

**Suivi des commandes des
revendeurs**

Distributech

Dossier technique

Nathalie BEDIEE

Table des matières

1. Introduction	2
1.1. Contexte et rappel des objectifs	2
1.2. Périmètre technique du projet	2
2. Spécifications fonctionnelles	3
2.1. Entités principales et règles de gestion	3
2.2. Cas d'usage (utilisation, exploitation, maintenance)	4
3. Architecture et modélisation	4
3.1. Schéma global des flux	4
3.2. Composants logiciels et contraintes	5
3.3. Modélisation des données (MCD, MLD)	6
3.3.1. MCD – modèle conceptuel	6
3.3.2. MLD – modèle logique	7
3.3.3. Schéma relationnel	8
4. Spécifications techniques du pipeline ETL	8
4.1. Processus principal	8
4.2. Extract – lecture et validation des sources (CSV, SQLite)	10
4.3. Transform – contrôles, normalisation, gestion erreurs/doublons	11
4.4. Load – insertion en base	13
4.5. Gestion du stock (vue et export)	14
4.6. Tableau de bord	15
5. Plan de test	18
5.1. Objectif et périmètre	18
5.2. Jeux de données de test et cas de tests	18
5.3. Critères de réussite et acceptation	18
6. Livrables attendus	18
6.1. Scripts et base relationnelle	18
6.2. Exports CSV et reporting	18
6.3. Documentation technique	18

1. Introduction

1.1. Contexte et rappel des objectifs

Distributech, grossiste en équipements électroniques, reçoit les **commandes** de ses **revendeurs régionaux** au format **CSV** et gère les **mouvements de stock** dans une base **SQLite**.

Le besoin est de **centraliser** et **fiabiliser** ces données dans une **base relationnelle unique**, alimentée par un processus **ETL**.

Les objectifs principaux sont :

- **Centraliser toutes les données** dans une base unique :
 - **Commandes** envoyées par les revendeurs (CSV),
 - **Produits** du catalogue (SQLite),
 - **Revendeurs** et leurs **régions** (SQLite),
 - **Mouvements de stock** (réapprovisionnements, SQLite).
- **Automatiser l'intégration** via un pipeline **ETL** (*Extract => Transform => Load*).
- **Calculer et exporter l'état des stocks** à chaque cycle.
- Fournir aux équipes **logistique** et **commerciale** un **accès simple** aux informations consolidées (stock, commandes, chiffre d'affaires).

1.2. Périmètre technique du projet

Le présent **dossier technique** décrit la **solution mise en place** pour atteindre ces objectifs.

Il comprend :

- l'**architecture cible** (sources, ETL, base SQL, exports, tableau de bord),
- la **modélisation de la base relationnelle** (MCD, MLD),
- les **spécifications techniques du pipeline ETL**,
- la **conception du tableau de bord**,
- le **plan de test**,
- et les **livrables attendus**.

2. Spécifications fonctionnelles

2.1. Entités principales et règles de gestion

- **Région** : zone géographique. Chaque revendeur appartient à une seule région.
- **Revendeur** : client régional, identifié par un ID, associé à une région.
- **Produit** : article du catalogue, identifié par un code, un nom et un coût unitaire.
- **Commande** : ensemble d'articles demandés par un revendeur, identifié par un numéro unique et une date.
- **Ligne de commande** : détail d'une commande (produit, quantité, prix unitaire).
- **Mouvement de production** : entrée de stock enregistrée dans SQLite (réassort).
- **Stock** : quantité disponible d'un produit, calculée à une date donnée :

$$\text{Stock}(\text{produit}) = \sum(\text{entrées production du produit}) - \sum(\text{sorties cmds du produit})$$

Règles de gestion clés :

- Chaque **revendeur** appartient à une seule **région** (référentiel SQLite).
- L'identifiant de **région** peut être absent dans les commandes CSV, mais il est systématiquement **retrouvé via le référentiel revendeur**.
- Chaque **commande** est associée à un seul revendeur et possède un **numéro unique**.
- Chaque **ligne de commande** est liée à un produit et contient la **quantité** et le **prix unitaire** au moment de la commande.
- Les identifiants (**revendeurs, produits**) sont uniques et non nuls.
- Les **quantités** et **coûts** sont strictement positifs.
- Les **dates** doivent être valides et ne pas être futures (dans la production).
- Les données incohérentes (valeurs négatives, références inconnues, doublons) sont **rejetées** et consignées dans les **logs**.

2.2. Cas d'usage (utilisation, exploitation, maintenance)

➤ En utilisation (équipes logistiques et commerciales)

- Consulter le **stock courant** et le **stock à une date donnée**.
- Consulter les **commandes** par période, produit ou région.
- Calculer le **chiffre d'affaires** par région ou global.
- Exporter les résultats au format **CSV** pour exploitation dans un tableur (ex : Excel).

➤ En exploitation (personnel technique)

- Déclencher et superviser l'**ETL**.
- Contrôler les **logs d'exécution**.
- Gérer les anomalies (fichiers manquants, données incohérentes).

➤ En maintenance (développeur / DBA)

- Assurer la **maintenance corrective** (erreurs, incidents).
- Assurer la **maintenance évolutive** (nouvelles règles, nouvelles sources).
- Mettre à jour la **documentation technique**.

3. Architecture et modélisation

3.1. Schéma global des flux

Le système repose sur deux **sources** de données :

- les **commandes** envoyées en **fichiers CSV** par les revendeurs,
- la base **SQLite** contenant les **référentiels** (régions, revendeurs, produits) et les **mouvements de production** (entrées de stock).

