
✅ Résumé haut-niveau (1 phrase)

Le programme **DB2PART2** fusionne deux fichiers de ventes (Europe + Asie), insère les commandes et lignes (orders/items) dans DB2, met à jour la **balance** des clients et le **stock** produit, et génère un fichier de réapprovisionnement si le stock passe sous le seuil.

📁 Entrées / sorties & structures

Fichiers d’entrée (identiques)

PROJET.VENTESEU.DATA et **PROJET.VENTESAS.DATA**
Chaque enregistrement (positions données dans ton énoncé) contient :

- N° COMMANDE (cols 1–3)
- DATE COMMANDE (cols 4–13, format JJ/MM/AAAA)
- N° EMPLOYE (14–15)
- N° CLIENT (16–19)
- N° PRODUIT (20–22)
- PRIX (23–27) — 5 digits, 2 décimales (USD)
- QUANTITE (28–29)
- RESERVE (30–35) (champ réservé)

> Les fichiers sont triés (N° commande, client, employé) et garantissent : pas de doublon produit dans une même commande, un seul employé/date/client par commande, et stock suffisant **dans les jeux de données fournis**.

Fichier STOKS (KSDS)

PROJET.STOCKS.DATA (KSDS) : pour chaque produit :

- CODE-PROD (1–3)
- SEUIL-PROD (4–6) — seuil de réappro
- QTE-COMM (7–8) — quantité à commander si en dessous du seuil

Fichier de sortie

PROJET.REAPPRO.DATA : liste (P_NO, QTE_A_COMMANDER, DATE_COURANTE) pour les produits à réapprovisionner.

Base DB2 (tables manipulées)

- **API7.ORDERS** (O_NO, S_NO, C_NO, O_DATE)
- **API7.ITEMS** (O_NO, P_NO, QUANTITY, PRICE)
- **API7.CUSTOMERS** (mise à jour BALANCE)
- **API7.PRODUCTS** (mise à jour STOCK)

🔁 Flux général du traitement

1. **Initialisation**
    - Open cursor/initialisation (ex. `OPEN CPROD` pour produits).
    - Ouverture des fichiers : F-VEU (EU), F-VAS (AS), F-ST (KSDS), F-REA (sortie réappro).
2. **Lecture / fusion (merge)**

- Lecture synchronisée des deux fichiers de ventes. Le programme lit une ligne de chaque fichier au démarrage.

- **Comparaison des IDs de commande** (`ID-VEU` vs `ID-VAS`).

    - Si ID différent → traiter la plus petite ID en premier (consommer les lignes correspondantes).

    - Si IDs égaux → fusionner les lignes (c.-à-d. traiter ensemble tous les items appartenant à la même commande provenant des deux fichiers).

3. **Traitement d'une commande**

    - Pour chaque nouvelle commande :

        - **Insérer un enregistrement dans `API7.ORDERS`** (n° commande, site/store S_NO (probablement EU/AS), client, date).

        - Pour chaque ligne (item) de commande :

            - **Insérer dans `API7.ITEMS`** (O_NO, P_NO, QUANTITY, PRICE).

            - Calculer montant item `item_amount = price * quantity`.

            - **Décrémenter le stock** en mémoire (et écrire `UPDATE API7.PRODUCTS SET STOCK = :PRO-STOCK WHERE P_NO = :PRO-P-NO`).

            - Cumuler montant total pour la commande (pour la mise à jour de la balance client).

            - Vérifier si le stock recalculé passe sous le seuil (via lecture du fichier Stocks KSDS). Si oui, **accumuler la QTE à commander** pour ce produit dans une zone mémoire (pour écriture ensuite dans F-REA).

    - Après traitement de tous les items de la commande :

        - **Mettre à jour la balance client** (`UPDATE API7.CUSTOMERS SET BALANCE = :CUS-BALANCE WHERE C_NO = :ORD-C-NO`), en ajoutant le montant total de la commande.

        - Commit / gestion transactionnelle selon SQLCODE (voir ci-dessous).

