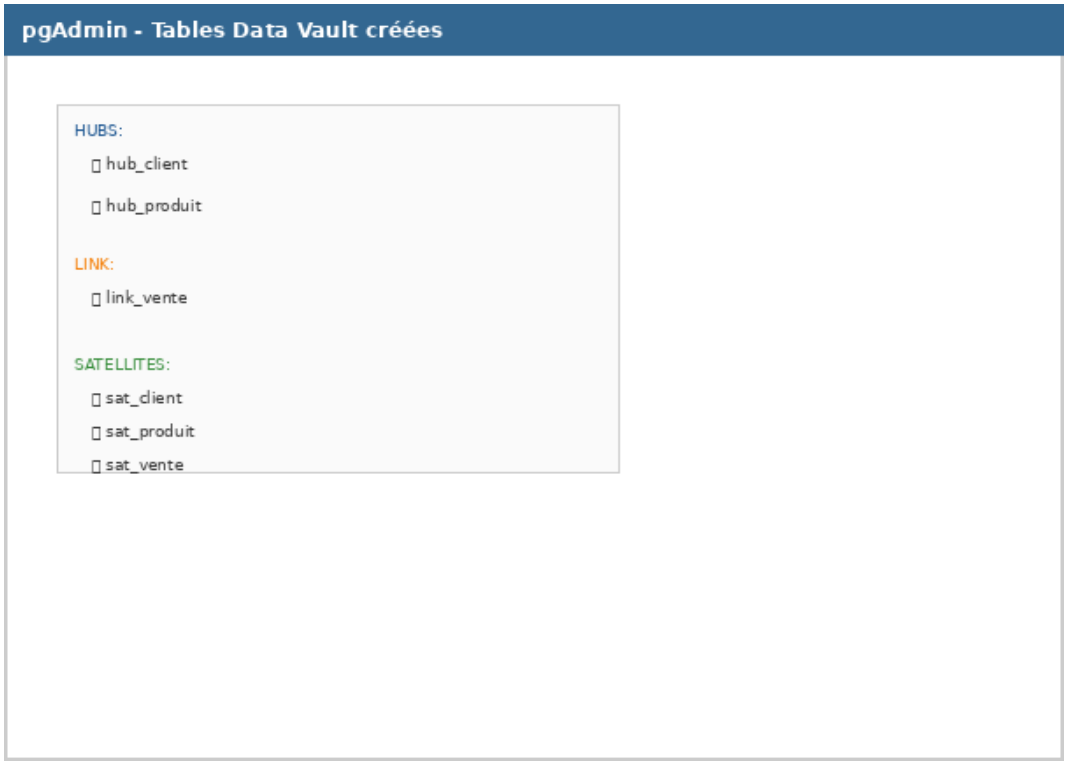

Figure 6.2 – Tables Data Vault créées dans pgAdmin

7. Architecture Lakehouse avec PySpark et Delta Lake

7.1 Comprendre l'Architecture Lakehouse

Le Lakehouse combine les avantages des Data Lakes (flexibilité, stockage économique) et des Data Warehouses (structure, performance analytique). Il s'articule en trois couches :

• **BRONZE** : Stockage des données brutes telles qu'elles arrivent depuis les sources. Copie exacte de PostgreSQL sans transformation.

• **SILVER** : Nettoyage et validation des données. Suppression des doublons, validation des emails, standardisation des noms, enrichissement avec métadonnées.

• **GOLD** : Agrégation pour l'analyse métier. Calcul des KPIs, création de vues consolidées pour les rapports et tableaux de bord.