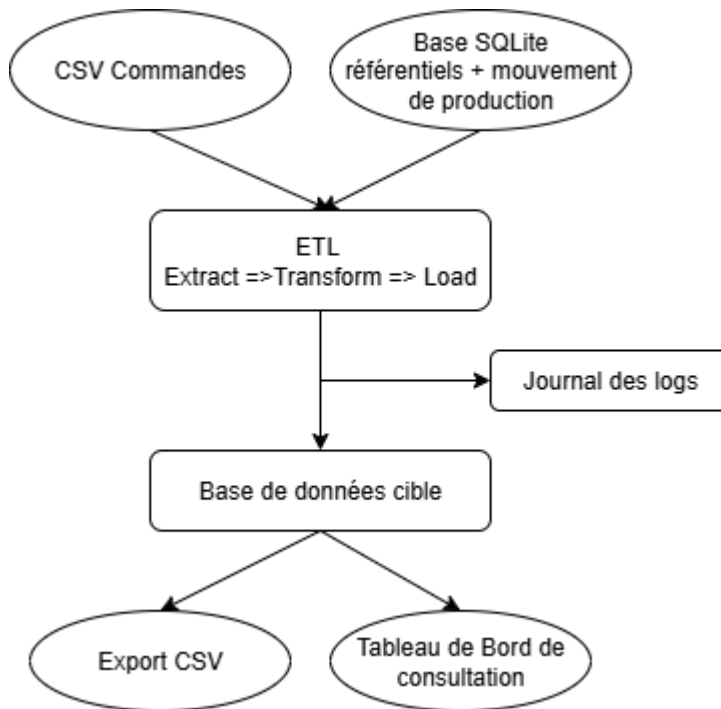
Ces sources alimentent un **pipeline ETL** en trois étapes (**Extract, Transform, Load**) développé en **Python**.

Tout incident (colonnes manquantes, incohérences, doublons) est enregistré dans un **journal des logs** pour assurer la traçabilité.

Les données sont ensuite centralisées dans une **base relationnelle SQL**. Les résultats sont accessibles via :

- des **exports CSV** (état des stocks, reporting),
- un **tableau de bord console** pour consultation simple.

Le schéma ci-dessous illustre le flux :



3.2. Composants logiciels et contraintes

- **Langage** : Python 3.x.
- **Bibliothèques principales** :
 - *pandas* pour le traitement de données,
 - *mysql.connector* pour la connexion à la base relationnelle,
 - *sqlite3* pour la lecture base SQLite.
- **Base relationnelle cible** : MySQL.
- **Compatibilité OS** : Windows et Linux.
- **Exécution** : mode **batch** (pipeline complet sans interaction utilisateur).
- **Organisation des répertoires de données**
 - *data/in* : dépôt des fichiers CSV entrants
 - *data/treated* : fichiers déplacés après traitement
 - *data/log* : journaux d'exécution
 - *data/stock* : exports CSV générés (état des stocks)

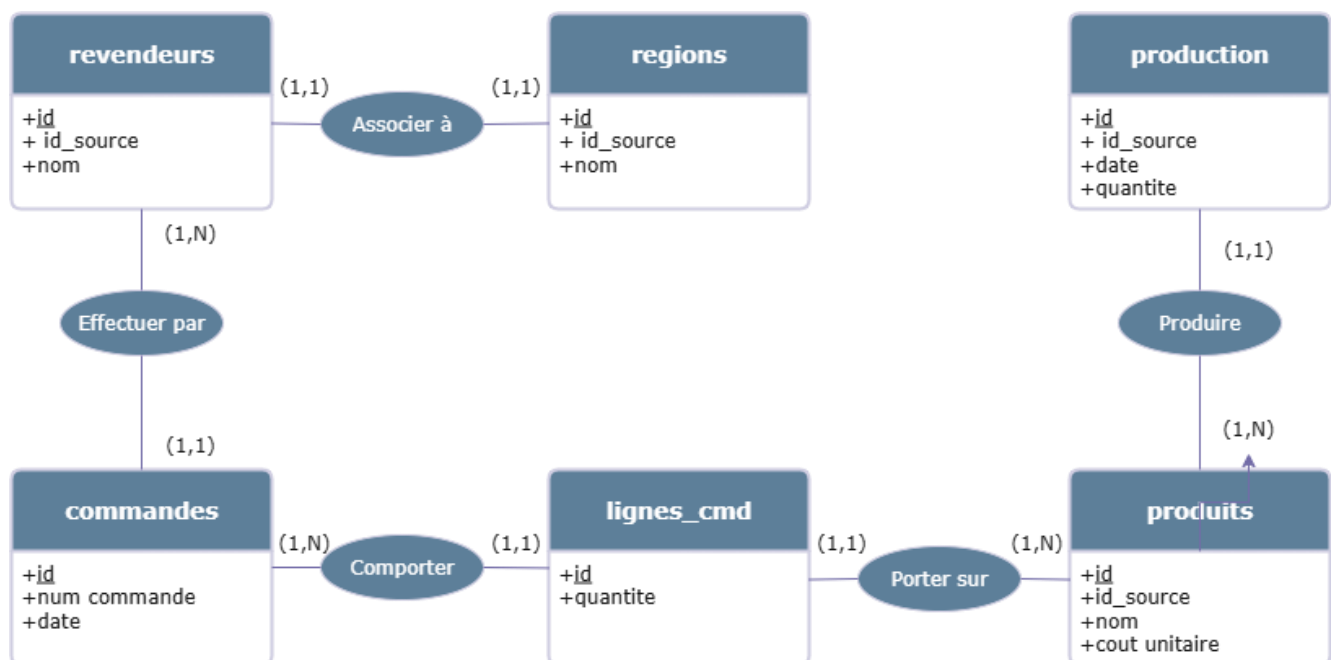
3.3. Modélisation des données (MCD, MLD)

