

Projet Data Engineering on Cloud

Analyse des transactions bancaires pour la
gestion des clients et comptes avec Microsoft
Fabric

Auceane TITOT Lena DEMANOU
Encadré par : Alexandre Bergère

MAI 2025

Table des matières

Introduction	3
Contexte	3
Objectif du Projet	3
Présentation des données	3
Sources et préparation	3
Outils et Technologies utilisés	4
Choix technologiques	4
Architecture du projet	5
Implémentation et traitements techniques	5
Création du workspace et du Lakehouse	5
Ingestion des données	6
Transformation des Données	7
Modélisation des données	8
Création des Tables SCD	9
Organisation des notebooks et pipeline d'exécution	10
Workflow & Pipeline	11
Visualisation Power BI	12
Connexion Power BI	12
Création des visuels	14
Conclusion	15
Annexes	15

Introduction

Contexte

Les établissements bancaires manipulent quotidiennement de vastes volumes de données transactionnelles. Ces données, souvent dispersées entre plusieurs systèmes, sont essentielles pour :

- détecter les comportements à risque,
- optimiser la relation client,
- suivre les flux financiers en temps réel,
- et piloter les performances commerciales.

Cependant, ces données sont souvent sous-exploitées à cause :

- d'un manque de consolidation centralisée,
- de la difficulté à les traiter à grande échelle,
- de l'absence de visualisation claire et exploitable pour les équipes métier.

Objectif du Projet

Un projet de Data Engineering vise à mettre en place l'infrastructure et les processus nécessaires pour permettre à une organisation de collecter, stocker, traiter et analyser efficacement ses données afin de répondre à un besoin et d'en tirer des informations précieuses pour la prise de décision et d'autres cas d'utilisation.

Ce projet vise à construire une architecture cloud sur Microsoft Fabric afin de :

- centraliser les données clients, de comptes et de transactions,
- structurer les traitements selon l'architecture en médaillon,
- exposer des indicateurs métiers fiables et dynamiques (KPI, tendances),
- fournir aux décideurs une interface Power BI pour piloter l'activité bancaire.

La solution met en œuvre un pipeline data complet avec ingestion, transformation, modélisation, et visualisation Power BI, en suivant une architecture en médaillon.

Présentation des données

Sources et préparation

Les données utilisées dans ce projet proviennent du jeu de données Bank Transaction Data publié sur Kaggle. Il s'agit d'un fichier Excel simulant des opérations bancaires (dépôts, retraits, soldes) sur plusieurs comptes clients. Le fichier source a été divisé en deux jeux de données principaux : comptes et transactions.

Afin de compléter ce dataset et d'enrichir le modèle analytique, on a On a généré des données fictives en utilisant la librairie Python Faker afin qu'elles soient associées à chaque "account_no" garantissant une structure cohérente

Cela permet de simuler :

- des noms, emails et pays variés,
- des dates d'inscription,
- un type client (particulier / entreprise).

Chaque fichier est structuré comme suit :

- clients.csv : customer_id, account_no, nom, pays, type client, date d'inscription (génééré)
- accounts.csv : solde initial/final, nombre de transactions (provenant de bank.xlsx)
- transactions.csv : retraits, dépôts, libellé, solde courant, date (provenant de bank.xlsx)

Les fichiers Excel ont été nettoyés (suppression de lignes vides, correction de formats) puis exportés en CSV pour ingestion dans le Lakehouse Fabric.

Outils et Technologies utilisés

Choix technologiques

Le projet s'appuie sur la suite Microsoft Fabric, qui offre une solution tout-en-un pour le traitement, le stockage et la visualisation des données.

Dans le cadre de ce projet d'analyse bancaire, nous avons fait le choix d'utiliser un Lakehouse au sein de Microsoft Fabric plutôt qu'un Data Warehouse classique.

Ce choix est motivé par les besoins spécifiques du pipeline :

1. Ingestion de données brutes hétérogènes :

Les fichiers d'entrée (au format .csv) nécessitent une ingestion flexible et tolérante, ce que permet la zone Bronze du Lakehouse, contrairement à un entrepôt relationnel traditionnel plus rigide.

2. Nettoyage et transformation avec PySpark :

Les traitements réalisés (nettoyage, enrichissement, historisation SCD) sont plus naturellement implémentés via des notebooks PySpark. Le Lakehouse permet de travailler directement en Delta Lake, offrant des performances élevées et une gestion native des versions.

