Lab avec Airflow, Docker, SQL et Python

I. Contexte de l'exercice – Analyse des ventes d'un réseau international de magasins de sport

Vous êtes professionnel de la data dans une entreprise internationale spécialisée dans la vente d'articles de sport. L'entreprise possède des **magasins physiques (type Decathlon)** dans plusieurs pays d'Europe, d'Amérique du Nord et d'Asie.

Chaque jour, des milliers de transactions sont enregistrées via les **systèmes de caisse (POS)**.

Vous avez à votre disposition un fichier contenant **30 000 transactions de ventes** déjà prétraitées et structurées sous forme d'un fichier json nommée `pos_sales_data_30k.json`.

II. Tâches attendues

L'objectif est de construire un **data pipeline complet avec Airflow**, qui modélise et charge une **table intermédiaire** et **trois tables d'analyse KPI** dans une base de données **PostgreSQL** exécutée dans un environnement **Docker**.

1. **Créer la table `exploded_products`** à partir du fichier JSON
2. **Créer les 3 tables analytiques** à partir de `exploded_products`
3. **Conteneuriser le tout avec Docker** :
    - PostgreSQL (base de données)
    - Airflow (orchestration)
4. **Créer un DAG Airflow** qui orchestre :
    - le chargement du JSON
    - la transformation vers `exploded_products`
    - l'alimentation des 3 tables finales

III. Objectifs pédagogiques

- Travailler avec un fichier JSON structuré et imbriqué
- Appliquer les bonnes pratiques de modélisation analytique
- Automatiser un pipeline de transformation avec Airflow
- Conteneuriser l'environnement de travail avec Docker

III. Travaux pratiques

1. Source de données : `pos_sales_data_30k.json`

Ce fichier contient **30 000 transactions de ventes** enregistrées entre 2023 et 2025.
Il représente des tickets de caisse réels, avec parfois plusieurs articles par transaction.

Chaque enregistrement contient :

- des métadonnées de transaction (`transaction_id`, `store_id`, `purchase_timestamp`, etc.)
- des produits imbriqués (liste `products`)
- des informations de paiement, de remise, et de statut de retour

Extrait simplifié du JSON :

```
Unset
{
  "transaction_id": "TXN000123",
  "store_id": "STR1001",
  "store_country": "France",
  "purchase_timestamp": "2024-05-10 15:30:22",
  "payment_method": "Visa",
  "total_amount": 210.0,
  "discount_applied": 10.0,
  "return_status": "Not Returned",
  "products": [
    {
      "product_id": "P001",
      "product_name": "Running Shoes",
      "products_category": "Shoes",
      "quantity": 1,
      "price": 120.0
    },
    ...
  ]
}
```

---

2. Création d'une table intermédiaire `exploded_products`

Avant d'analyser les données, vous devez transformer le fichier JSON en une table **normalisée**.

Créer la table `exploded_products` qui contiendra **une ligne par produit vendu**, avec :

- les métadonnées de la transaction (magasin, date, devise, etc.)
- les données produit (catégorie, prix, quantité, etc.)

Structure de la table :

Nom de la colonne	Type SQL	Description / Raison
transaction_id	TEXT	Identifiant unique de la transaction (ticket de caisse)
store_id	TEXT	Identifiant du magasin
store_name	TEXT	Nom du magasin (ex : "Sports Hub")
store_country	TEXT	Pays dans lequel la transaction a été effectuée
store_city	TEXT	Ville du magasin
store_type	TEXT	Type de magasin (ex : Mall, Standalone Store)
purchase_timestamp	TIMESTAMP	Date et heure de l'achat
payment_method	TEXT	Moyen de paiement utilisé (ex : Visa, Cash, Apple Pay...)
currency	TEXT	Devise utilisée lors de la transaction (USD, EUR, CAD...)
total_amount	FLOAT	Montant total de la transaction (tous produits confondus)
total_quantity_sold	INTEGER	Nombre total de produits achetés dans cette transaction
discount_applied	FLOAT	Remise totale appliquée à la transaction (en devise locale)
return_status	TEXT	Statut du retour : Returned ou Not Returned
product_id	TEXT	Identifiant unique du produit vendu
product_name	TEXT	Nom du produit (ex : "Running Shoes")
products_category	TEXT	Catégorie du produit (Shoes, Accessories, Equipment…)
quantity	INTEGER	Quantité de cet article spécifique dans la transaction

price	FLOAT	Prix unitaire du produit (ajusté si besoin au coût de la vie par pays)

✅ C'est **à partir de cette table** que vous construirez les 3 tables analytiques ci-dessous.

