Construction Data Lake et Data Warehouse

Auteurs : Saad & Noam

Date : 27 avril 2025

1. Introduction

Technos utilisées:

- Kafka et ksqlDB pour les flux de données
- SQLite pour le Data Warehouse
- Fichiers JSON pour le Data Lake
- Apache Beam et `schedule` pour l'orchestration des jobs

2. Design du Data Lake

2.1 Structure du Data Lake

Le Data Lake est implémenté dans le dossier suivant :

- Chemin : `~/data_lake_project/data_lake`
- Les données provenant de Kafka (via ksqlDB) sont stockées dans des fichiers `.json`
- Les fichiers sont partitionnés par date, par exemple :
  - `~/data_lake_project/data_lake/2025-04-27/stream_transaction_log/transaction_log.json`
  - `~/data_lake_project/data_lake/2025-04-27/table_TOTAL_PAR_TRANSACTION_TYPE/TOTAL_PAR_TRANSACTION_TYPE.json`

2.2 Entités stockées

Actuellement, les entités suivantes sont stockées dans le Data Lake :

- Stream ksqlDB `TRANSACTIONS_STREAM` (via le topic `transaction_log`)
- Table ksqlDB `TOTAL_PAR_TRANSACTION_TYPE` (via le topic `TOTAL_PAR_TRANSACTION_TYPE`)

2.3 Mode de stockage

- **Streams** (ex. `TRANSACTIONS_STREAM`) :
  - Mode : **append** (ajout à la suite)
  - Justification : Les streams représentent un flux continu de données (comme des transactions individuelles). L'ajout à la suite permet de conserver un historique complet des messages.

- **Tables** (ex. `TOTAL_PAR_TRANSACTION_TYPE`) :
  - Mode : **overwrite** (écrasement)
  - Justification : Une table ksqlDB est une vue agrégée (ex. somme des montants par type de transaction). Chaque nouveau message reflète la dernière valeur agrégée, donc on écrase le fichier pour garder uniquement cette valeur.

2.4 Ajout d'un nouveau feed Kafka

En cas d'ajout d'un nouveau flux de données (feed) dans Kafka, par exemple un topic `user_log` pour des logs d'utilisateurs, la procédure est la suivante :

1. Créer un nouveau topic Kafka dans le Control Center (ex. `user_log`)
2. Créer un producteur (ex. `producer_user_log.py`) pour générer des messages, comme :

   ```
   json
   ```

   ```
   {"user_id": "USER-1234", "action": "login", "timestamp": "2025-04-27T12:00:00.000Z"
   ```

3. Si le feed est une table ksqlDB, mettre à jour `init_db.py` pour ajouter une table SQLite :

   ```
   python
   ```

   ```
   cursor.execute('''
   CREATE TABLE IF NOT EXISTS user_logs (
       user_id TEXT,
       action TEXT,
       timestamp TEXT
   )
   ''')
   ```

4. Mettre à jour `consumer.py` :
   - Ajouter le topic à `STREAM_TOPICS` (ou `TABLE_TOPICS` si c'est une table ksqlDB) :

     ```
     python
     ```

     ```
     STREAM_TOPICS = ['transaction_log', 'user_log']
     ```

   - Ajouter la logique pour insérer dans SQLite (si c'est une table ksqlDB) :

     ```
     python
     ```

     ```
     if topic == 'user_log':
         cursor.execute('''
         INSERT INTO user_logs (user_id, action, timestamp)
         VALUES (?, ?, ?)
         ''', (data['user_id'], data['action'], data['timestamp']))
     ```

5. Relancer les scripts : `init_db.py`, le producteur, et `consumer.py`

6. Mettre à jour `pipeline.py` pour inclure le nouveau topic dans la lecture du Data Lake :
    - Ajouter le dossier correspondant dans la fonction `read_data_lake()` :

      ```
      python

      for folder in ['stream_transaction_log', 'table_TOTAL_PAR_TRANSACTION_TYPE', 'st
      ```

- Réutilisation : Le script `consumer.py` est générique et gère plusieurs topics. Les dossiers du Data Lake sont créés dynamiquement (`stream_{topic}` ou `table_{topic}`), ce qui permet d'ajouter un nouveau feed sans modifier la logique principale.

3. Design du Data Warehouse (SQLite)

3.1 Structure du Data Warehouse

Le Data Warehouse est implémenté dans une base SQLite locale :

- Chemin : `~/data_lake_project/data_warehouse.db`
- Tables actuelles :
    - `total_par_transaction_type` (lié à la table ksqlDB `TOTAL_PAR_TRANSACTION_TYPE`)
    - `transactions` (lié au stream ksqlDB `TRANSACTIONS_STREAM`, non demandé mais inclus pour test)
    - `users` (pour la gouvernance)
    - `permissions` (pour la gouvernance)