3. Séparation logique des zones (médaillon) :

L'architecture Bronze / Silver / Gold est typique du modèle Lakehouse et facilite la traçabilité, le debugging et la qualité des données.

4. Compatibilité Power BI native :

Les vues générées en zone Gold sont directement exposables à Power BI grâce à la structure Delta, sans nécessiter de duplication ou ETL supplémentaire.

Par ailleurs, le stockage OneLake est utilisé automatiquement dès lors que nous créons un Lakehouse dans Fabric. OneLake agit comme un Data Lake unifié pour tous les services Fabric : Lakehouse, Power BI, Data Factory, Notebooks.

Architecture du projet

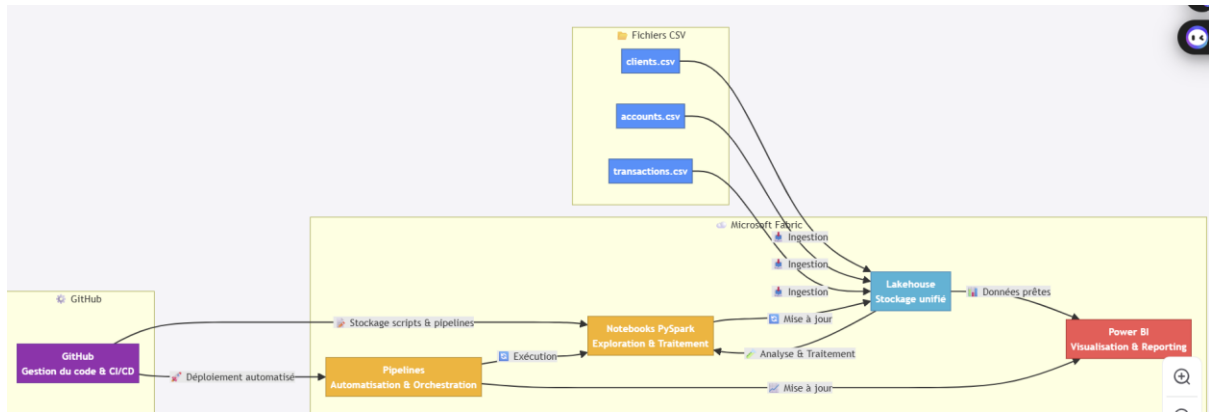


Figure 1: Schéma d'architecture

Implémentation et traitements techniques

Création du workspace et du Lakehouse

Les données sont stockées dans le Lakehouse de Microsoft Fabric, puis traitées en plusieurs couches (bronze, silver, gold).

Créer un espace de travail



Nom *

Projet_Data_Bancaire

✓ Ce nom est disponible

Figure 2 : Création du Workspace

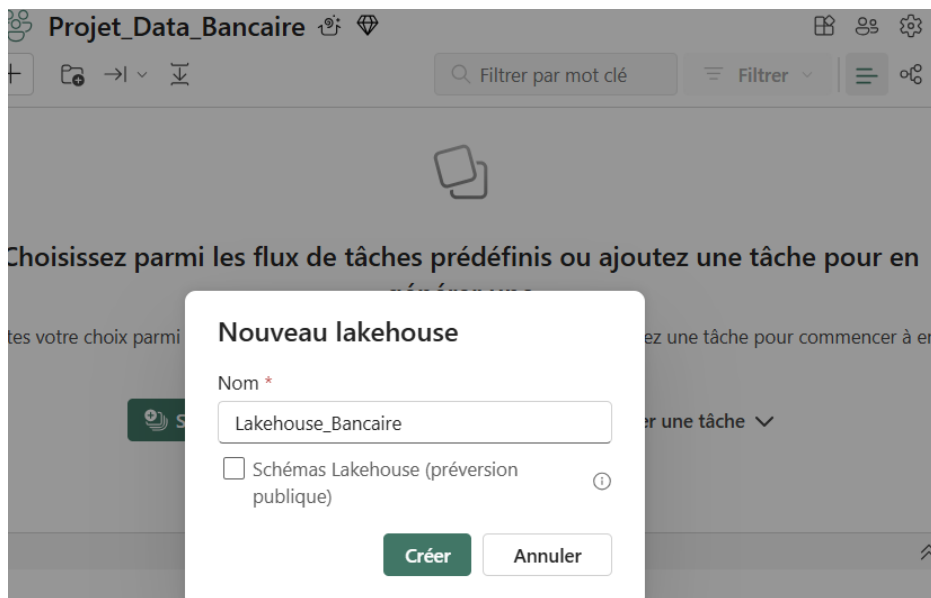


Figure 3 : Création du lakehouse

Ingestion des données

Pour l'ingestion des données, nous avons créé dans le dossier "Files" un sous-dossier nommé "bronze" dans lequel nous avons chargé les trois fichiers sources : clients.csv, accounts.csv et transactions.csv

