

# SmartShop 360

Plateforme Data & Agent IA E-commerce

## Rapport Technique

Stratégie MDM & Dictionnaire de Données

Projet	SmartShop 360
Domaine	E-commerce B2C (Décoration & Cadeaux)
Stack	PostgreSQL 16, Python 3.11, SQLAlchemy, scikit-learn
Date	Février 2026

## Table des matières

---

## Contexte et problématique

SmartShop 360 agrège deux sources de données hétérogènes :

- **Source ERP** – fichier `online_retail_II.csv` (transactions de vente : Invoice, StockCode, Description, Quantity, UnitPrice, CustomerID, Country).
- **Source Avis** – fichier JSON `labeledReview.datasetFix.json` (avis clients crawlés sur une marketplace : ProductName, ReviewText, Sentiment, Note).

### Problème central

Les deux sources décrivent les mêmes produits physiques, mais sans clé commune : l'ERP identifie un produit par son `StockCode` (ex. 85123A), tandis que la source avis utilise un nom textuel libre (ex. “*White Hanging Heart Tlight Holder*”).

L'enjeu du **Master Data Management (MDM)** est de créer une table de correspondance (*Golden Record*) reliant chaque article ERP à ses avis clients, afin de calculer des KPI croisés (CA × Note × Réputation).

## Stratégie MDM – Choix et justifications

### Architecture générale du pipeline MDM

Source / Étape	Transformation	Sortie
Source ERP (StockCode)	Fuzzy Match TF-IDF	product_mapping (Golden Record)
Source Avis (nom textuel)	Fuzzy Match TF-IDF	product_mapping (Golden Record)
product_mapping	Chargement ETL	Entrepôt PostgreSQL
Source ERP	Chargement direct	sales_facts (entrepôt)
Source Avis	Chargement direct	review_facts (entrepôt)

## Niveaux de matching – hiérarchie de priorité

#	Stratégie	Description	Fiabilité	Contexte
1	Hard Match (EAN/GTIN)	Code-barres universel commun aux deux sources	100 %	Production
2	Fuzzy Match (TF-IDF + cosinus)	Vectorisation par n-grammes de caractères ; seuil cosinus $\geq 0,35$	70–90 %	Implémenté
3	Semantic Match (Sentence-BERT)	Embeddings 384 dimensions + <code>pgvector</code> ; robuste aux noms très différents	60–85 %	Implémenté (RAG)
4	Rank-based (POC)	Association rang par rang (Top-50 ERP $\leftrightarrow$ Top-50 avis)	Artificielle	POC uniquement

## Implémentation POC – Fuzzy Match TF-IDF

### Algorithme

L'implémentation se trouve dans `src/etl/mdm_mapping.py`. Le vecteur TF-IDF est construit en mode **character n-grams** (`char_wb`,  $n \in [2, 4]$ ) pour être robuste aux :

- fautes d'orthographe et abréviations (ex. “*Tlight*” vs “*T-Light*”);
- différences de casse et d'espacement ;
- noms partiels (“*Cupid Hearts Coat*” vs “*Cream Cupid Hearts Coat Hanger*”).

Extrait – `fuzzy_match_tfidf()` dans `mdm_mapping.py`

```
vectorizer = TfidfVectorizer(
    analyzer      = "char_wb",    # n-grammes de caracteres
    ngram_range   = (2, 4),
    lowercase     = True,
)
tfidf_matrix = vectorizer.fit_transform(erp_names + review_names)

# Matrice de similarite cosinus (n_erp x n_review)
sim_matrix = cosine_similarity(erp_vec, rev_vec)

# Meilleur match si score >= seuil
for erp_idx in range(n_erp):
    best_rev_idx = np.argmax(sim_matrix[erp_idx])
    best_score   = sim_matrix[erp_idx, best_rev_idx]
    if best_score >= threshold:    # threshold = 0.35
        matches[erp_idx] = (best_rev_idx, best_score)
```

*Paramètres retenus*

Paramètre	Valeur	Justification
analyzer	char_wb	Robustesse aux fautes ; pas de dépendance lexicale
ngram_range	(2, 4)	Capte les di-, tri- et quad-grammes de caractères
threshold	0,35	Équilibre précision/rappel sur données UK retail
SAMPLE_SIZE	30 000	Représentatif ; limite le temps de vectorisation

*Résultat – table `product_mapping`*

La fonction `build_product_mapping()` produit un *Golden Record* par produit ERP, avec les colonnes suivantes :

Colonne	Type	Signification
MappingID	SERIAL	Clé surrogate auto-incrémentée
ERP_StockCode	VARCHAR	Code article source ERP
ERP_ProductName	VARCHAR	Libellé ERP brut
Review_ProductCode	VARCHAR	Code article côté avis (REV_001...)
Review_ProductName	VARCHAR	Libellé avis brut
Category	VARCHAR	Catégorie inférée (règles mots-clés)
GoldenRecordName	VARCHAR	Nom unifié retenu (Golden Record)
MatchScore	FLOAT	Score cosinus TF-IDF $\in [0, 1]$
MatchStrategy	VARCHAR	tfidf ou rank

**Stratégie production – Comment nous aurions procédé en contexte réel**

Dans un contexte réel, le pipeline MDM suivrait une cascade de quatre niveaux de confiance décroissante.