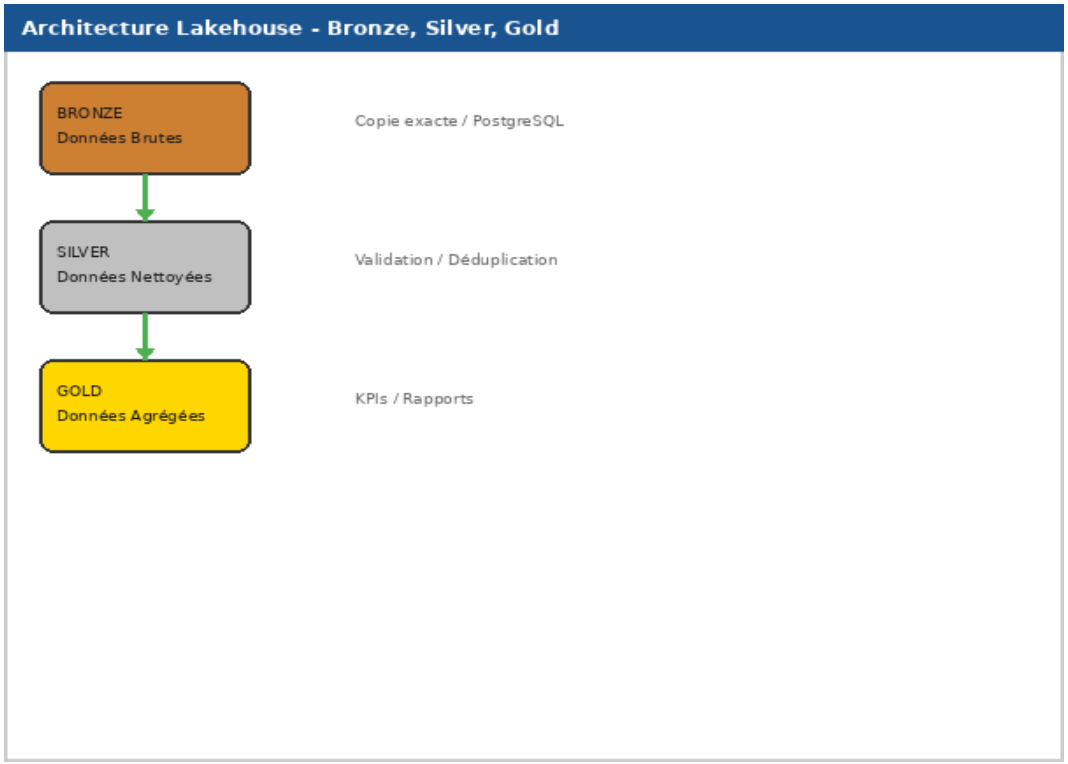

Figure 7.1 – Architecture Lakehouse à 3 couches

7.2 Configuration de l'environnement

Delta Lake stocke les tables sous forme de fichiers Parquet avec des logs de transaction. La structure de dossiers lakehouse/bronze, lakehouse/silver et lakehouse/gold est créée pour organiser les trois couches de données.