Charger des fichiers

Files/bronze/

☐ Remplacer si les fichiers existent déjà

Charger

Chargements en cours

Ignorer: Terminé Tout

Nom de fichier	Nom de Lakehouse			
accounts.csv	Lakehouse_Bancaire	✓	6 KB / 6 KB	
clients.csv	Lakehouse_Bancaire	✓	6 KB / 6 KB	
transactions.csv	Lakehouse_Bancaire	✓	25 KB / 25 KB	

Les fichiers CSV bruts (clients.csv, accounts.csv, transactions.csv) sont ingérés dans la zone Bronze sous forme de tables Delta.

Transformation des Données

Les données sont nettoyées et enrichies via PySpark dans le notebook2

```
# Nettoyage des emails : suppression des espaces et mise en minuscule
clients_clean = clients_clean.withColumn("email_clean", regexp_replace(lower(col("email")), "\\s+", ""))

# Nettoyage des comptes
accounts_clean = accounts_df.dropDuplicates(["account_id"])

# Nettoyage des transactions
transactions_clean = transactions_df \
    .withColumn("transaction_date", to_date("transaction_date")) \
    .filter((col("deposit_amount").isNotNull()) | (col("withdrawal_amount").isNotNull())) \
    .filter(~((col("deposit_amount") == 0) & (col("withdrawal_amount") == 0)))

# Ajout de colonnes calculées enrichies
transactions_enriched = transactions_clean \
    .withColumn("transaction_type", when(col("withdrawal_amount") > 0, lit("withdrawal")) \
    .when(col("deposit_amount") > 0, lit("deposit")) \
    .otherwise(lit("unknown"))) \
    .withColumn("net_transaction", (col("deposit_amount") - col("withdrawal_amount"))) \
    .withColumn("year", year("transaction_date")) \
    .withColumn("month", month("transaction_date")) \
    .withColumn("weekday", dayofweek("transaction_date")) \
    .withColumn("is_large_deposit", when(col("deposit_amount") > 10000, lit(True)).otherwise(lit(False))) \
    .withColumn("is_unusual_withdrawal", when(col("withdrawal_amount") > 5000, lit(True)).otherwise(lit(False)))

# Enrichissement par jointure client (pays, type)
transactions_joined = transactions_enriched \
    .join(clients_clean.select("client_id", "country", "account_type"), on="client_id", how="left")

# Sauvegarde des données Silver
clients_clean.write.format("delta").mode("overwrite").saveAsTable("silver_clients")
accounts_clean.write.format("delta").mode("overwrite").saveAsTable("silver_accounts")
transactions_joined.write.format("delta").mode("overwrite").saveAsTable("silver_transactions")
```

Il effectue :

Nettoyage avancé

- Supprimer les valeurs aberrantes : dépôts ou retraits négatifs
- Gérer les valeurs nulles : solde manquant, dates manquantes
- Filtrer les transactions incohérentes (ex. 0 partout)
- Nettoyer les noms clients (espaces en trop, majuscules...)
- Nettoyage des données (dropDuplicates, typage des dates)

Jointures enrichissantes

- Ajouter les noms de clients directement dans les transactions
- Créer une jointure client-compte pour propager le pays ou type client
- Créer une table dim_date enrichie : année, mois, trimestre, weekday

Enrichissement analytique

- Ajouter une colonne net_transaction = deposit - withdrawal
- Calculer solde après transaction si possible (ordonné par date)
- Ajouter des flags métier :
 - is_large_deposit : si dépôt > 10 000€
 - is_unusual_withdrawal : si retrait > moyenne x 2
- Ajouter une durée de relation client (à partir de signup_date)
- Ajout de colonnes calculées sur les transactions :