💡 ***Conseil : à faire en Python avec json (import json) et psycopg2 (import psycopg2)***

---

3. Vos 3 tables analytiques à construire

À partir de la table intermédiaire `exploded_products`, construire les 3 tables suivantes

1. store_sales_summary

Objectif : analyser les performances par magasin

Nom de la colonne	Type SQL	Description / Raison
store_id	TEXT	Identifiant du magasin → **Granularité**
store_name	TEXT	Nom du magasin
store_country	TEXT	Pays du magasin
store_city	TEXT	Ville du magasin
store_type	TEXT	Type de magasin
total_sales_amount	FLOAT	Total des ventes (somme de total_amount)
total_quantity_sold	INTEGER	Quantité totale de produits vendus

---

2. daily_sales_country_currency

Objectif : suivre les ventes par jour, pays et devise

Nom de la colonne	Type SQL	Description / Raison
purchase_date	DATE	Date de la transaction
store_country	TEXT	Pays du magasin
currency	TEXT	Devise locale utilisée
daily_total_sales	FLOAT	Total des ventes pour la journée
number_of_transactions	INTEGER	Nombre de transactions uniques
average_transaction_value	FLOAT	Valeur moyenne des transactions

---

3. `payment_method_analysis`

Objectif : analyser les moyens de paiement utilisés

Nom de la colonne	Type SQL	Description / Raison
payment_method	TEXT	Méthode de paiement utilisée
total_transactions	INTEGER	Nombre total de transactions
total_sales_amount	FLOAT	Montant total des ventes
average_discount	FLOAT	Remise moyenne appliquée
return_rate	FLOAT	Pourcentage de transactions retournées
average_items_per_transaction	FLOAT	Nombre moyen d'articles par transaction

💡 ***Conseil : faisable en Python avec json (import json) et psycopg2 (import psycopg2) ou en SQL***

---

4. Partie Docker

Vous devez conteneuriser l'environnement complet :

- **PostgreSQL** : base de données relationnelle
- **Airflow** : orchestrateur des traitements

Utilisez un `docker-compose.yml` pour :

- démarrer Airflow avec les services nécessaires
- exposer un port PostgreSQL accessible localement

5. Partie Airflow

Vous devez créer un **DAG Airflow** qui :

1. Charge le fichier JSON dans la table `exploded_products`
2. Alimente les 3 tables analytiques avec les scripts SQL ou Python
3. S’exécute quotidiennement ou à la demande (mode batch, tous les 24h)

6. Partie Analytics

🧱 Table : `store_sales_summary`

1. Quels sont les 5 magasins ayant généré le plus de chiffre d'affaires ?
2. Quelle est la ville avec le chiffre d'affaires cumulé le plus élevé ?
3. Combien de magasins se trouvent dans chaque pays ?
4. Quelle est la quantité totale de produits vendus par pays ?

🧱 Table : `daily_sales_country_currency`

1. Quel est le chiffre d'affaires total en CAD ?
2. Combien de jours ont dépassé 6 000 € de ventes en France ?
3. Quel est le total des ventes par devise ?
4. Quelle est la valeur moyenne d'une transaction par pays ?

🧱 Table : `payment_method_analysis`

1. Quelle méthode de paiement génère le plus de ventes ?
2. Quelles méthodes ont un taux de retour supérieur à 10 % ?
3. Quelle méthode est associée au plus grand nombre d'articles en moyenne ?
4. Classement des méthodes par volume de transactions

---

7. Les fichiers sources générés attendues

- le fichier JSON source
- le `docker-compose.yml` (Airflow + Postgres)
- les scripts Python et SQL
- un DAG Airflow fonctionnel dans `/dags`