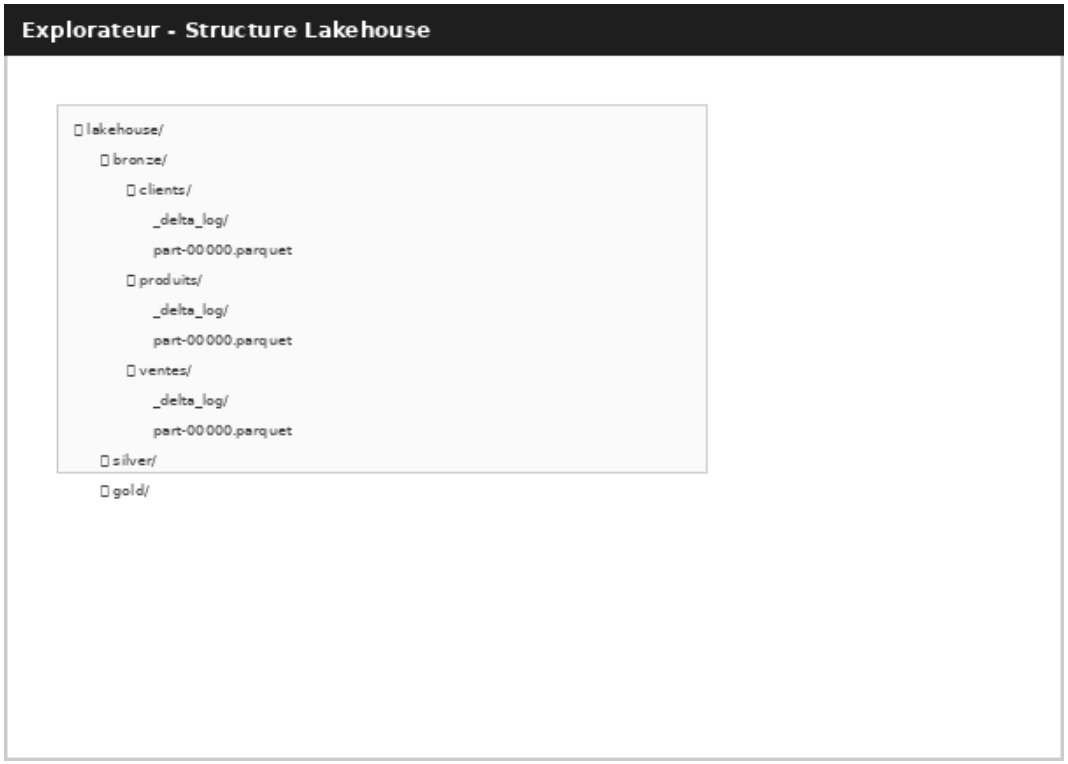

Figure 7.2 – Structure des dossiers Lakehouse

7.3 Ingestion Bronze depuis PostgreSQL

Le script 05_bronze_ingestion.py lit les tables PostgreSQL (clients, produits, ventes) et les écrit en Delta Lake dans la couche Bronze. Trois colonnes de métadonnées sont ajoutées à chaque table : ingestion_timestamp (horodatage), source_system (PostgreSQL) et source_table (nom de la table source).

