PROJET FINAL – SPARK SCALA

Module : Data Engineer – Spark & Scala

Durée estimée : 6jours

Modalité : Projet individuel

Thème : Système d'analyse de données e-commerce

PARTIE 1 : CONFIGURATION ET STRUCTURE DU PROJET

Question 1.1 – Structure de projet SBT

Objectif : Créer une arborescence de projet standard, modulaire et maintenable.Créez un projet Scala utilisant **SBT** (Simple Build Tool) avec la structure standard suivante :

```
EcommerceAnalytics/
├── build.sbt
├── README.md
└── src/main/scala/
|   └── com/
|       └── ecommerce/
|           └── analytics/
|               └── DataIngestion.scala
|               └── DataTransformation.scala
|               └── Analytics.scala
|               └── MainApp.scala
├── src/main/resources/
|   ├── application.conf
|   └── data/
|       ├── transactions.csv
|       ├── users.json
|       ├── products.parquet
|       └── merchants.csv
```

Remarque : Placez également les case classes dans un package `com.ecommerce.models`.

Description des données

merchants

Colonne	Type	Description
merchant_id	String	Identifiant unique du marchand (ex: M00001)
name	String	Nom du marchand ou de la boutique
category	String	Catégorie principale de produits vendus (ex: Electronics, Books...)
region	String	Région d'implantation géographique (ex: Nouvelle-Aquitaine)
commission_rate	Double	Taux de commission appliqué aux ventes (entre 0 et 1)
establishment_date	String(`yyyyMMdd`)	Date de création de l'établissement (ex: 20220918)

transactions

Colonne	Type	Description
transaction_id	String	Identifiant unique de la transaction
user_id	String	Référence vers l'utilisateur concerné par la transaction
product_id	String	Référence vers le produit acheté
merchant_id	String	Référence vers le marchand concerné
amount	Double	Montant payé pour la transaction (ex: 57.23)
timestamp	String (Format `yyyyMMddHHmmss`)	Date et heure de la transaction (ex: 20240701021822)
location	String	Ville ou lieu de la transaction (ex: Grenoble)
payment_method	String	Méthode de paiement (ex: CARD, CASH, CRYPTO, etc.)
category	String	Catégorie du produit ou service acheté (ex: Books, Electronics...)

users

Colonne	Type	Description
user_id	String	Identifiant unique de l'utilisateur
age	Integer	Âge de l'utilisateur
annual_income	Double	Revenu annuel de l'user (en euros)
city	String	Ville de résidence
customer_segment	String	Segment client (ex: Premium, Standard, Budget)
preferred_categories	Array[String]	Liste des catégories préférées (ex: ["Books", "Electronics"])
registration_date	String (Format `yyyyMMdd`)	Date d'inscription à la plateforme (ex: 20221123)

products

Colonne	Type	Description
`product_id`	String	Identifiant unique du produit (ex: P03521)
`name`	String	Nom du produit
`category`	String	Catégorie du produit (ex: Electronics, Books...)
`price`	Double	Prix du produit en euros
`merchant_id`	String	Référence vers le marchand qui vend le produit
`rating`	Double	Note moyenne attribuée par les utilisateurs (valeur entre 0.0 et 5.0)
`stock`	Integer	Quantité en stock disponible pour ce produit

Résumé des formats par fichier :

Fichier	Format	Contenu
`merchants.csv`	CSV	Informations sur les marchands
`transactions.csv`	CSV	Détails des achats
`users.json`	JSON	Informations clients
`products.parquet`	Parquet	Catalogue produit

Question 1.2 – Configuration du fichier `build.sbt`

Objectif : Gérer les dépendances nécessaires à l'utilisation de Spark.

Configurez le fichier `build.sbt` pour :

- Spécifier une version de Scala compatible avec Spark
- Ajouter les dépendances de Spark Core , Spark SQL et autres celon le besoin
- Intégrer la librairie **Typesafe Config** pour la gestion des configurations (ex: `com.typesafe.config`).

Question 1.3 – Documentation `README.md`

Objectif : Rendre le projet facilement exécutable par d'autres développeurs.

Rédigez un fichier `README.md` qui contient des instructions claires et détaillées pour :

- **Prérequis :** L'installation de Scala, Spark et SBT (optionnel).
- **Compilation :** Les commandes pour compiler le projet et générer le JAR.
- **Exécution locale :** Les commandes pour lancer l'application avec SBT en mode local.
- **Déploiement :** La commande `spark-submit` pour exécuter l'application sur un cluster.

PARTIE 2: INGESTION ET VALIDATION DES DONNÉES

Question 2.1 – Ingestion multi-format

Objectif : Lire et typer les données provenant de différentes sources.

1. **Créer une classe `DataIngestion`** pour centraliser les lectures de données.
2. **Définir des `case class`** pour chaque dataset (`Transaction`, `User`, `Product`, `Merchant`).
3. **Implémenter les méthodes de lecture** pour chacun des datasets en utilisant les `case class` pour obtenir des `Dataset[T]`.
    - `transactions.csv` : Définir explicitement le schéma
    - `users.json` : Gérer les champs potentiellement imbriqués.
    - `products.parquet` : Charger le fichier optimisé en format Parquet.
    - `merchants.csv` : Laisser Spark inférer le schéma.

Question 2.2 – Validation des données

Objectif : Nettoyer les données en éliminant les valeurs incohérentes.

Pour chaque dataset, implémentez une fonction de validation qui applique les règles suivantes :

- **Transactions :** `amount > 0` et `timestamp` a 14 caractères.
- **Users :** `age` entre 16 et 100, et `income > 0`.
- **Products :** `price > 0` et `rating` entre 1 et 5.
- **Merchants :** `commission_rate` entre 0 et 1.