La modélisation des données repose sur la méthode **Merise** avec deux représentations complémentaires :

1. **Modèle Conceptuel de Données (MCD)** : permet d'identifier les entités métier, leurs propriétés et les relations entre elles, sans contrainte technique.
2. **Modèle Logique de Données (MLD)** : traduit le MCD en tables et colonnes adaptées à une base relationnelle.
3. **Schéma relationnel** : illustre visuellement la structure finale de la base avec les clés primaires, étrangères et les associations.

3.3.1. MCD – modèle conceptuel

Le schéma ci-dessous présente les principales entités (revendeurs, commandes, produits, etc.) et leurs relations métier.



3.3.2. MLD – modèle logique

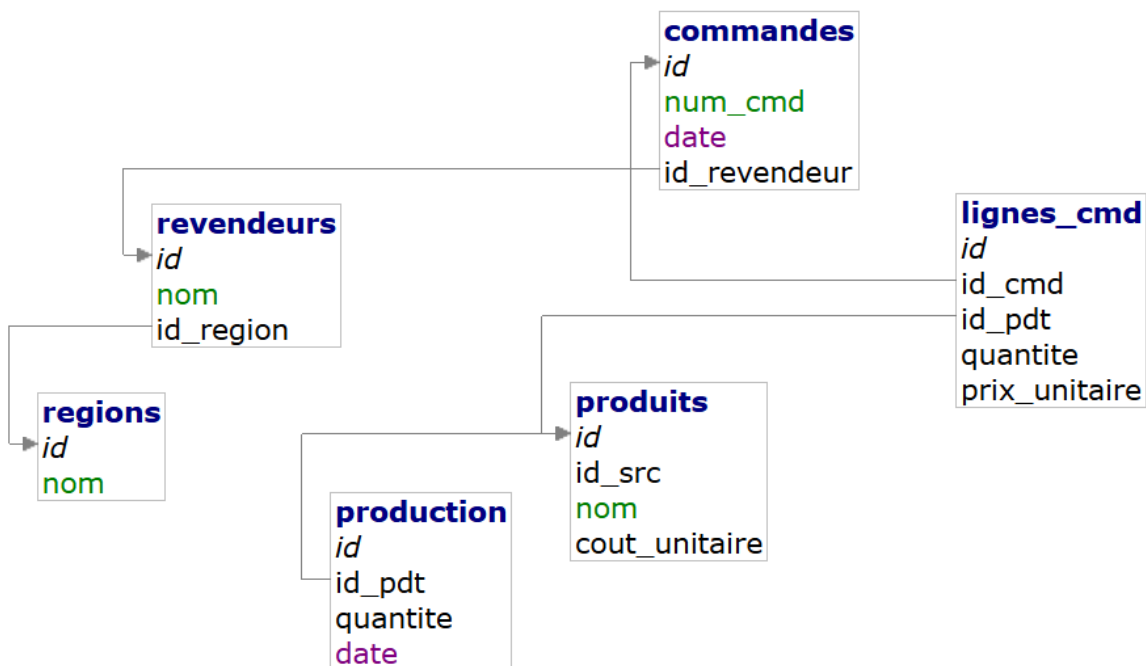
À partir du MCD, le modèle logique définit les tables relationnelles, leurs attributs, et les contraintes (PK, FK, unicité, intégrité).

- **regions**
 - **id** : INT, PK
 - **nom** : VARCHAR(50) NOT NULL
- **Revendeurs**
 - **id** : INT, PK
 - **nom** : VARCHAR(50) NOT NULL
 - **#id_region** : INT, FK → regions.id, NOT NULL
- **Produits**
 - **id** : INT, PK, AUTO_INCREMENT
 - **id_src** : INT NOT NULL
 - **nom** : VARCHAR(50) NOT NULL
 - **cout_unitaire** : FLOAT NOT NULL
- **Production**
 - **id** : INT, PK
 - **#id_pdt** : INT, FK → produits.id, NOT NULL
 - **quantite** : INT NOT NULL
 - **date** : DATE NOT NULL
- **commandes**
 - **id** : INT, PK, AUTO_INCREMENT
 - **num_cmd** : VARCHAR(30) NOT NULL
 - **date** : DATE NOT NULL
 - **#id_revendeur** : INT, FK → revendeurs.id, NOT NULL
- **lignes_cmd**
 - **id** : INT, PK, AUTO_INCREMENT
 - **#id_cmd** : INT, FK → commandes.id, NOT NULL
 - **#id_pdt** : INT, FK → produits.id, NOT NULL
 - **quantite** : INT NOT NULL
 - **prix_unitaire** : FLOAT NOT NULL

3.3.3. Schéma relationnel

Le schéma relationnel ci-dessous a été généré à partir de l'outil **Adminer**, connecté à la base cible **MySQL**.

Il représente la structure finale réellement implémentée : les tables avec leurs colonnes principales, ainsi que les clés primaires et étrangères qui assurent l'intégrité des liens entre entités.