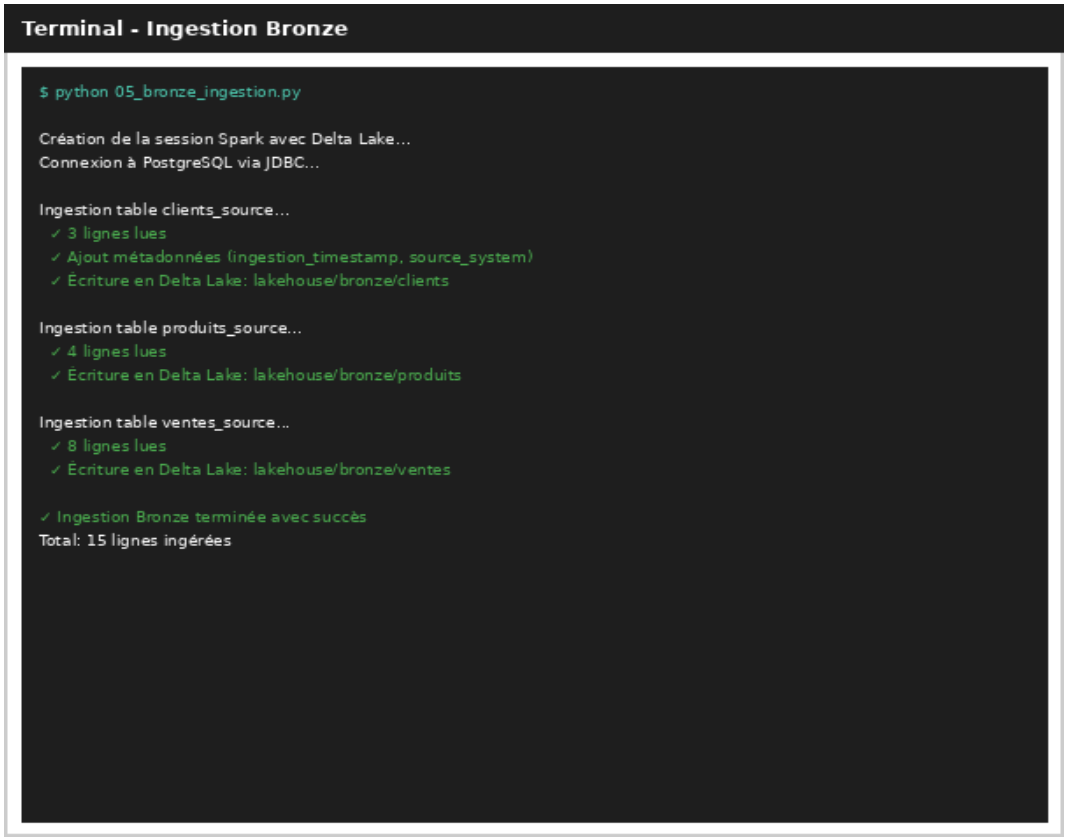

Figure 7.3 – Exécution de l'ingestion Bronze

7.4 Vérification des données Bronze

Le script 06_verify_bronze.py lit les tables Delta Lake de la couche Bronze et affiche le nombre de lignes et un échantillon de données pour chaque table, confirmant que l'ingestion s'est déroulée correctement.

