Final Project Report
Local Lakehouse : Bronze → Silver → Gold

Programme : Data & Applications – Engineering (FD)
Cours : Data Engineering I
Année académique : 2025–2026
JABRY Rana / DEBBAGH Chama

1. Introduction et contexte du projet

L'objectif principal de ce projet est de construire une architecture **lakehouse locale** en utilisant **PySpark**, en suivant le modèle classique à trois couches : **Bronze, Silver et Gold**. Cette approche permet de gérer efficacement le cycle de vie des données, depuis leur ingestion brute jusqu'à leur transformation en tables analytiques exploitables.

Le projet devait également inclure une dimension d'optimisation des performances, avec une comparaison entre une version baseline et une version optimisée, en s'appuyant sur les plans d'exécution Spark et les métriques collectées via Spark UI.

2. Choix et description du jeu de données

2.1 Présentation du dataset

Le dataset utilisé est le **Wikipedia Clickstream Dataset**, disponible publiquement sur le site de Wikimedia :
https://dumps.wikimedia.org/other/clickstream/

Ce jeu de données décrit les flux de navigation entre les pages Wikipédia. Il enregistre combien de fois les utilisateurs passent d'une page source à une page cible sur une période donnée.

Les données sont fournies au format **TSV (Tab Separated Values)**, sans en-tête de colonnes. Pour ce projet, un fichier mensuel contenant environ **6 millions de lignes** a été utilisé.

2.2 Structure des données

Le dataset contient quatre colonnes principales :

- **source_page** : page Wikipédia d'origine
- **target_page** : page Wikipédia de destination
- **link_type** : type de lien (interne, externe, autre)
- **click_count** : nombre de clics observés pour cette transition

3. Architecture du projet

3.1 Organisation du code

Le projet est organisé autour des éléments suivants :

- `de1_project_config.yml` : fichier de configuration
- `DE1_Project_Notebook_EN.ipynb` : notebook principal
- `outputs/project/` : dossiers des couches Bronze, Silver et Gold
- `proof/` : plans d'exécution Spark sauvegardés
- `project_metrics_log.csv` : fichier de log des métriques de performance

> **Note :** Le dépôt du projet contient l'ensemble du code, des configurations et des preuves d'exécution.

> Les dossiers des couches **Bronze, Silver et Gold** ne sont pas présents dans le dépôt en raison de leur taille importante, si besoin on peut vous les envoyer.

3.2 Fichier de configuration YAML

Le fichier `de1_project_config.yml` contient notamment :

- Les chemins d'entrée et de sortie (raw, bronze, silver, gold)
- Les paramètres liés au partitionnement et au layout
- Les requêtes SQL utilisées pour les analyses (Q1, Q2, Q3)

4. Pipeline de traitement des données

4.1 Couche Bronze — Ingestion des données brutes

La couche Bronze correspond à l'ingestion des données brutes telles qu'elles sont fournies. Le fichier TSV est lu avec `spark.read.csv()` en précisant :

- `header = false` (absence d'en-tête)
- `sep = "\t"` (séparateur tabulation)

Les données sont ensuite écrites au format **CSV** dans le dossier Bronze.

Choix technique :
Le format CSV a été conservé à cette étape afin de rester au plus proche du format source, même si cela n'est pas optimal en termes de performance.

4.2 Couche Silver — Nettoyage et typage

La couche Silver est dédiée à l'amélioration de la qualité des données. Les transformations suivantes ont été appliquées :

- **Renommage des colonnes** (`_c0`, `_c1`, etc. → noms explicites)
- **Typage** : conversion de `click_count` en entier
- **Nettoyage des données** :
  - Suppression des valeurs nulles
  - Filtrage des valeurs négatives
  - Suppression des pages vides
  - Élimination des doublons

Les données sont ensuite écrites au format **Parquet** afin d'améliorer les performances de lecture et de réduire l'espace de stockage.

Résultat :
La table Silver finale contient **6 072 131 lignes**.

4.3 Couche Gold — Tables analytiques

La couche Gold contient trois tables analytiques, chacune correspondant à une question métier.

Q1 — Top pages par nombre total de clics

```
SELECT
  source_page AS page,
  SUM(click_count) AS total_clicks
FROM silver
GROUP BY source_page
ORDER BY total_clicks DESC
LIMIT 100
```

Q2 — Top transitions (source → target)

```
SELECT
  source_page,
  target_page,
  SUM(click_count) AS total_clicks
FROM silver
GROUP BY source_page, target_page
ORDER BY total_clicks DESC
```

```
LIMIT 100
```

Q3 — Statistiques par type de lien

```
SELECT
  link_type,
  COUNT(*) AS edges,
  SUM(click_count) AS total_clicks
FROM silver
GROUP BY link_type
ORDER BY total_clicks DESC
LIMIT 100
```

5. Analyse des performances

5.1 Version Baseline

Les trois requêtes ont d'abord été exécutées sur la table Silver sans optimisation particulière.
Les plans d'exécution Spark ont été sauvegardés dans le dossier `proof/`.

Les métriques suivantes ont été collectées via **Spark UI** :

- Nombre de fichiers lus
- Taille totale des données lues
- Volume de shuffle (read/write)
- Temps d'exécution des stages

(Des captures d'écran de Spark UI sont fournies dans le rapport "metrics_SparkUI_projet.pdf")

5.2 Stratégie d'optimisation

Une version optimisée de la table Silver a été créée à partir des principes suivants :

- Repartitionnement des données
- Organisation des données au sein des partitions
- Réécriture dans une table Silver optimisée

Les trois requêtes analytiques ont ensuite été réexécutées sur cette version optimisée, et les nouveaux plans d'exécution ont été sauvegardés.

7. Utilisation de l'IA générative dans le projet

7.1 Configuration du projet

On a utilisé l'IA pour nous proposer une structure initiale du fichier YAML, servant de point de départ.

7.2 Couche Silver

L'IA a proposé les **grandes lignes des transformations** à appliquer :

- Renommage des colonnes
- Typage
- Nettoyage et filtrage des données

Elle nous a également fourni le code, qu'on a analysé par la suite

7.3 Couche Gold

L'IA a suggéré les **idées d'analyses** (Q1, Q2, Q3). Mais l'ensemble des requêtes SQL a été rédigé par nous-mêmes, puis relu et corrigé à l'aide de l'IA.

7.4 Collecte des métriques

Face au grand nombre de jobs visibles dans Spark UI, l'IA a suggéré d'ajouter des cellules `count()` après chaque requête (Q1, Q2, Q3) pour nous faciliter la détection des jobs dans Spark UI