4. Spécifications techniques du pipeline ETL

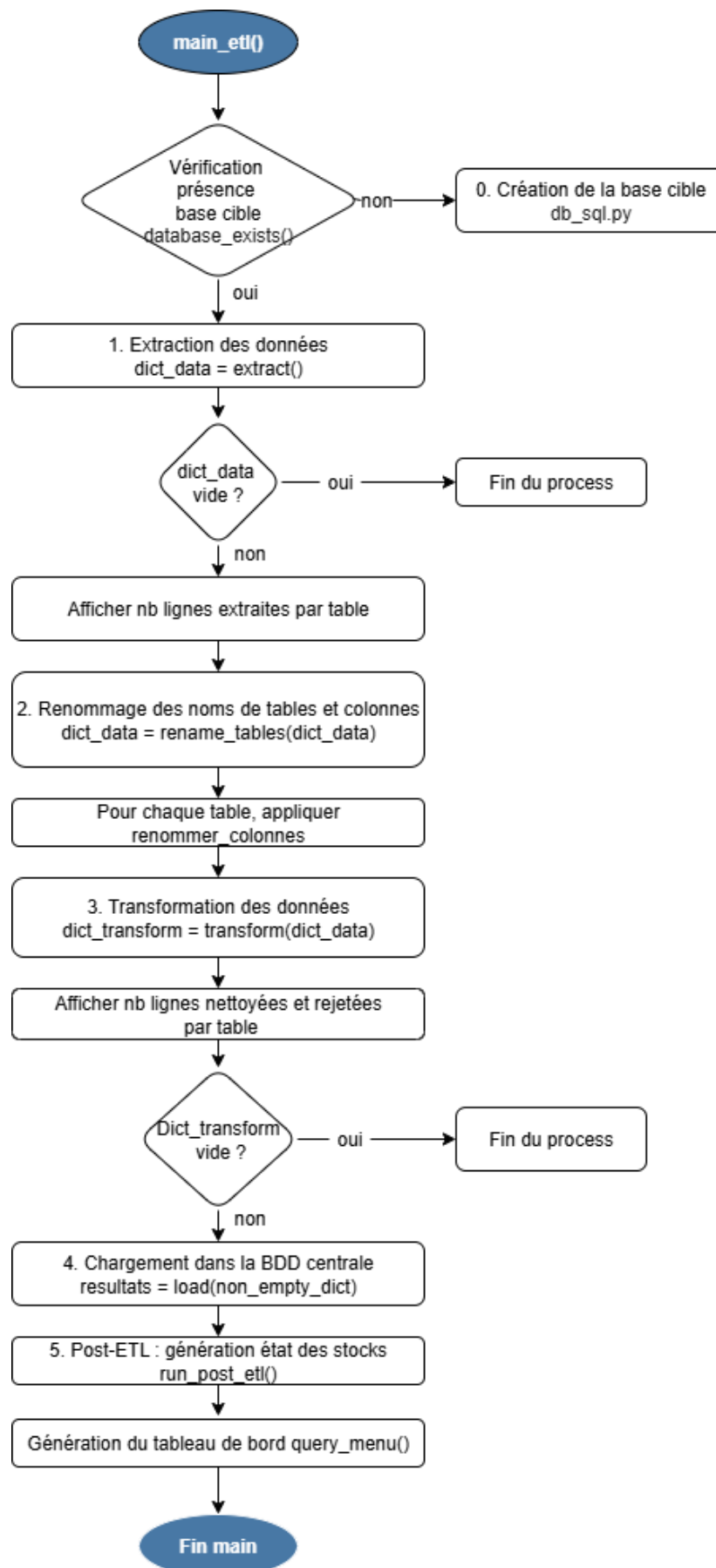
4.1. Processus principal

Le fichier **main_etl.py** orchestre l'ensemble du processus ETL, depuis la vérification de la base cible jusqu'à la génération de l'état de stock et l'accès au tableau de bord.

Chaque étape est exécutée en séquence, avec journalisation des événements ([INFO], [ALERTE], [ERREUR], / !\) dans le log.

Le pipeline fonctionne en mode batch, sans interaction utilisateur, jusqu'au lancement du tableau de bord.

Le diagramme suivant illustre les principales étapes :



En complément, chaque fichier Python joue un rôle spécifique dans le processus.
Le tableau ci-dessous synthétise ces responsabilités :

Fichier	Rôle principal
main.py	Pilotage global du processus ETL
db_sql.py	Création de la base cible (si absente)
extract.py	Extraction des données depuis CSV et SQLite
transform.py	Nettoyage et validation des données
load.py	Chargement dans la base centrale
post_etl.py	Génération de l'état des stocks après ETL
commun.py	Fonctions partagées (renommage, logs, vérification base, etc.)
query_menu.py	Génère le tableau de Bord (interrogations SQL sur la base cible, affichage / reporting)