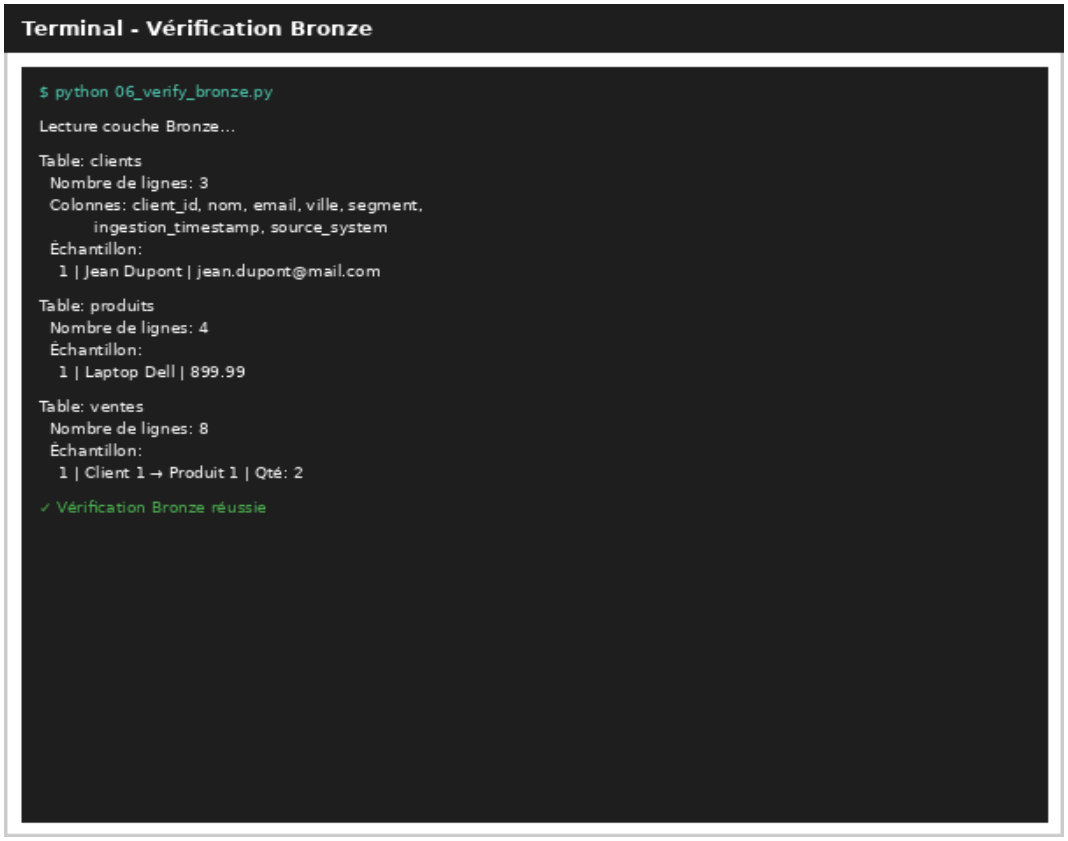

Figure 7.4 – Vérification des données Bronze

8. Projet Final - Intégration Complète

8.1 Objectif du Projet

Le projet final consiste à créer un pipeline de données complet de bout en bout : lecture depuis PostgreSQL, ingestion en Bronze, transformation en Silver, agrégation en Gold, et génération d'un rapport analytique avec les insights clés.

8.2 Transformation Bronze → Silver

Le script 07_silver_transformation.py lit les données Bronze, applique le nettoyage (suppression des doublons, standardisation des noms, validation des emails), ajoute des métadonnées de qualité, puis écrit les résultats en Delta Lake dans la couche Silver.

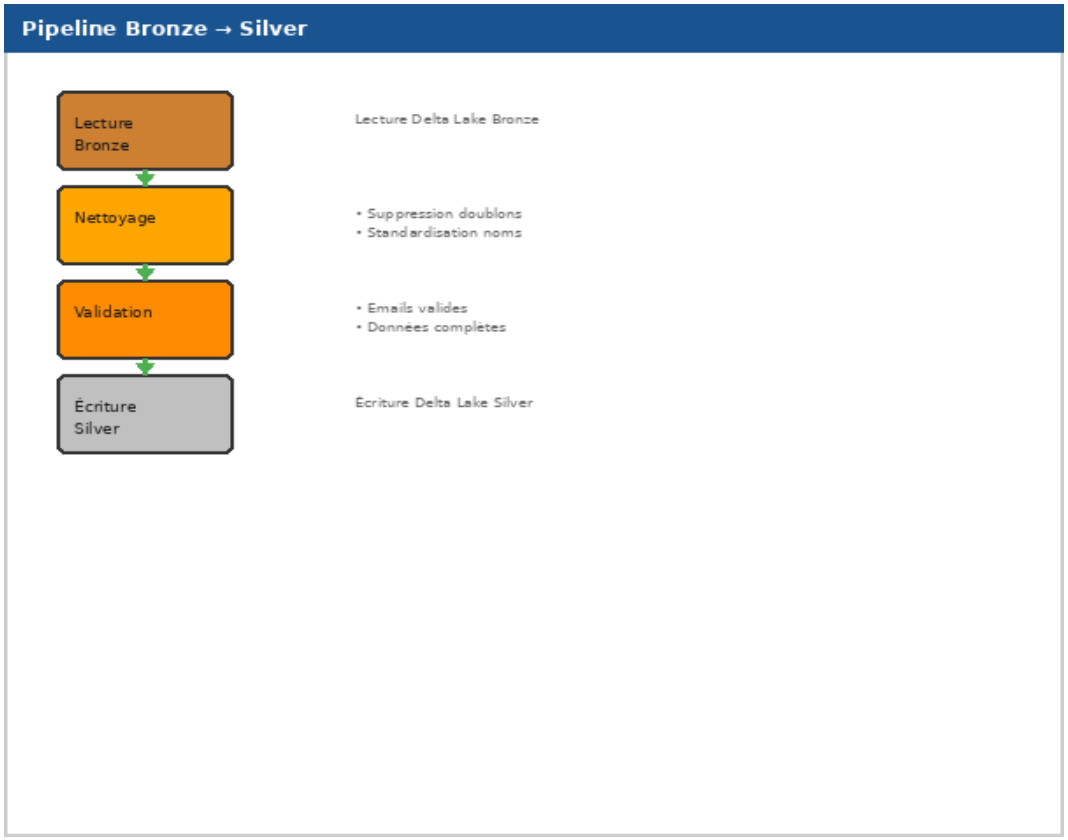

Figure 8.1 – Pipeline Bronze → Silver

8.3 Agrégation Silver → Gold

Le script 08_gold_aggregation.py lit les données Silver, effectue des agrégations par jour (nombre de ventes, chiffre d'affaires total, panier moyen), et écrit les résultats en Delta Lake dans la couche Gold pour alimenter les analyses métier.