- Type d'opération (dépôt ou retrait)
- Dimensions temporelles (year, month, weekday)
- Résultat : silver_clients, silver_accounts, silver_transactions

```
1 df = spark.sql("SELECT * FROM Lakehouse_Bancaire.silver_transactions LIMIT 1000")
2 display(df)
```

val_amount	transaction_date	ABC transaction_channel	ABC fees	ABC transaction_type	12 net_transaction	123 year	123 month	123 weekday	0/1 is_large_deposit	0/1 is_unusual_withdrawal
	2024-12-05	Mobile App	4.08	deposit	2769.7	2024	12	5	false	false
	2023-01-09	Mobile App	1.62	withdrawal	-3373.63	2023	1	2	false	false
	2024-02-11	Agence	3.34	deposit	1425.67	2024	2	1	false	false
	2023-09-12	Agence	3.28	deposit	2947.28	2023	9	3	false	false
	2024-07-27	ATM	2.95	withdrawal	-6149.07	2024	7	7	false	true
	2022-08-25	Agence	3.92	deposit	4714.35	2022	8	5	false	false
	2023-02-02	Web	1.04	deposit	2800.88	2023	2	5	false	false
	2022-12-05	Agence	2.5	withdrawal	-4208.82	2022	12	2	false	false
	2022-12-19	Mobile App	3.18	withdrawal	-6041.78	2022	12	2	false	true
	2024-06-10	ATM	1.03	deposit	3056.04	2024	6	2	false	false

Modélisation des données

- Objectif : Construire le modèle en étoile dans la zone Gold.
- Tables créées :
 - dim_client, dim_account, dim_date (données de référence)
 - fact_transaction (mesures, clés étrangères, jointures)
- Résultat : Tables prêtes pour l'analyse multidimensionnelle.

```
# DIMENSIONS
dim_client = clients.withColumn("client_sk", monotonically_increasing_id())
dim_account = accounts.withColumn("account_sk", monotonically_increasing_id())
dim_date = transactions.select("transaction_date").dropna().dropDuplicates().withColumn("date_sk", monotonically_increasing_id())

dim_client.write.format("delta").mode("overwrite").saveAsTable("dim_client")
dim_account.write.format("delta").mode("overwrite").saveAsTable("dim_account")
dim_date.write.format("delta").mode("overwrite").saveAsTable("dim_date")

# TABLE DE FAITS
fact_transaction = transactions \
    .join(dim_client, on="client_id", how="left") \
    .join(dim_account, on="account_id", how="left") \
    .join(dim_date, on="transaction_date", how="left") \
    .select("client_sk", "account_sk", "date_sk", "withdrawal_amount", "deposit_amount", "net_transaction", "transaction_type")

fact_transaction = transactions \
    .join(dim_client, transactions.account_id == dim_client.account_no, how="left") \
    .join(dim_account, transactions.account_id == dim_account.account_id, how="left") \
    .join(dim_date, on="transaction_date", how="left") \
    .select("client_sk", "account_sk", "date_sk", "withdrawal_amount", "deposit_amount", "net_transaction", "transaction_type")

fact_transaction.write.format("delta").mode("overwrite").saveAsTable("fact_transaction")

print("Modèle en étoile enregistré avec succès.")
```

Le modèle en étoile comprend :