4. **Fin de boucle**
    - Lire la ligne suivante des fichiers de ventes, continuer jusqu'à fin de fichiers.
    - Une fois toutes les commandes traitées : écrire le fichier `PROJET.REAPPRO.DATA` à partir des cumuls collectés (produit, quantité à commander, date courante).
    - Commit global et fermeture fichiers / ressources.

💾 Gestion SQL & erreurs

- Après chaque `INSERT` / `UPDATE`, le programme lit `SQLCODE` et :
    - **SQLCODE = 0** → OK (affiche message "AJOUTE"/"OK").
    - **SQLCODE = -803** (doublon clé unique) → log + appel `LOGDATA` (journalisation); le doublon est reporté mais le traitement continue (le code gère ce cas explicitement).
    - **SQLCODE > 0** → warning : log via `LOGDATA`.
    - **SQLCODE < 0 et != -803** → erreur bloquante : `ROLLBACK`, log, puis appel `TEST-SQLCODE` / `ABEND-PROG` si nécessaire.
- Il y a des `COMMIT`/`ROLLBACK` en cas d'erreur grave et un `COMMIT` global final.
- Le programme récupère l'utilisateur courant (`SELECT USER FROM SYSIBM.SYSDUMMY1`) avant d'appeler `LOGDATA`.

🧮 Calculs principaux

- **Item amount** = `PRICE * QUANTITY` (les prix sont fournis en USD dans les fichiers sources).
- **Balance client** = balance actuelle + somme des montants des items de la commande.
- **Stock produit** = stock actuel − quantité vendue (puis vérification vs seuil).

🔍 Points d'attention / hypothèses (à évoquer à l'oral)

- Les fichiers sources sont triés et garantissent l'absence de doublons produit par commande → le code repose sur cela pour simplifier l'agrégation.
- Transaction : le code effectue un `INSERT ORDERS` puis `INSERT ITEMS` et met à jour `CUSTOMERS` et `PRODUCTS`. Il y a gestion d'erreurs SQL mais il faut clarifier si le commit se fait par commande ou en lot (le programme fait un commit global en fin ; en cas d'erreur critique il effectue rollback).
    - **Remarque:** pour sécurité/concurrence, il peut être préférable de `COMMIT` par commande traitée (si acceptable) afin d'éviter d'avoir une seule grosse transaction.
- Le programme suppose que les produits existent dans la table `PRODUCTS` (il utilise un cursor/lecture sur products/KSDS) ; si produit absent il loggue / affiche une erreur.
- Format des prix (5,2) et du stock est bien pris en charge (NUMVAL, remplacements '.'→',' si nécessaire).
- Date courante pour F-REA est ajoutée — vérifier le format attendu.

🛑 Risques / améliorations recommandées (à mentionner rapidement)

1. **Atomicité par commande** : faire un `COMMIT` par commande pour limiter l'impact d'un rollback global.

2. **Contrôle concurrence** : si le batch tourne sur un environnement partagé, il faut gérer verrouillage / isolation pour updates stock.