3.2 Schéma Entité-Association

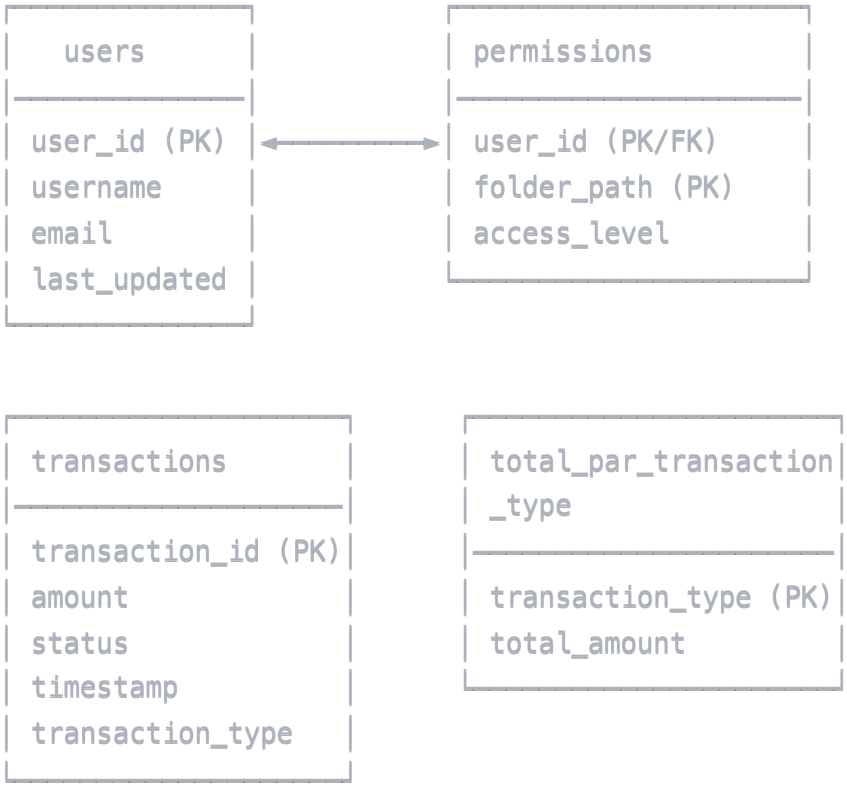
Clés primaires et relations :

- `total_par_transaction_type.transaction_type` → Clé primaire
- `transactions.transaction_id` → Clé primaire
- `users.user_id` → Clé primaire
- `permissions` → Clé primaire composée `(user_id, folder_path)`

Relation logique :

- `permissions.user_id` ←→ `users.user_id`

Diagramme simplifié :



4. Kafka Consumers

- Le consumer lit depuis les topics Kafka suivants :
  - `transaction_log` (lié au stream ksqlDB `TRANSACTIONS_STREAM`)
  - `TOTAL_PAR_TRANSACTION_TYPE` (lié à la table ksqlDB `TOTAL_PAR_TRANSACTION_TYPE`)
- Actions du consumer :
  - Stocke les messages dans le Data Lake (fichiers `.json`)
  - Enregistre les données dans SQLite :
    - `transaction_log` → Table `transactions`
    - `TOTAL_PAR_TRANSACTION_TYPE` → Table `total_par_transaction_type`
- Implémentation : Le fichier `consumer.py` gère cette logique

5. Gouvernance et Sécurité

5.1 Gestion des droits d'accès

- Table SQLite `permissions` :
  - Définit les droits d'accès des utilisateurs aux dossiers du Data Lake
  - Exemple : `(user_id=1, folder_path='~/data_lake_project/data_lake/2025-04-27/stream_transaction_log', access_level='read')`

5.2 Suppression des données historiques

- Implémentation : Le script `clean_old_data.py` supprime les dossiers du Data Lake datant de plus de 30 jours
- Fonctionnement :
  - Le script parcourt les dossiers du Data Lake (ex. `2025-04-27`)
  - Si un dossier est antérieur à 30 jours, il est supprimé avec `shutil.rmtree()`
  - Les dossiers non conformes au format de date (ex. `YYYY-MM-DD`) sont ignorés pour éviter les erreurs
- Exemple : Si aujourd'hui est le 27 avril 2025, tous les dossiers antérieurs au 28 mars 2025 (ex. `2025-03-27`) seront supprimés

6. Orchestration et Optimisation

6.1 Orchestration

- Implémentation : Le script `pipeline.py` utilise `apache-beam` et `schedule` pour exécuter le pipeline toutes les 10 minutes
- Fonctionnement :
  - Le pipeline lit les fichiers du Data Lake (générés par `consumer.py`)
  - Il insère les données dans SQLite (tables `transactions` et `total_par_transaction_type`)
  - Le pipeline est planifié pour s'exécuter toutes les 10 minutes via `schedule.every(10).minutes.do(run_pipeline)`
- Configuration Dataflow :
  - Projet : `mon-projet-dataflow`
  - Région : `us-central1`
  - Emplacements temporaires : `gs://mon-bucket-dataflow/temp` et `gs://mon-bucket-dataflow/staging`

6.2 Producer

- Le script `producer.py` génère des messages en boucle pour le topic `transaction_log`
- Configuration actuelle : 2000 messages générés (5 threads x 400 messages)

7. Instructions d'exécution

1. Lancer Kafka et ksqlDB
2. Initialiser la base SQLite :

   ```
   python3 init_db.py
   ```

3. Lancer le producer :

```
python3 producer.py
```

4. Lancer le consumer :

```
python3 consumer.py
```

5. Lancer le pipeline (dans un terminal séparé) :

```
python3.9 pipeline.py
```

6. (Facultatif) Lancer le script de nettoyage des données historiques :

```
python3 clean_old_data.py
```