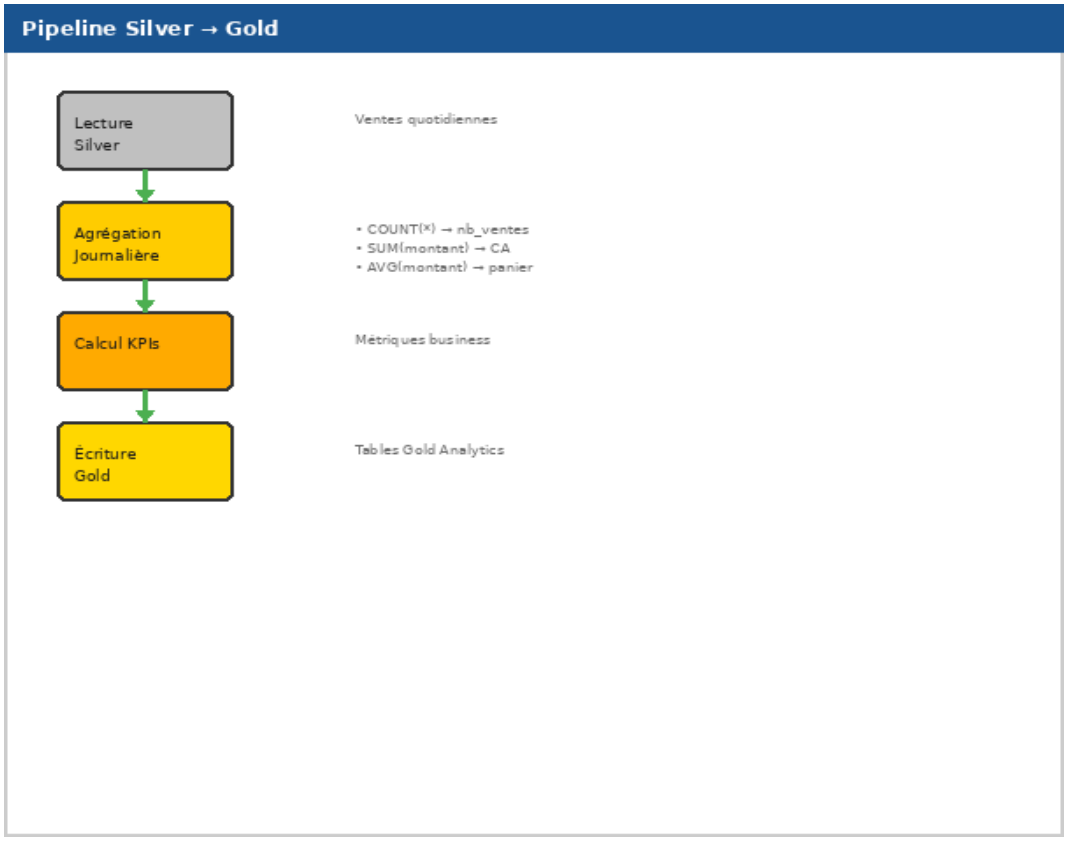

Figure 8.2 – Pipeline Silver → Gold

8.4 Génération du Rapport Final

Le script 09_generer_rapport.py lit les données Gold et génère un rapport analytique complet avec : statistiques globales (total des ventes, chiffre d'affaires, panier moyen), top 5 des meilleurs jours par chiffre d'affaires, et autres insights clés pour la prise de décision.