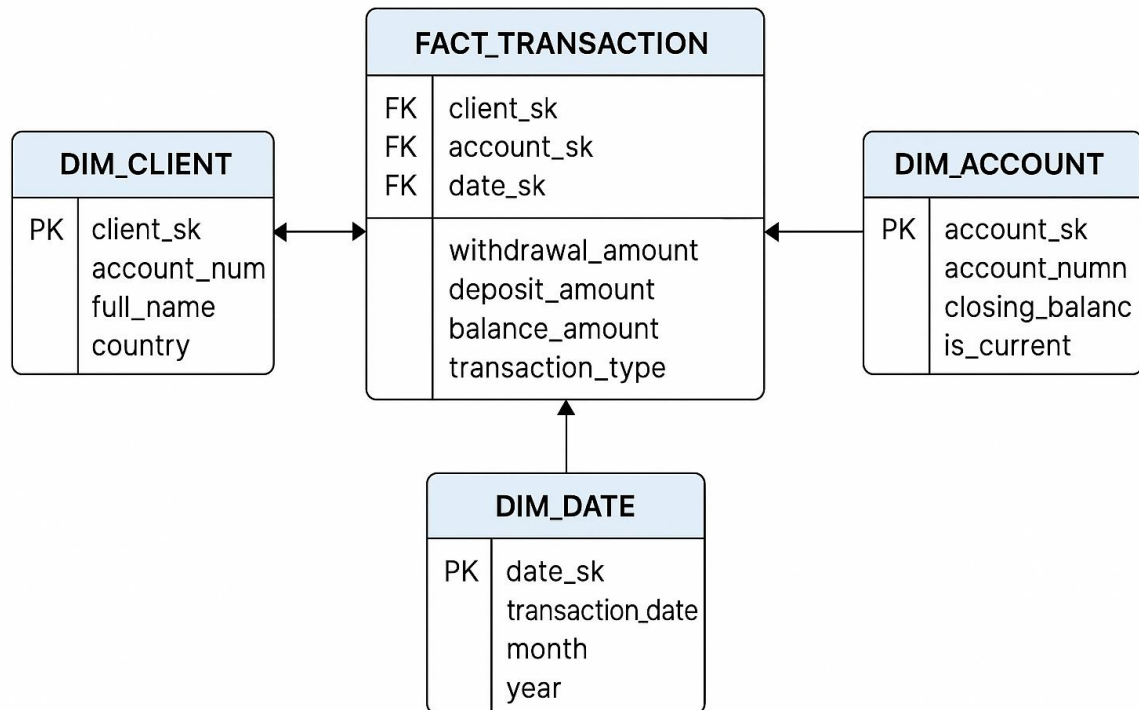
Dimensions :

- `dim_client` (SCD1)

- `dim_account` (statique ou SCD2 si enrichie)
- `dim_date` (générée à partir des dates de transaction)

Fait :

- `fact_transaction` : enregistre les retraits, dépôts, soldes, types de transaction, etc.



Création des Tables SCD

- Objectif : Implémenter SCD1 et SCD2.
- SCD1 : dim_client → écrasement des modifications (email, type client).
- SCD2 : dim_account → conservation de l'historique via l'ajout de colonnes d'historisation : start_date, end_date, is_current

```

# Notebook : 04_scd_et_versioning.py
# Objectif : SCD1 et SCD2

from delta.tables import DeltaTable
from pyspark.sql.functions import current_date, lit

clients_df = spark.read.format("delta").load("Files/silver/clients")
accounts_df = spark.read.format("delta").load("Files/silver/accounts")

# SCD1 : client (overwrite)
clients_df.write.format("delta").mode("overwrite").save("Files/gold/dim_client")

# SCD2 : account
accounts_scd2 = accounts_df \
    .withColumn("start_date", current_date()) \
    .withColumn("end_date", lit(None).cast("date")) \
    .withColumn("is_current", lit(True))

if DeltaTable.isDeltaTable(spark, "Files/gold/dim_account"):
    target = DeltaTable.forPath(spark, "Files/gold/dim_account")
    source = accounts_scd2.alias("source")









    target.alias("target").merge(
        source,
        "target.account_id = source.account_id AND target.is_current = true"
    ).whenMatchedUpdate(condition="target.balance_amount != source.balance_amount",
        set={
            "end_date": "current_date()",
            "is_current": "false"
        }).whenNotMatchedInsertAll()
else:
    accounts_scd2.write.format("delta").mode("overwrite").save("Files/gold/dim_account")

```

Organisation des notebooks et pipeline d'exécution

Afin d'assurer une bonne lisibilité du pipeline de traitement, nous avons créé un notebook distinct pour chaque étape clé de la solution : ingestion, transformation, modélisation, gestion des dimensions SCD et préparation pour Power BI.

Chaque notebook est nommé de façon explicite, ce qui facilite la compréhension du processus et l'analyse en cas d'erreur. Chaque notebook est autonome, facilitant la maintenance et le debug.