4.2. Extract – lecture et validation des sources (CSV, SQLite)

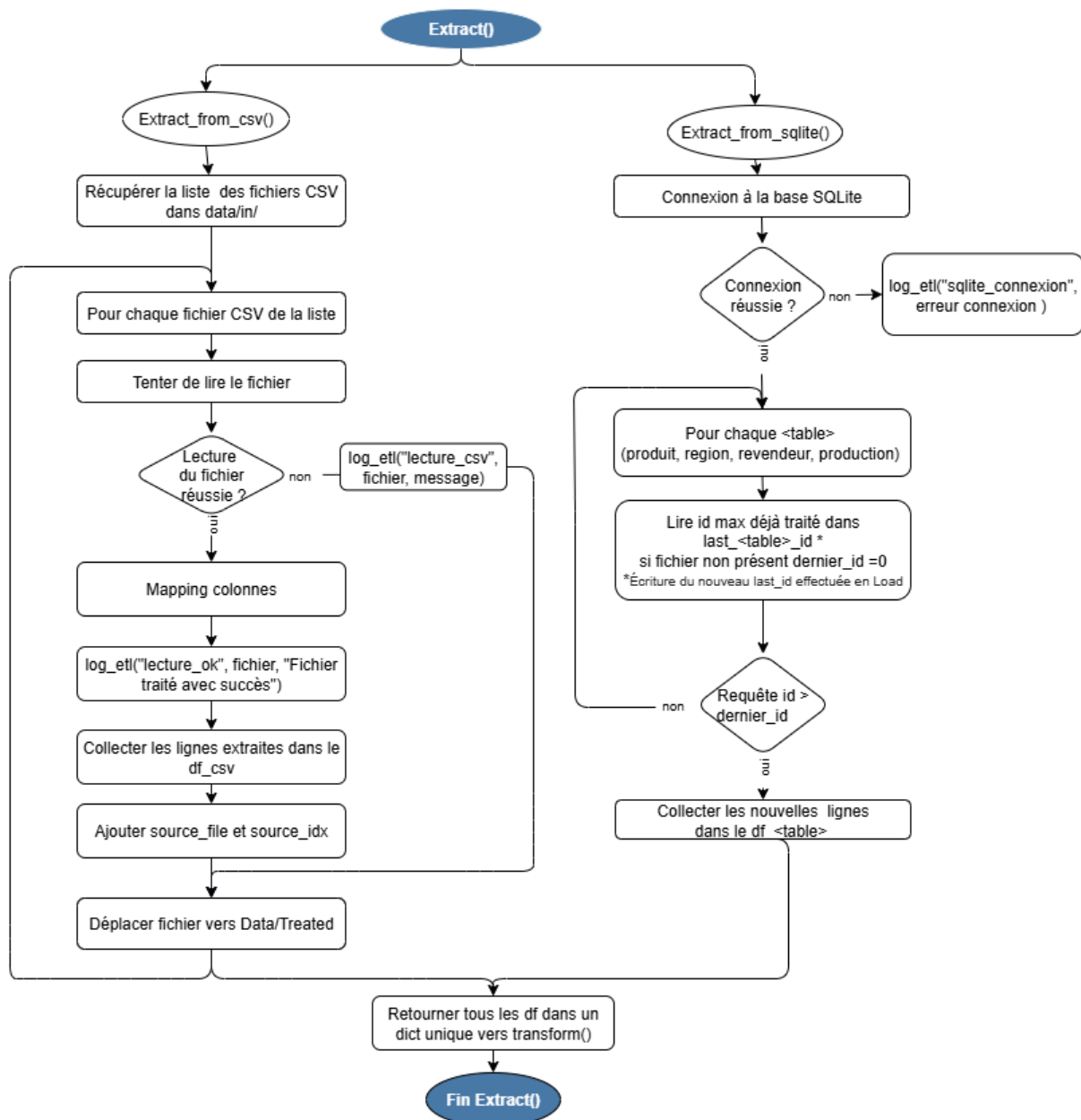
L'extraction est assurée par le script **extract.py**.

Deux sources sont prises en compte :

- les **fichiers CSV** contenant les commandes,
- la **base SQLite** fournissant les données de référence (produits, régions, revendeurs, production).

Pour chaque source, une gestion d'erreur (connexion, structure, lecture) est prévue : les incidents sont loggés et les fichiers ou lignes KO sont exclus du traitement.

Le schéma ci-dessous illustre le fonctionnement de ce module, en distinguant la branche CSV et la branche SQLite.