*Niveau 1 – Hard Match par code EAN/GTIN (déterministe)***Principe**

Chaque produit physique porte un code-barres EAN-13 ou GTIN-14 unique au monde. Si l'ERP et la plateforme d'avis exposent cette valeur, la réconciliation est déterministe et ne nécessite aucun algorithme.

**Condition préalable** : l'ERP doit stocker le GTIN dans une colonne dédiée, et le crawler d'avis doit extraire le GTIN depuis la fiche produit marketplace.

**Implémentation :**

```
-- Jointure déterministe sur GTIN
UPDATE product_mapping pm
SET   "Review_ProductCode" = r."ProductID",
      "MatchStrategy"      = 'ean_hard',
      "MatchScore"         = 1.0
FROM   review_source r
WHERE  pm."ERP_GTIN" = r."GTIN"
AND    pm."MatchStrategy" IS NULL;
```

**Fiabilité** : 100 %. Aucun faux positif possible.

*Niveau 2 – Fuzzy Match sur les noms (Levenshtein + TF-IDF)*

Pour les produits sans GTIN commun, on applique deux algorithmes en parallèle et on combine leurs scores :

Algorithme	Avantage	Limite
Distance de Levenshtein ( <code>thefuzz</code> )	Simple, intuitif, seuil $\geq 90$ pour un lien automatique	Sensible aux transpositions (“ <i>Red Bag</i> ” vs “ <i>Bag Red</i> ”)
TF-IDF caractères + cosinus ( <code>scikit-learn</code> )	Robuste aux fautes, ordres de mots	Requiert corpus suffisant ; seuil à calibrer

**Règle de décision :**

$$\text{score}_{final} = \alpha \cdot \text{Levenshtein} + (1 - \alpha) \cdot \text{TF-IDF\_cosinus} \quad (\alpha = 0,5)$$

Si  $\text{score}_{final} \geq 0,80$  : lien automatique.

Si  $0,50 \leq \text{score}_{final} < 0,80$  : proposition à validation humaine.

Si  $\text{score}_{final} < 0,50$  : non apparié, envoyé en niveau 3.

*Niveau 3 – Semantic Match par embeddings (Sentence-BERT + pgvector)*

Pour les noms trop différents stylistiquement mais dénotant le même produit (ex. “*Coeur Blanc Lumineux Suspendu*” vs “*White Hanging Heart T-Light Holder*”), les embeddings sémantiques capturent le sens au-delà de la forme.

```
# Déjà implémenté dans src/agent/rag.py
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("all-MiniLM-L6-v2") # 384 dimensions

# Stockage dans PostgreSQL via pgvector
# CREATE EXTENSION IF NOT EXISTS vector;
# ALTER TABLE products ADD COLUMN embedding vector(384);
# CREATE INDEX ON products USING ivfflat (embedding vector_cosine_ops);
```

#### Niveau 4 – Validation humaine (stewardship)

Les cas ambigus ( $0,50 \leq \text{score} < 0,80$ ) sont soumis à un *Data Steward* via une interface de revue (tableau Kanban ou micro-tâche). Le retour humain alimente un ensemble d'entraînement pour améliorer les seuils en continu (*active learning*).

#### Gouvernance MDM en production

Dimension	Mesure
Couverture	% produits ERP ayant au moins un avis lié
Précision	% liens validés comme corrects par audit
Fraîcheur	Délai max entre mise à jour ERP et re-match
Traçabilité	Colonne MatchStrategy + horodatage

## Dictionnaire de données

L'entrepôt SmartShop 360 est hébergé dans PostgreSQL 16. Il suit un schéma en **étoile** : une table de faits de ventes, une table de faits d'avis, et trois dimensions (produit, client, mapping MDM).

Table source	Clé de liaison	Table cible
sales_facts	StockCode	product_mapping
review_facts	ProductID	product_mapping
sales_facts	CustomerID	customers
product_mapping	—	products

#### Table product\_mapping (Unified Product / Golden Record)

Table pivot centrale du MDM. Chaque ligne est un *Golden Record* qui relie un article ERP à son équivalent dans la source avis.