	01_ingestion_bronze	Bloc-notes	—	Auceane-...	—	—	—	—
	02_transformation_silver	Bloc-notes	—	Auceane-...	—	—	—	—
	03_modelisation_gold	Bloc-notes	—	Auceane-...	—	—	—	—
	04_scd_et_versioning	Bloc-notes	—	Auceane-...	—	—	—	—
	05_powerbi_extraction	Bloc-notes	—	Auceane-...	—	—	—	—
	Lakehouse_Bancaire	Lakehouse	—	Auceane-...	—	—	—	—
	Lakehouse_Bancaire	Modèle s...	—	Projet_Da...	25/05/2025...	Non applic...	—	—
	Lakehouse_Bancaire	Point de t...	—	Auceane-...	—	—	—	—
	Pipeline_Projet_Bancaire	Data pipe...	—	Auceane-...	—	—	—	—

Les traitements sont orchestrés dans Fabric à travers 5 notebooks :

1. `01_ingestion_bronze.py` – ingestion brute,
2. `02_transformation_silver.py` – nettoyage et enrichissements,
3. `03_modelisation_gold.py` – construction des dimensions et faits,
4. `04_scd_et_versioning.py` – gestion des historiques SCD,
5. `05_powerbi_extraction.py` – création des vues métier pour Power BI.

Les données sont transformées via une série de notebooks PySpark orchestrés dans Microsoft Fabric. Voici les étapes clés du traitement :

- `01_ingestion_bronze.py` : ingestion brute des fichiers CSV dans le Lakehouse (format Delta)
- `02_transformation_silver.py` :
 - Nettoyage :
 - Suppression des transactions nulles
 - Nettoyage des noms clients (majuscules, espaces)
 - Enrichissement :
 - `transaction_type` : dépôt ou retrait
 - `net_transaction` : montant net = dépôt - retrait
 - `is_large_deposit` (dépôt > 10 000€)
 - `is_unusual_withdrawal` (retrait > 5 000€)
 - Jointure pour ajouter `country`, `client_type` dans les transactions
- `03_modelisation_gold.py` : création du schéma en étoile avec :
 - `fact_transaction`
 - `dim_client`, `dim_account`, `dim_date`
- `04_scd_et_versioning.py` :
 - SCD1 : mise à jour directe des clients
 - SCD2 : historisation des comptes (avec `start_date`, `end_date`, `is_current`)
- `05_powerbi_extraction.py` :
 - `kpi_clients` : total dépôts/retraits, solde moyen, nb comptes
 - `kpi_monthly` : activité agrégée par mois

Workflow & Pipeline

Chaque notebook est conçu pour être exécuté indépendamment, mais orchestré via un pipeline Fabric.

Pipeline_Projet_Bancaire

pow

Essai : 27 jours r...

Accueil

Activités

Exécuter

Affichage

Valider

Annuler

Planification

Ajouter un déclencheur

Afficher l'historique de l'exécution

Bloc-notes

01_ingestion_bronze

02_transformation_silver

03_modelisation_gold

04_scd_et_versioning

05_powerbi_extraction

Réglages

Variables

Paramètres

Sortie

Variables de bibliothèque (préversion)

ID d'exécution de pipeline: cfc1439d-800b-4d90-9c55-2e9224572a57

État du pipeline En cours

Exporter au format CSV

Filtrer

Options d

Affichage des éléments 1-5

Nom de l'activité	Statut de l'activité	Début de l'exécution	Durée	Entrée	Sortie
05_powerbi_extraction	En cours	5/25/2025, 8:36:13 PM	24s		
04_scd_et_versioning	Opération réussie	5/25/2025, 8:35:36 PM	36s		
03_modelisation_gold	Opération réussie	5/25/2025, 8:34:44 PM	52s		
02_transformation_silver	Opération réussie	5/25/2025, 8:33:51 PM	52s		
01_ingestion_bronze	Opération réussie	5/25/2025, 8:33:14 PM	36s		

Visualisation Power BI

Connexion Power BI

La connexion Power BI se fait directement sur les vues Gold via Direct Lake.

Obtenir les données

Rechercher

Tout

Fichier

Base de données

Microsoft Fabric

Power Platform

Azure

Services en ligne

Autre

Microsoft Fabric

- Modèles sémantiques Power BI
- Flux de données
- Datamarts (préversion)
- Entrepôts
- Lakehouses
- Bases de données KQL
- Metric Sets
- Base de données SQL

Connecteurs certifiés | Applications modèles

Se connecter

Annuler

Connexion à vos données

Databases

Rechercher

Paramètres

Sélectionnez la base de données ou des tables spécifiques aux laquelle vous souhaitez vous connecter. [En savoir plus](#)

- powerbi://api.powerbi.com/v1.0/myorg/Projet_Data_Bancaire
- ☒ Lakehouse_Bancaire

Envoyer

Annuler

Découvrez les données de votre organisation et utilisez-les pour créer des rapports

Tout

Mes données

Approuvé dans votre organisation

Rechercher

Filtrer par mot clé

Filtrer(1)