4.3. Transform – contrôles, normalisation, gestion erreurs/doublons

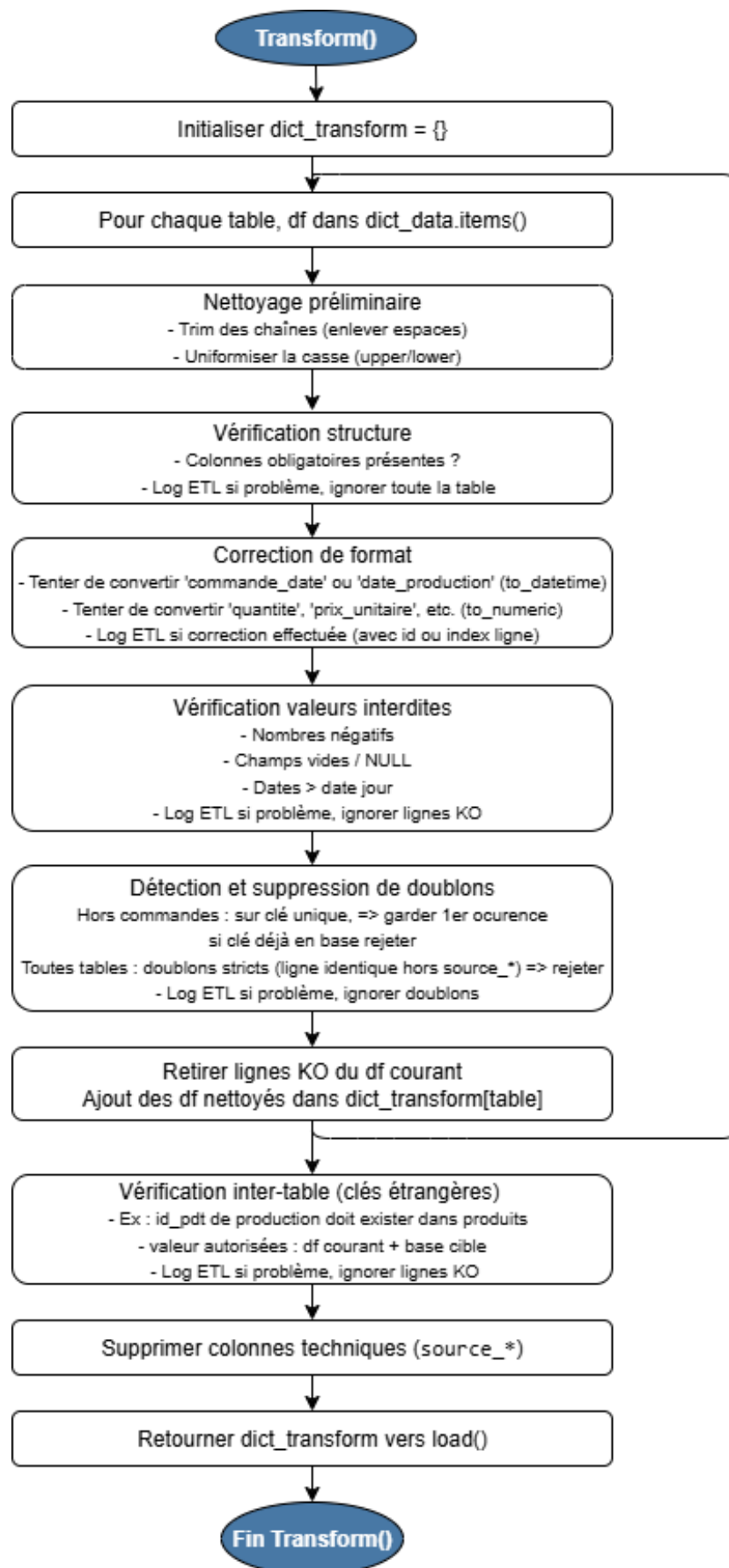
Le module **transform.py** applique les règles de qualité définies :

- normalisation des formats (dates, numériques, chaînes),
- suppression des doublons,
- gestion des incohérences détectées.

Chaque rejet est tracé, assurant la transparence des pertes de données.

À l'issue de cette étape, les tables sont prêtes pour une intégration fiable en base centrale.

Le processus suivi est représenté dans le diagramme ci-dessous.