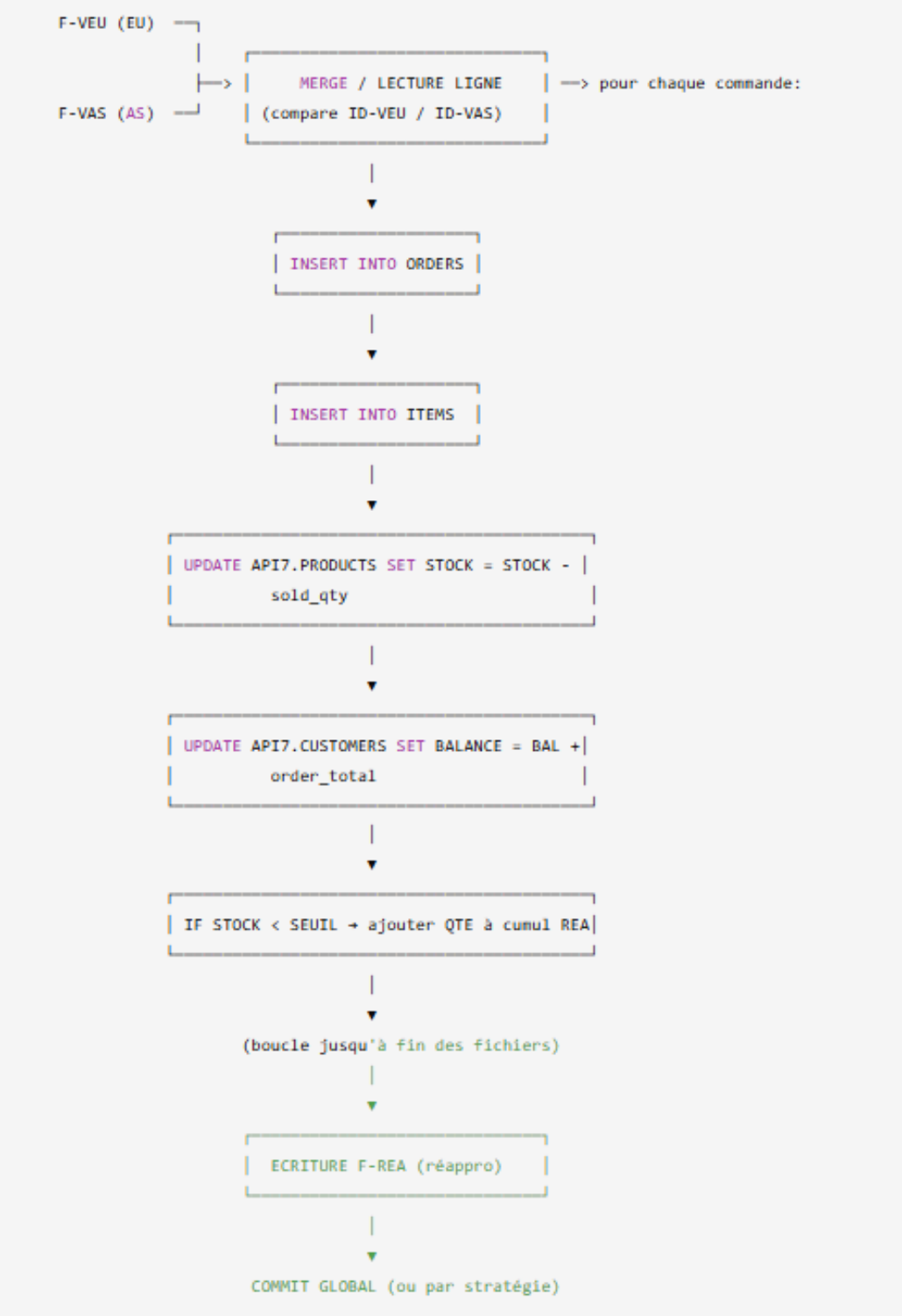
3. **Idempotence** : si un fichier est rechargé par erreur, on risque des doublons — prévoir marqueurs (audit) ou traitement idempotent.

4. **Validation entrée** : meilleure validation des dates / prix / quantités et journalisation centralisée en cas d'anomalies format.

5. **Logging** : enrichir le log avec un identifiant de traitement, timestamp, et niveau (INFO/WARN/ERROR).

6. **Surveillance seuil** : le réappro est créé si stock < seuil ; on pourrait éviter commandes répétées si plusieurs lots traitent le même produit (regrouper QTE-CUMUL-REA).

Schéma d’architecture (flux)



🎤 Proposition de pitch oral (30–40 secondes)

> « Le programme DB2PART2 fusionne les ventes Europe et Asie, puis pour chaque commande il crée l'en-tête dans la table ORDERS et les lignes dans ITEMS. Il calcule le montant des articles, met à jour la balance client et décrémente le stock produit dans la table PRODUCTS. Si après vente le stock d'un produit passe sous son seuil, on prépare un fichier de réapprovisionnement (PROJET.REAPPRO.DATA) avec la quantité à commander et la date. Le code gère les erreurs SQL (doublons, warnings, erreurs bloquantes) et journalise les anomalies. »