<div><div></div></div> Nom	Propriétaire	Actualisé	Emplacement	Approbation	Confidentialité
<div><div></div></div> Lakehouse_Bancaire	Auceane-Eve TITOT	—	Projet_Data_Bancaire	—	—

Données

Rechercher

>

dim_account

>

dim_client

>

dim_date

>

fact_transaction

>

kpi_clients

>

kpi_monthly

Création des visuels

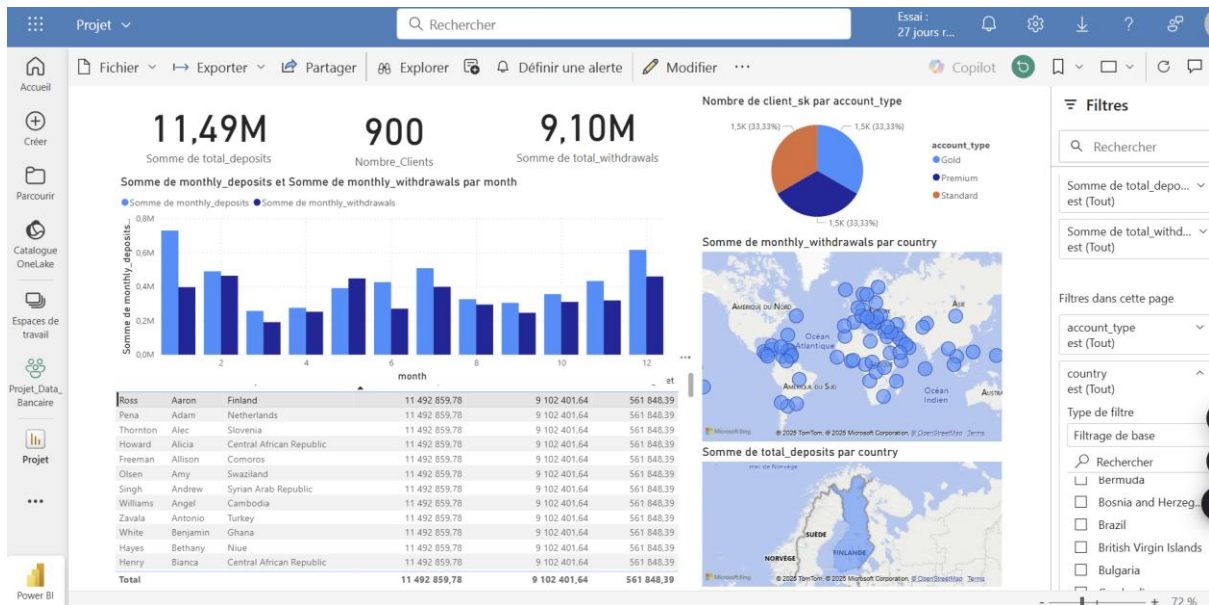
Le rapport Power BI est connecté directement au Lakehouse de Microsoft Fabric, en utilisant les vues agrégées `kpi_clients` et `kpi_monthly`. Ces vues permettent de visualiser les comportements clients, la répartition géographique, et l'évolution des soldes et opérations.

Les visuels créés sont :

- Cartes de KPI : Total des dépôts, retraits, nombre de clients
- Graphique en barres : Dépôts/retraits mensuels (par mois et année)
- Carte géographique : Répartition des clients par pays
- Diagramme circulaire : Types de clients (particuliers vs entreprises)
- Tableau détaillé : Nom du client, solde moyen, total déposé/retrait, nombre de comptes

Des segments (slicers) sont ajoutés pour filtrer les vues par pays, année ou type de client. Le tableau de bord est rafraîchi automatiquement grâce aux pipelines Fabric qui alimentent les données chaque fois qu'un traitement est exécuté.

Ce rapport facilite l'analyse stratégique de l'activité bancaire à tous les niveaux : client, compte, temps et pays.



Conclusion

Ce projet a démontré la faisabilité de la mise en place d'un pipeline cloud moderne sur Microsoft Fabric. Les indicateurs obtenus permettent une meilleure compréhension du comportement client et de l'activité financière.

Ce projet de data engineering a permis de mettre en œuvre une architecture moderne dans le cloud à l'aide de Microsoft Fabric, autour d'un cas d'usage concret : l'analyse de données bancaires.