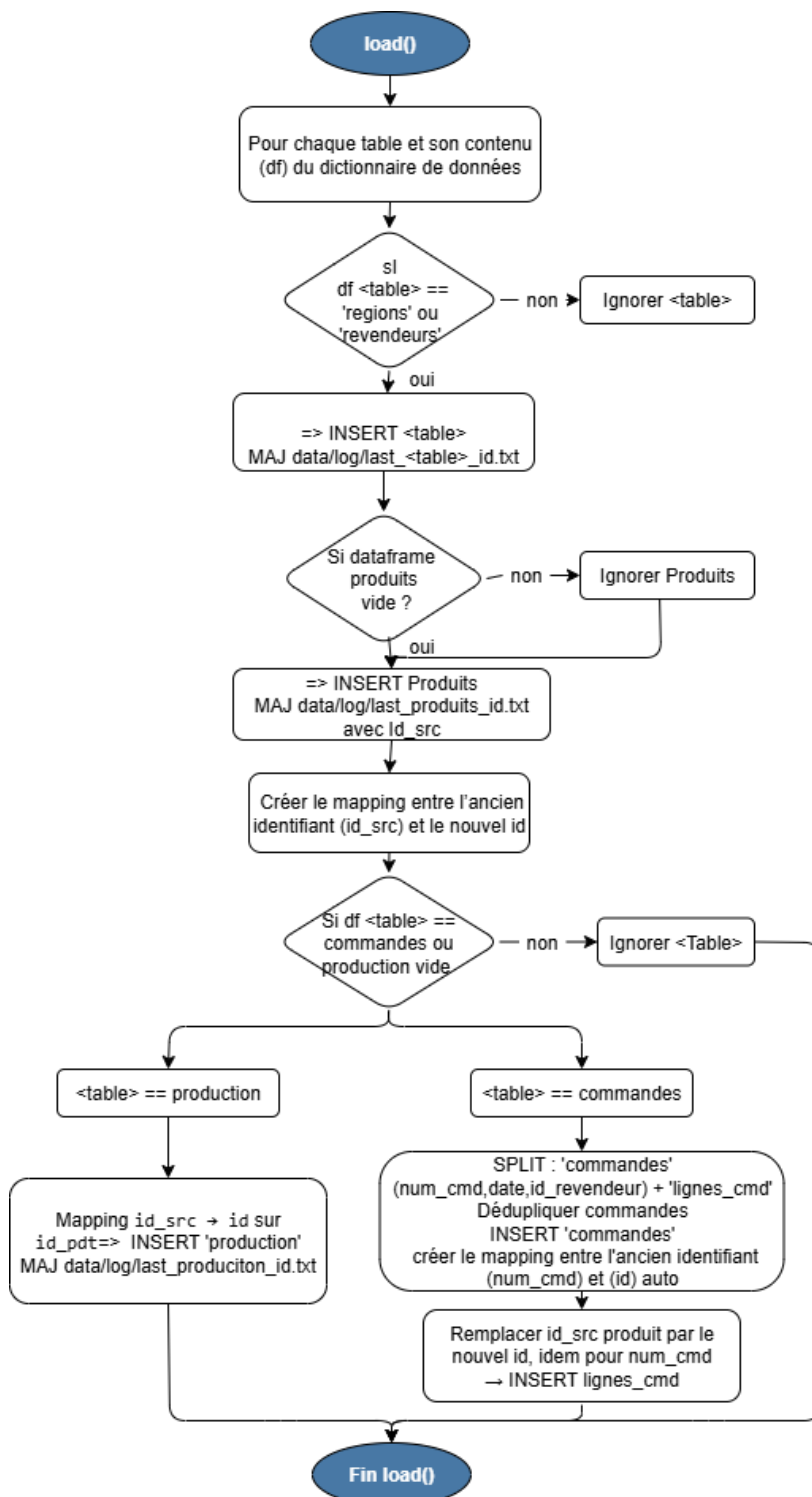


4.4. Load – insertion en base

Le module **load.py** prend en charge le chargement des données transformées dans la base MySQL.

Les dépendances entre tables et la correspondance des clés sont gérées automatiquement.

Un suivi est enregistré : nombre de lignes insérées, anomalies détectées et erreurs critiques bloquantes.



4.5. Gestion du stock (vue et export)

La gestion du stock et des indicateurs associés s'appuie sur trois **vues SQL** créées en fin de pipeline ETL par **post_etl.py**. Elles centralisent les calculs et simplifient l'export.

- Vue **v_stock** : État courant du stock.

Principe :

$$\text{Stock}(\text{produit}) = \sum(\text{entrées production du produit}) - \sum(\text{sorties cmds du produit})$$

```
CREATE OR REPLACE VIEW v_stock AS
SELECT
  p.id      AS produit_id,
  p.nom     AS produit_nom,
  COALESCE(SUM(prod.quantite), 0) AS total_entrees,
  COALESCE(SUM(lcmd.quantite), 0) AS total_sorties,
  COALESCE(SUM(prod.quantite), 0) - COALESCE(SUM(lcmd.quantite), 0) AS stock_courant
FROM produits p
LEFT JOIN production prod ON prod.id_pdt = p.id
LEFT JOIN lignes_cmd lcmd ON lcmd.id_pdt = p.id
GROUP BY p.id, p.nom;
```

Usage : détection des stocks négatifs et base de l'export CSV.

- Vue **v_cmds_par_region** : Commandes agrégées par région et produit.

Principe :

$$\text{Cmds}(\text{region}, \text{produit}) = \sum(\text{quantités commandées du produit par la région})$$

```
CREATE OR REPLACE VIEW v_cmds_par_region AS
SELECT
  r.nom AS region_nom,
  p.id  AS produit_id,
  p.nom AS produit_nom,
  SUM(lc.quantite) AS total_commandes
FROM lignes_cmd lc
JOIN commandes c ON lc.id_cmd = c.id
JOIN revendeurs v ON c.id_revendeur = v.id
JOIN regions r ON v.id_region = r.id
JOIN produits p ON lc.id_pdt = p.id
GROUP BY r.nom, p.id, p.nom
ORDER BY r.nom, p.id;
```

Usage : analyse de la répartition des ventes par région.

- Vue **v_chiffre_affaires_par_region** : Chiffre d'affaires consolidé.

Principe :

- $CA(region) = \sum(quantité \times prix\ unitaire)$
- $Nb_commandes(region) = \sum(commandes\ distinctes)$

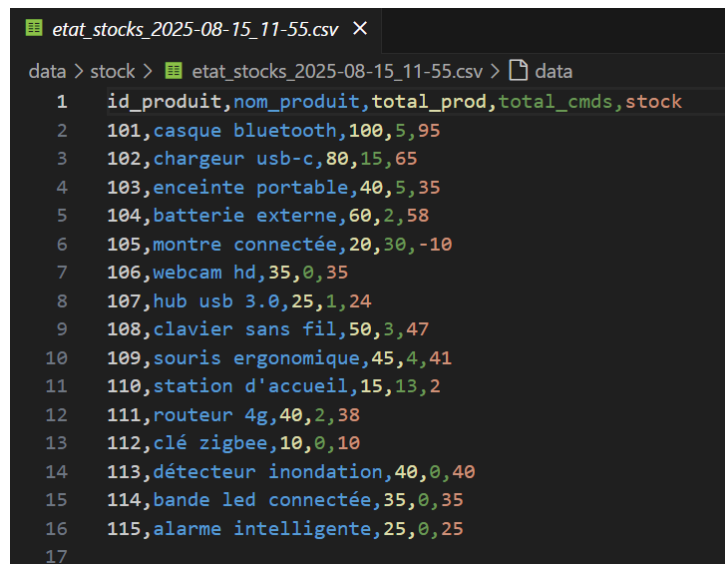
```
CREATE OR REPLACE VIEW v_chiffre_affaires_par_region AS
SELECT
    r.id    AS region_id,
    r.nom   AS region_nom,
    ROUND(SUM(lc.quantite * lc.prix_unitaire), 2) AS chiffre_affaires,
    COUNT(DISTINCT c.num_cmd) AS nb_commandes
FROM lignes_cmd lc
JOIN commandes c    ON lc.id_cmd      = c.id
JOIN revendeurs re  ON c.id_revendeur = re.id
JOIN regions r      ON re.id_region   = r.id
GROUP BY r.id, r.nom
ORDER BY r.nom;
```

- **Export CSV du stock**

En fin de pipeline, post_etl.py lit v_stock, signale les stocks négatifs, puis génère un fichier dans data/stock/.