Question 2.3 – Gestion d'erreurs et résumé

Objectif : Gérer les erreurs de lecture et fournir un bilan des données.

Lors du chargement des données, utilisez des blocs `try-catch` pour :

- Capturer et afficher les erreurs (fichier introuvable, structure incorrecte, etc.).
- Afficher le **nombre de lignes lues** avant validation.
- Afficher le **nombre de lignes valides** après validation.

PARTIE 3 : TRANSFORMATIONS AVANCÉES

Question 3.1 – UDF `extractTimeFeatures`

Objectif : Extraire des caractéristiques temporelles enrichies à partir d'un timestamp.

Écrivez une **UDF** (User-Defined Function) qui prend en entrée une chaîne de caractères (format `yyyyMMddHHmmss`) et renvoie une structure de données contenant :

- L'heure (`HH`).
- Le jour de la semaine (en texte).
- Le mois (en texte).
- Un flag `is_weekend` (1 si samedi/dimanche, 0 sinon).
- Une étiquette `day_period` (« Morning » [6h-12[, « Afternoon » [12h-18h[, « Evening » [18h-22h[, « Night » [22h, +]).
- Un flag `is_working_hours` (1 si entre 9h et 17h, 0 sinon).

Question 3.2 – Fonction `enrichTransactionData`

Objectif : Combiner et enrichir les données de plusieurs tables.

Dans une classe `DataTransformation`, implémentez une fonction qui :

- Joint les `DataFrame` de transactions, utilisateurs, produits et marchands.
- Applique l'UDF `extractTimeFeatures` pour ajouter les caractéristiques temporelles.
- Ajoute les colonnes suivantes en utilisant des **fonctions de fenêtrage** (Window functions) :
  - Le **rang** de la transaction par utilisateur ordonné par date de transaction.
  - Le **nombre total de transactions** par utilisateur.
  - Calcule la **tranche d'âge** du client (« Jeune » - de 25ans, « Adulte » 26 à 44 ans, « Age Moyen » 45 à 64ans, « Senior » + de 65ans).

Question 3.3 – Analyse par partition Window

Objectif : Utiliser les fonctions de fenêtrage pour détecter des comportements complexes.

Enrichissez le `DataFrame` des transactions pour :

- **Calculer le montant cumulé :** Créer une colonne contenant le montant total des transactions sur une fenêtre glissante de 7 jours.
- **Utilisateur Actif :** Créez une colonne (1 ou 0) qui indique si un utilisateur a **effectué une transaction au moins 5 jours** sur une période glissante de 7 jours.

PARTIE 4 : ANALYTIQUE BUSINESS

Question 4.1 – Rapport détaillé par marchand

Objectif : Générer des indicateurs de performance clés (KPI) par marchand.

Calculez les métriques suivantes pour chaque marchand (merchant) :

- Chiffre d'affaires total, nombre de transactions, et clients uniques.
- Montant moyen des transactions.
- Son **classement** par CA dans sa catégorie et sa région (utilisez `Window functions`).
- La commission totale perçue.
- La répartition des ventes par tranche d'âge des clients.

Question 4.2 – Analyse de cohortes utilisateurs

Objectif : Mesurer la fidélité et la rétention des clients.

Réalisez une analyse de cohortes en :

- Groupant les utilisateurs par leur **mois de première transaction**.

PARTIE 5 : OPTIMISATIONS SPARK

Question 5.1 – Optimisation du stockage

Objectif : Améliorer les performances en gérant l'utilisation de la mémoire.

Implémentez les stratégies d'optimisation suivantes dans votre code :

- Utilisez `cache()` pour stocker en mémoire les `DataFrame` réutilisés.
- Utilisez `persist(StorageLevel.MEMORY_AND_DISK_SER)` pour les `DataFrame` trop volumineux pour la mémoire seule.
- Utilisez `unpersist()` pour libérer explicitement le cache lorsque ce n'est plus nécessaire.

Question 5.2 – Optimisation des jointures

Objectif : Réduire les coûts de communication réseau (shuffle) lors des jointures.

Utilisez la fonction **`broadcast()`** sur les petites tables (comme `merchants`) lors des jointures avec les tables plus volumineuses.

PARTIE 6 : APPLICATION PRINCIPALE

Question 6.1 – `EcommerceAnalyticsApp`

Objectif : Créer une application principale qui orchestre l'ensemble du pipeline.

Créez un `object` principal qui :

- Initialise une `SparkSession` avec les paramètres de configuration externalisés (voir Partie 7).
- Exécute toutes les phases du pipeline dans l'ordre (ingestion, transformation, analytique).
- Affiche tous les résultats dans la console
- Sauvegarde les résultats finaux aux formats **CSV** et **Parquet** dans un répertoire de sortie.
- Implémente une gestion des erreurs globale pour un comportement robuste.

PARTIE 7 : CONFIGURATION EXTERNALISÉE

Question 7.1 – Fichier `application.conf`

Objectif : Séparer la configuration du code.

Créez un fichier de configuration `application.conf` dans `src/main/resources` pour externaliser certains paramètres.

Exemple de format à adapter.

```
app {

  name = "EcommerceAnalytics"

  data {

    input {

      transactions = "data/transactions.csv"

    }

  }

}
```

Modalités de remise du projet à envoyer par mail **« senendigue@gmail.com » au plutard le Dimanche 2 Aout à 00h.**

Présentation à faire le mardi 2 Aout.

- Archive ZIP avec code source complet
- JAR exécutable généré
- README.md

- Fichiers de configuration (application.conf)