En partant de fichiers bruts, nous avons construit une chaîne de traitement complète, basée sur le modèle en médaille, jusqu'à la restitution visuelle dans Power BI. Les indicateurs obtenus permettent une meilleure compréhension du comportement client et de l'activité financière.

Annexes

Le code source complet du projet est disponible dans le dépôt GitHub suivant : <https://github.com/auceanev/data-engineering-bank-on-fabric/tree/main>

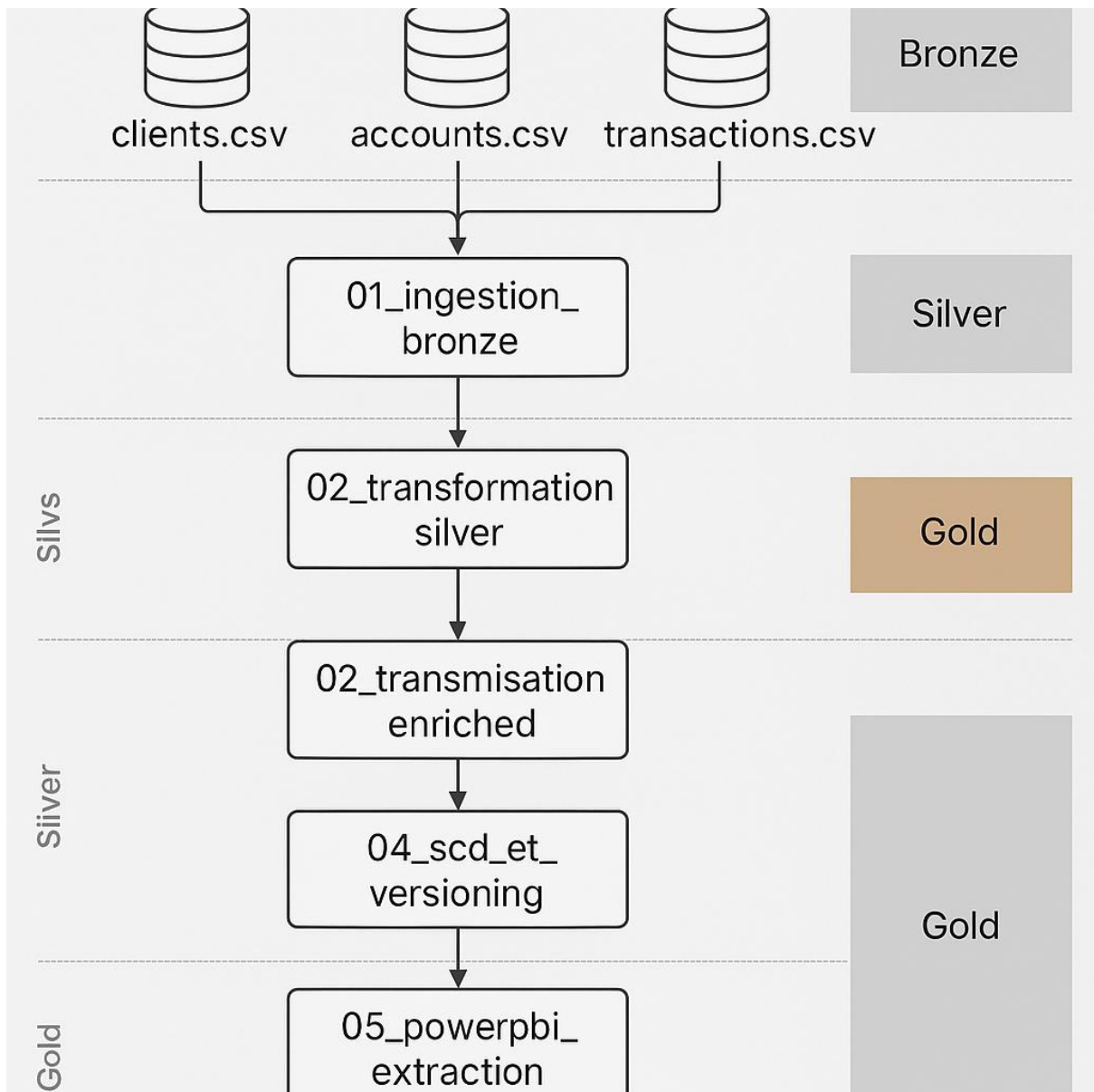


Figure 4 : architecture médaillon