- Nom : etat_stocks_YYYY-MM-DD_HH-MM.csv
- Colonnes : produit_id,produit_nom,total_entrees,total_sorties,stock_courant
- Format : UTF-8, séparateur ,
- Exploitation : ouverture dans un tableur.

Exemple de fichier :



```
etat_stocks_2025-08-15_11-55.csv X
data > stock > etat_stocks_2025-08-15_11-55.csv > data
1 id_produit,nom_produit,total_prod,total_cmds,stock
2 101,casque bluetooth,100,5,95
3 102,chargeur usb-c,80,15,65
4 103,enceinte portable,40,5,35
5 104,batterie externe,60,2,58
6 105,montre connectée,20,30,-10
7 106,webcam hd,35,0,35
8 107,hub usb 3.0,25,1,24
9 108,clavier sans fil,50,3,47
10 109,souris ergonomique,45,4,41
11 110,station d'accueil,15,13,2
12 111,routeur 4g,40,2,38
13 112,clé zigbee,10,0,10
14 113,détecteur inondation,40,0,40
15 114,bande led connectée,35,0,35
16 115,alarme intelligente,25,0,25
17
```

4.6. Tableau de bord

Le module query_menu.py fournit un **tableau de bord en ligne de commande** destiné aux équipes logistiques et commerciales. Il permet d'interroger directement la base

relationnelle, de consulter les principaux indicateurs et d'exporter les résultats au format CSV.

Fonctionnalités disponibles

Le tableau de bord propose six interrogations réparties en trois volets :

- **Stock global**
 1. État actuel (lecture de la vue v_stock).
 2. État à une date donnée (requête avec borne temporelle sur production.date et commandes.date).
- **Commandes par région**
 3. Totaux globaux par région et produit (vue v_cmds_par_region).
 4. Totaux limités à une période définie par l'utilisateur.
- **Chiffre d'affaires par région**
 5. Totaux globaux par région (vue v_chiffre_affaires_par_region).
 6. Totaux limités à une période définie par l'utilisateur.

L'écran principal se présente ainsi :

```
***** TABLEAU DE BORD *****

[ STOCK GLOBAL ]
  1. Actuel
  2. A une date précise

[ COMMANDES PAR REGION ]
  3. Globales
  4. Sur une période

[ CHIFFRE D'AFFAIRES PAR REGION ]
  5. Globales
  6. Sur une période

0. Quitter

Votre choix : █
```

Affichage et export

Les résultats sont affichés sous forme de tableau dans la console. L'utilisateur peut ensuite confirmer l'export en CSV :

- dossier de destination : data/stock/
- encodage UTF-8, séparateur ,
- nommage horodaté (stock_global_actuel_2025-08-18_21-30.csv)

Exemples de résultats

➤ Stock global (actuel)

```
***** STOCK GLOBAL (actuel) *****

produit_id      produit_nom  total_entrees  total_sorties  stock_courant
1      casque bluetooth      100           10           90
2      chargeur usb-c        240           15          225
3      enceinte portable      80            5           75
4      batterie externe      60            2           58
5      montre connectée      60           30           30
6      webcam hd             35            0           35
7      hub usb 3.0            25            1           24
8      clavier sans fil       50            6           44
9      souris ergonomique     90            4           86
10     station d'accueil      30           13           17
11     routeur 4g             40            2           38
12     clé zigbee             10            0           10
13     détecteur inondation    40            0           40
14     bande led connectée    35            0           35
15     alarme intelligente    25            0           25

15 ligne(s).
Exporter en CSV ? (o/n) :
```

➤ Chiffre d'affaires par région (à partir d'une date)

```
Votre choix : 4
Date début (YYYY-MM-DD, vide = commande jusqu'à date fin) : 2025-07-15
Date fin (YYYY-MM-DD, vide = commande à partir de date début) :

***** COMMANDES PAR REGION (à partir du 2025-07-15) *****

region  id_produit      produit  total_commandes
auvergne-rhône-alpes  102  chargeur usb-c      3
bretagne              110  station d'accueil   3
île-de-france         107  hub usb 3.0         1
occitanie             104  batterie externe    2
occitanie             109  souris ergonomique  2

5 ligne(s).
Exporter en CSV ? (o/n) :
```

➤ Chiffre d'affaires par région (sur une période)

```
Votre choix : 6
Date début (YYYY-MM-DD, vide = commande jusqu'à date fin) : 2025-07-01
Date fin (YYYY-MM-DD, vide = commande à partir de date début) : 2025-08-01

***** CHIFFRE D'AFFAIRES PAR REGION (du 2025-07-01 au 2025-08-01) *****

region  total_chiffre_affaires  nb_commandes
auvergne-rhône-alpes      40.5           1
bretagne                  274.5           3
île-de-france             1628.6           6
occitanie                  184.8           3

4 ligne(s).
Exporter en CSV ? (o/n) :
```

Gestion des dates et contrôles

- Les saisies de dates sont vérifiées au format YYYY-MM-DD.
- En cas d'inversion des bornes, l'ordre est corrigé automatiquement.
- Si une vue n'existe pas encore (ETL non exécuté), un message explicite invite à lancer le processus.

5. Plan de test

- 5.1. Objectif et périmètre**
- 5.2. Jeux de données de test et cas de tests**
- 5.3. Critères de réussite et acceptation**

6. Livrables attendus

- 6.1. Scripts et base relationnelle**
- 6.2. Exports CSV et reporting**
- 6.3. Documentation technique**