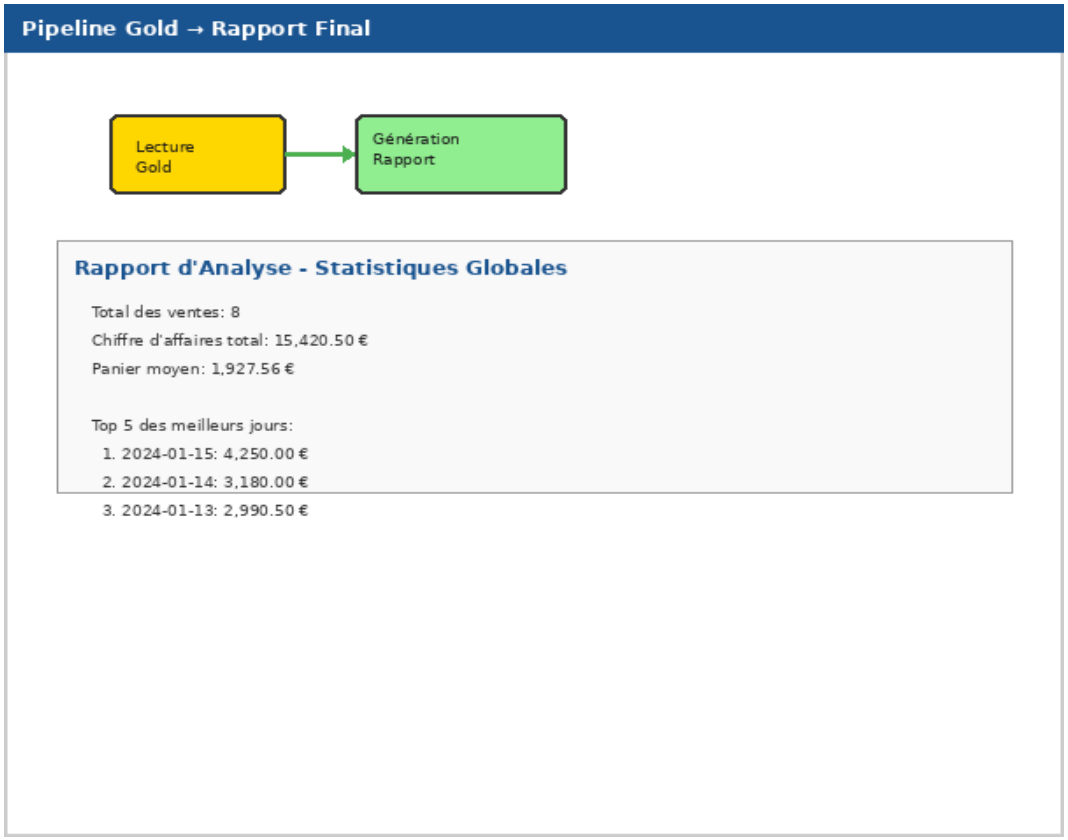

Figure 8.3 – Génération du rapport final depuis Gold

9. Conclusion

Ce travail pratique a permis d'acquérir une compréhension approfondie et une expérience concrète des bonnes pratiques modernes en ingénierie des données. Les concepts clés maîtrisés incluent :

• **Conservation de l'historique** : Grâce aux SCD Type 2 et à Data Vault, chaque modification des données est historisée, permettant des analyses temporelles précises.

• **Gestion des métadonnées et versioning** : Delta Lake assure les transactions ACID et le Time Travel, permettant de revenir à n'importe quelle version antérieure des données.

• **Transformation et agrégation** : L'architecture Lakehouse en trois couches (Bronze-Silver-Gold) structure le traitement des données de manière claire et évolutive.

• **Production d'insights exploitables** : Le pipeline complet transforme les données transactionnelles brutes en indicateurs métier actionnables.

Ce projet constitue une base solide pour concevoir et implémenter des pipelines de données industriels, fiables, évolutifs et faciles à maintenir. Les compétences acquises sont directement applicables dans des environnements de production professionnels.