Colonne	Type SQL	Nullable	Description
MappingID	SERIAL	NON	Clé primaire auto-incrémentée
ERP_StockCode	VARCHAR(50)	OUI	Code article dans l'ERP (ex. 85123A)
ERP_ProductName	VARCHAR(255)	OUI	Libellé brut extrait du CSV ERP
Review_ProductCode	VARCHAR(50)	OUI	Code produit côté avis (ex. REV_001)
Review_ProductName	VARCHAR(255)	OUI	Libellé brut extrait du fichier JSON
Category	VARCHAR(100)	OUI	Catégorie consolidée (7 valeurs : Luminaires, Sacs & Accessoires, Art de la Table, Decoration Murale, Emballages & Cadeaux, Nature & Romantique, Divers)
GoldenRecord Name	VARCHAR(255)	OUI	Nom unifié retenu pour l'affichage (Golden Record)
MatchScore	FLOAT	OUI	Score de similarité cosinus TF-IDF $\in [0, 1]$ ; 0.0 si fallback rang
MatchStrategy	VARCHAR(20)	OUI	Stratégie utilisée : <code>tfidf</code> ou <code>rank</code>

**Clés étrangères (cible production) :**

- ERP\_StockCode → products.ProductID
- Review\_ProductCode → review\_facts.ProductID



**Table sales\_facts (Faits de Ventes)**

Granularité : **une ligne = une ligne de facture ERP** (article + quantité + date).  
 Chargée depuis le CSV *Online Retail II* (échantillon de 30 000 lignes, graine 42).

Colonne	Type SQL	Nullable	Description
FactID	SERIAL	NON	Clé primaire surrogate
InvoiceNo	VARCHAR(20)	OUI	Numéro de facture (ex. 536365) ; les codes commençant par C (annulations) sont exclus lors du nettoyage
StockCode	VARCHAR(50)	OUI	Code article ERP ; clé étrangère vers product_mapping.ERP_StockCode
Quantity	INTEGER	OUI	Quantité vendue (> 0 après nettoyage)
Revenue	NUMERIC(12,2)	OUI	Chiffre d'affaires = Quantity × UnitPrice
Margin	NUMERIC(12,2)	OUI	Marge simulée = Revenue × 0,35 (taux fixe POC)
InvoiceDate	TIMESTAMP	OUI	Date et heure de la facture
CustomerID	VARCHAR(50)	OUI	Identifiant client ; clé étrangère vers customers.ClientID

**Règles de nettoyage appliquées :**

- Suppression des factures annulées (InvoiceNo commence par 'C').
- Suppression des lignes avec Quantity ≤ 0 ou UnitPrice ≤ 0.
- Suppression des lignes avec CustomerID manquant.
- Calcul de Revenue et Margin post-nettoyage.

**Vues dérivées :**

- v\_product\_kpi – CA, marge, quantité vendue agrégés par produit.
- v\_customer\_kpi – CA total, panier moyen, nb commandes par client.

- `v_alerts` – statut CRITIQUE / A\_SURVEILLER / OK par produit (seuils :  
Note < 3,0 ET Qté > 50 = CRITIQUE; Note < 3,5 = A\_SURVEILLER).

### Table `review_facts` (Faits d'Avis Clients)

Granularité : **une ligne = un avis client**. Chargée depuis le JSON `labeledReview.datasetFix.json` (données réelles crawlées).

Colonne	Type SQL	Nullable	Description
ReviewID	SERIAL	NON	Clé primaire surrogate auto-incrémentée
ProductID	VARCHAR(50)	OUI	Code produit côté avis (REV_001...); clé étrangère vers product_mapping.Review_ProductCode
Rating	NUMERIC(3,1)	OUI	Note attribuée par le client $\in [1,0; 5,0]$ ; simulée depuis le champ Note du JSON (distribution normale)
ReviewText	TEXT	OUI	Texte intégral de l'avis client (utilisé pour les embeddings RAG)
Sentiment	VARCHAR(20)	OUI	Sentiment binaire : <b>positive</b> (champ <b>sentimen</b> = 1) ou <b>negative</b> (champ <b>sentimen</b> = 0); pas de neutre dans ce jeu de données
ReviewDate	TIMESTAMP	OUI	Date de l'avis; simulée sur 18 mois (2024-01-01 à 2025-06-30) avec graine 42

Table dérivée – review\_embeddings (hors DDL principal) :

Colonne	Type SQL	Description
id	SERIAL	Clé primaire
review_id	INTEGER	Référence vers <code>review_facts.ReviewID</code>
embedding	vector(384)	Vecteur d'embedding Sentence-BERT (all-MiniLM-L6-v2)

Indexé avec `ivfflat` (extension `pgvector`) pour la recherche ANN (*Approximate Nearest Neighbour*) dans le module RAG.

## Synthèse des choix techniques

Besoin	Choix retenu	Justification
Matching flou de noms	TF-IDF char n-grams + cosinus	Robustesse aux fautes, pas de dépendance externe
Matching sémantique	Sentence-BERT + pgvector	Capte la sémantique au-delà de la forme lexicale
Catégorisation produit	Règles mots-clés (7 catégories)	Reproductible, explicable, sans ML
Seuil de matching	0,35 (TF-IDF cosinus)	Calibré expérimentalement sur données UK retail
Stockage	PostgreSQL 16 + pgvector	SQL standard + indexation vectorielle native
Reproductibilité ETL	SHA-256 + <code>.etl_hashes.json</code>	Détection incrémentale des changements de source